1992

APPLICATION OF IMAGE ANALYSIS TECHNIQUES TO SATELLITE CLOUD MOTION TRACKING

LAU, KING SHING ALBERT

APPLICATION OF IMAGE ANALYSIS TECHNIQUES TO SATELLITE CLOUD MOTION TRACKING

K. S. A. LAU

Ph. D. 1992

Dedicated to my mother

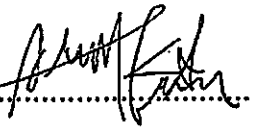
Wong How Chun

and to the memory of my father

Lau Wing Hee

1916—1992

Copyright

Signed: 

APPLICATION OF IMAGE ANALYSIS TECHNIQUES TO SATELLITE CLOUD MOTION TRACKING

KING SHING ALBERT LAU

School of Electronic, Communication & Electrical Eng.,
University of Plymouth, Plymouth, PL4 8AA, England.
In collaboration with the Meteorological Office, Bracknell.

A thesis submitted in partial fulfilment of the
requirements of the Council for National Academic Awards
for the degree of Doctor of Philosophy

May 1992

Abstract

Application of Image Analysis Techniques to Satellite Cloud Motion Tracking

Author: King Shing Albert Lau

Cloud motion wind (CMW) determination requires tracking of individual cloud targets. This is achieved by first clustering and then tracking each cloud cluster. Ideally, different cloud clusters correspond to different pressure levels. Two new clustering techniques have been developed for the identification of cloud types in multi-spectral satellite imagery.

The first technique is the Global-Local clustering algorithm. It is a cascade of a histogram clustering algorithm and a dynamic clustering algorithm. The histogram clustering algorithm divides the multi-spectral histogram into non-overlapped regions, and these regions are used to initialise the dynamic clustering algorithm. The dynamic clustering algorithm assumes clusters have a Gaussian distributed probability density function with different population size and variance.

The second technique uses graph theory to exploit the spatial information which is often ignored in per-pixel clustering. The algorithm is in two stages: spatial clustering and spectral clustering. The first stage extracts homogeneous objects in the image using a family of algorithms based on stepwise optimization. This family of algorithms can be further divided into two approaches: Top-down and Bottom-up. The second stage groups similar segments into clusters using a statistical hypothesis test on their similarities. The clusters generated are less noisy along class boundaries and are in hierarchical order. A criterion based on mutual information is derived to monitor the spatial clustering process and to suggest an optimal number of segments.

An automated cloud motion tracking program has been developed. Three images (each separated by 30 minutes) are used to track cloud motion and the middle image is clustered using Global-Local clustering prior to tracking. Compared with traditional methods based on raw images, it is found that separation of cloud types before cloud tracking can reduce the ambiguity due to multi-layers of cloud moving at different speeds and direction. Three matching techniques are used and their reliability compared. Target sizes ranging from 4 x 4 to 32 x 32 are tested and their errors compared. The optimum target size for first generation METEOSAT images has also been found.

Acknowledgements

I gratefully acknowledge my supervisors Dr J.G. Wade and Dr N.L.H. Wood for setting up this project, introduce me to clustering and many other helps throughout this study.

I should also like to thank my friends, Raymond Ho and Terry Cheung for typing some of the reference wind data, and Paul Davey for reading parts of my manuscript.

This work was funded by the Science and Engineering Research Council under grant number GR/E/74007.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Meteorological satellites provide a continuous observation of the globe. The imagery plays a vital part in modern forecasting practice, allowing the forecaster to observe directly both the movement and development of the individual weather systems. Its role may be regarded as complementary to that of the numerical models, providing the basis for more accurate analyses as well as more effective use of the numerical output (Woodroffe 1987). Many meteorological parameters can be extracted from satellite images, such as wind direction and speed from clouds, ice movement, atmospheric stability, relative humidity and precipitation from clouds, and turbulence from clouds etc.

1.1 Objective

Wind speed and direction are important parameters in the study of weather systems, and previous studies show that wind can be inferred from cloud motion. Cloud tracking requires identification of cloud type and cloud altitude. The aim of this study is to investigate and develop pattern recognition techniques to improve cloud motion wind (CMW) derivation. Clustering has been chosen for cloud separation since it requires minimum a priori information. Two clustering techniques are developed: the first algorithm partitions only the measurement space, whilst the second algorithm first partition the spatial space and then the measure-

ment space, thereby exploiting contextual information. The concept of tracking clouds of the same height is tested by applying the first algorithm in an automated wind tracking system and encouraging results are obtained.

1.2 Atmospheric Motion

The atmosphere is a gigantic heat engine in which the constantly maintained difference in temperature existing between the poles and the equator provides the energy supply necessary to drive the planetary atmospheric circulation. The conversion of the heat energy into kinetic energy to produce motion must involve rising and descending air, but vertical movements are generally much less in evidence than horizontal ones, which may cover vast areas and persist for periods of a few days to several months.

Atmospheric motion, either vertical or horizontal is caused by an imbalance of forces due to pressure difference. Vertical air motion while exceedingly significant for atmospheric processes is greatly limited by the shallow depth of the atmosphere, and the balance of the downward acting gravitational force of the earth and the vertical pressure gradient. While the horizontal motion with much less constraint of the gravitational force usually happens in large scale and this motion is termed wind.

Wind speed is generally lowest on the earth surface and gradually increases with altitude. This effect is due to surface friction reducing the rate of flow in the lowest layers of the atmosphere. At increasing heights above the surface, frictional effects become smaller, and the wind speeds generally increase in magnitude. A zone of maximum wind speed is frequently found near the tropopause ($\approx$10km height).

On the hemispheric scale, horizontal variations in pressure brought about by temperature differences sets air in motion. The rotation of the earth significantly modifies the direction of large-scale flow, but superimposed on the major wind patterns are smaller secondary disturbances induced by local variations in

temperature-pressure patterns. Motion plays a fundamental role in the transport of heat, moisture and mechanical energy from one part of the earth's surface to another.

In total there are four forces which determine the horizontal motion.

They are (Parikh 1976):

1. Pressure gradient; The pressure gradient is defined as the pressure difference between two isobars divided by the distance. The forces caused by a pressure gradient acts from high pressure to low pressure.

2. Coriolis acceleration; If the earth did not rotate warm air would rise near the equator and flow at a high altitude towards the pole where it would cool, sink and flow back towards the equator near the surface. However, the rotation of the Earth prevents winds from blowing directly northwards and southwards from the equator, instead they tend to be deflected sideways by the coriolis effect, so that most poleward flowing air is deflected towards the east whereas equator flowing air is directed towards the west.

3. Centrifugal acceleration; the centrifugal effect arises in conjunction with the Coriolis effect. It is an apparent restoring force opposing motion in a curved path by attempting to establish straight line flow. It is directed radially outward from the centre of curvature.

4. Friction forces; This forces always oppose the motion. It arises from contact resistance to relative motion between systems.

The resultant of these four forces determine the speed and direction of the horizontal motion. When friction is small, the motion is determine by pressure gradient, Coriolis force and centrifugal force. In areas where the motion is not turning, the effect of centrifugal force can be ignored. The resulting motion is termed geostrophic wind. The geostrophic wind balance is obtained when the wind is blowing parallel to the isobars. In the Northern Hemisphere the Coriolis force has an effect of turning a northward motion eastward. If the motion is turning

then the wind is non-geostrophic and is called gradient wind. The resultant wind blows in a counter-clockwise direction parallel to the curved isobars within a low pressure system. In the Northern Hemisphere this circulation is called cyclonic. Friction always exists at the earth's surface, here the wind will no longer flow parallel to the isobars, but slightly across the isobars. This creates convergence in a low pressure system.

Clouds are abundant in the atmosphere and useful wind vectors can be generated by tracking cloud targets whose motion seems in approximate agreement with the synoptic situation.

1.3 Cloud Identification and Characterisation from Satellite

Clouds are usually wet atmospheric aerosols composed of tiny spheres of liquid water ranging in radius from 2 to 200μm. Clouds form when air becomes supersaturated with respect to liquid water or ice; the most common means by which supersaturation occurs in the atmosphere is through the ascent of air parcels, which results in the expansion of the air and adiabatic cooling. Under these conditions, water vapour condenses onto some of the aerosol in the air to form a cloud of small water droplets.

The principle types of ascent, each of which produces distinctive cloud forms, are (Henderson-Sellers Ch.2 1984):

1. Local ascent of warm, buoyant air parcels in a conditionally unstable environment which produces convective clouds. These clouds have diameters ranging from about 0.1 to 10km. The lifetimes of convective clouds range from minutes to hours (cumulus, cumulonimbus).

2. Forced lifting of stable air which produces layer clouds. These clouds can occur at altitudes from ground level up to the tropopause and extend over areas of hundreds of thousands of square kilometres. Layer clouds gener-

ally exists over periods of tens of hours (stratus, cirrocumulus, altocumulus altostratus and stratocumulus). These two ascents account for most cloud types that are visible in satellite imagery and which are suitable for wind motion determination.

3. Lee wave, standing wave, not real motion.

Water by evaporation from the surface stays in the atmosphere as clouds, then followed by precipitation falls back to the surface, this endless cycle is responsible for the transportation of water around the Earth. Since clouds are usually advected by surrounding winds, tracking the cloud movement allows us to deduce the pattern of atmospheric motion.

Cirriform clouds are composed of ice crystals, altiform are supercooled water droplets existing at temperatures below 273K, while stratiform clouds are generally layered. Cumuliform clouds occur in unstable conditions. Nimbus clouds are rain or snow-producing. The World Meteorological Organisation (WMO) classify cloud into 10 types (Table 1.1and 1.2, International Cloud Altas 1957).

Level	Cloud	Appearance
High	Cirrus	Detached, fibres
High	Cirrostratus	Transparent sheet
High	Cirrocumulus	Small regular elements, no shading
Middle	Altostratus	Grey/bluish sheet, slightly transparent
Middle	Altocumulus	Layer with structure and shading
Low	Nimbostratus	Grey layer, precipitation
Low	Stratocumulus	Layer with structure and shading
Low	Stratus	Grey layer, uniform base
Cloud with vertical Development	Cumulus	Detached, fluffy
	Cumulonimbus	Heavy, dense, very tall

Table 1.1: Traditional cloud classes.

The height levels of these cloud genera are determined by their cloud base. The separation these cloud types are crucial to success of cloud wind generation (Parikh 1976).

Latitude			Cloud type
Polar	Temperate	Tropical	
3-8	5-18	6-18	Cirro—High
2-4	2-7	2-8	Alto—Middle
0-2	0-2	0-2	Strato—Low

Table 1.2: Cloud type and typical cloud base height (km) as a function of latitude.

1.3.1 Optical Properties of Clouds

Optical properties of clouds are reflectivity (visible band) and emissivity (infrared band), these depend on the physical properties of clouds. Cloud can exists as ice or supercooled water droplets. Water clouds have many small particles while ice clouds have relatively few but larger particles.

Both water and ice clouds have high reflectivities due to low emission/absorption at 0.5—1.3μm (visible band), and are good emitters; therefore the radiance emitted by them can be used to estimate the temperature of themselves by means of the Planck function (Henderson-Sellers Ch.2 1984). However, most cirrus type clouds are semi-transparent, therefore the energy sensed by a satellite is not a reliable indicator of the cirrus cloud top temperature, but it is the sum of the cloud and background radiance underneath it.

Property	Wavelength	Water cloud	Ice cloud
Reflectivity of sunlight when the Sun is overhead	0.7μm	0.8	0.8
Emissivity for thermal radiation	11.5μm	0.95	0.95

Table 1.3: Representative cloud response in atmospheric window. These number vary with cloud depth, width and particle size. Only typical values are given here (After Bunting and Hardy in Ch.6 in Henderson-Sellers 1984).

1.3.2 Interpretation of Cloudy Images

Cloud observations at high latitude from the satellite are degraded due to the Earth's curvature. Firstly, the degradation of the horizontal resolution of the imagery as a result of changing the nadir angle is accentuated. Secondly, the data requires corrections at oblique viewing angles for changing viewing geometry between the Sun, the cloud and the satellite. Thirdly, the satellite tends to see more cloud at oblique viewing angles since it looks through more atmosphere and therefore has a higher probability of encountering a cloud. Moreover, it may confuse the sides of a cloud with the top and also simultaneously see several types of cloud. Finally, most Earth location procedures assign a pixel to a location at the Earth's surface based on spherical trigonometry. A high cloud may be mislocated horizontally by twice its altitude if it is viewed at an angle of 60° off local vertical (Anderson and Veltishchev ed. Chap.1 1973).

The Meteorological satellite usually makes multi-spectral observations in order to distinguish different cloud types. The most effective wavebands are the visible (0.4—1.1μm) and infrared (10.5—12.5μm) channels. Due to different cloud thickness, background such as snow, ice, land surface, sea surface, which have different reflectivities at visible wavelengths and often have different temperature at thermal wavelengths; when using computer analysis, these varying backgrounds may be confused with clouds.

Meteorological phenomena can be roughly observed in two scales. The first scale is called *mesoscale* it refers to phenomena in small scale (usually an area less than 100 x 100km^2 but greater than 10 x 10km^2), for example a thunder storm. The second is called synoptic scale it refers to phenomena happening in large area, such as a weather front. One objective of this work is to derive mesoscale cloud motion wind.

Some useful features for interpreting weather images are *size, shape, tone*, and *texture* of individual clouds or cloudy regions. Image sequences also help to locate clouds in relation to other weather or geographical information. Detail analysis of satellite imagery can be found in Anderson and Veltishchev ed. (1973).

Cloud *size* varies considerably and gives important information to the image analyst. Individual cumulus clouds are normally too small to be resolved in the visible image. When the Cumulus clouds develop to large Cumulonimbus in thunderstorms, the contrast between the background and cloud increases and are evident in the visible and infrared images. Mid-latitude low pressure areas or cyclonic storms and the fronts extending from them often have cloudy areas exceeding 1000km. Meteorologists use these sizes to identify isolated clouds, mesoscale cloud lines, larger-scale fronts and storms.

Shape is used along with size as a means to characterise clouds. It is particularly useful for identifying layer cloud such as Stratus since its boundaries are sharply defined and the shape of a low Stratus area often outlines topography, such as coastlines, mountains and valleys. Cold air blowing over warm water often produces long lines of clouds known as cloud streets. Cirrostratus is normally smooth and uniform in appearance, thus may appear in the form of long bands extending for hundreds of kilometres, or as an extensive sheet. Jet stream Cirrus forms at high altitudes and has a characteristic shape on the poleward edge; this shape usually consists of a long smooth curve close to the maximum winds of the jet. Mature tropical storms called hurricanes or typhoons may appear nearly circular with a smaller circle or eye, lacking high clouds, in the centre. High clouds associated with cyclogenesis or storm formation at mid-latitudes may look first like an elongated leaf, smooth on the poleward size and ragged on the opposite, and later like a larger comma, the tail of the comma corresponding to clouds on a cold front. Frontal zones appear as long, multi-layered cloud bands extending for several thousand kilometres and range from one to several hundred kilometres in width.

The cloud *tone* represents how bright the cloud appears on the image. In visible channels, it relates to cloud reflectivity, brighter clouds being more reflective and thicker. In thermal channels, the grey level for cloud pictures are usually reversed so that bright tones represent low thermal energy of cold clouds and dark tones represent high thermal energy of warmer clouds or clear areas. Fig-

texture is useful for distinguishing cumuliform clouds from stratiform clouds. infrared images show fewer areas of rough texture even when the resolution is the same as a coincident visible image. The fields of thermal radiation emitted by clouds or the Earth's surface are smoother than the scattered sunlight at shorter wavelengths. Patchy cirrus clouds are an exception and often show rougher texture on infrared images.

1.4 Cloud and Wind Relationship

Wind can be derived from geostationary satellite images by measuring the displacements of cloud fields as displayed by a sequence of images (Izawa and Fujita 1969). This measurement is based on the assumption that the clouds move with the surrounding air parcel. In order to justify this assumption, we have to consider the following factors (Hubert 1979):

1. The nature of the cloud targets to be tracked in relation to the image resolution and to cloud target persistence as compared to image frequency.

2. The relation between cloud motion and wind.

Cloud targets which are tracked in the low troposphere are different from those of the upper troposphere because the mesoscale circulation systems that persists at low levels are not apparent in the upper troposphere. Therefore satellite wind is usually derived for low and high levels. Middle level clouds are usually difficult to track and are not good wind tracers (Parikh 1974).

Convective clouds provide most of the low tropospheric wind tracers because they are abundant and well suited to tracking. As with other cloud types convective clouds are not inert bodies, individual clouds form and disappear while the air parcels are carried along by wind. However, animated sequences taken at different time intervals reveal that mesoscale patterns of cloud such as trade cumulis, cumulus congestus, and even strato cumulus, have life times in the order of hours.

Many empirical studies have shown that the motion of the lower mesoscale patterns correlate well with air motion. An analysis by Hubert and Whitney (1971) showed that low level cloud vectors represented the 850mb flow with approximately the same degree of accuracy as rawinsonde data. Hasler et al. (1976,1977) conducted a multi-aircraft experiment to measure simultaneous cloud motions and winds at various levels and found that cumuli motions were correlated best with cloud-base winds, while cirrus appears to represent a layer mean. Similar studies with altocumulus have not been made, mostly because of the great difficulty in obtaining the necessary data.

In contrast, many upper clouds that are suitable for tracking appear to be layer clouds. At upper levels patches of layer cloud often change slowly—they may last many hours.

Comparisons of cloud motions in persistent features with nearby rawinsonde observations indicate that many of these persistent cloud patterns are embedded in layers with small vertical shear and they are advected with the layer wind.

For cloud wind to be useful it must be assigned to a level which best represents the wind, and vectors should be derived from targets which move with the environment wind. The first generation of meteorological satellites (early 1960's) only carried visible sensors, so no information about temperature was provided. If the cloud top temperature can be inferred then the vector can be assigned to a level which best represents the wind. Shenk and Kerins (1970) were among the first to investigate the use of an infrared channel for cloud top height estimation applied to cloud track wind.

Hubert and Whitney (1971) investigated the usability of cloud winds. They found that low and high cloud motions correspond best to winds at 3000 ft and 30000 ft respectively. The median vector deviation of the cloud velocities are 9 knots and 17 knots for low and high cloud winds. The deviations are due to

1. uncertainty of cloud height

2. non-advective cloud motion

3. photograph measurement errors (mapping error)

4. tracking errors

5. unrepresentative rawinsonde observations

The relationship between observed cloud motion and wind is strongly influenced by the type of cloud. Fujita and Pearl (1975) tracked single turret cumulus target and found that the best target size was 0.3 to 2 km, and their movement may not correspond to the environmental wind as a consequence of the complicated nature of clouds including the vertical wind shear, updrafts and downdrafts.

The quality of cloud wind was studied by Bauer (1976). He made a comparison of cloud motion winds with coninciding radiosonde winds which showed that both have a similar capability to represent atmospheric motions; the study showed that the differences between cloud motion winds and radiosonde winds fall within the limits determined by computational techniques, observational methods and atmospheric variability. Maddox and Vonder Haar (1979) developed a quantitative estimate of the random error inherent in satellite derived winds. They concluded that random error in vector winds derived from cumulus cloud tracking using high-frequency satellite data is less than 1.75 ms^{-1}.

Hasler et al. (1979) have presented high resolution aerial photography taken as frequently as one every 7 minutes which show that the lifetimes of individual cells are short, but that cumulus ensembles can maintain a recognisable pattern for well over an hour. Cirrus cloud lifetimes can often be resolved by the 30 minutes satellite observations. Orographic clouds tend to be stationary and clouds caused by gravity waves tend to move with the wave phase velocity and neither would be good estimators of the ambient wind.

The data used throughout this work was generated from METEOSAT. The METEOSAT images have resolution of 5km in the infrared channel and 2.5km or 5km in the visible channel at the sub satellite point and an image frequency of two per hour. The area covered by a single pixel can be much larger than some cloud types, therefore the identification of features of these cloud types would require at least a few pixels.

It is clear that satellite images do not display individual clouds, but rather patches of cloud. Furthermore, the target usually contains a much larger area than a few pixels, and so cloud motion is tracked by means of patterns rather than individual patches.

It is apparent that, at this resolution and frequency of present geostationary satellite imagery, large classes of cloud fields provide trackable targets whose motion are closely related to the winds.

1.5 Cloud Classification

One of the objectives of this study is to develop cloud classification algorithms for cloud motion wind generation, these algorithms belong to a field of study called statistical pattern recognition.

Pattern Recognition has been used to classify cloud images with variable degreesof success. Two approaches can be used (Duda and Hart 1973). The first approach is called supervised classification, it is based on Bayes decision theory. This approach is based on the assumption that the decision problem is posed in probabilistic terms, and that all of the relevant probability values are know. This implies that the class conditional probability density function (pdf) $p(\mathbf{x}|\omega_i)$ describing each class is known exactly. Here $\mathbf{x}$ is a pixel vector and ω_i denotes class i. Usually these pdfs are estimated from a set of training samples from known classes. The classification of a new pattern is achieved by a set of *decision rules* such that the error probability of classification is minimised.

A priori knowledge is difficult to obtain in some practical uses such as cloud classification. The cloud classes in visible and infrared imagery vary substantially during different times of day and season, if a supervised method is used a database of cloud signatures in different conditions must be build. The second approach called unsupervised classification requires a minimum of a priori knowledge and therefore was chosen for this study. Clustering is the most commonly used unsupervised method and if used correctly it can seek out the natural structure of the

data, and therefore the boundaries which separate the different classes. Partitioning of even a small set of data requires that an enormous number of combinations be tested, therefore a suboptimal method such as clustering must be employed. Clustering will be discussed in detail throughout the rest of this thesis.

1.6 METEOSAT images

METEOSAT images are used in this study for cloud motion wind generation. METEOSAT is a geostationary satellite with a nominal position on the Greenwich Meridian over the equator at an altitude of about 36000 km. The raw data obtained from the three-channel radiometer are visible and infrared radiances of the full Earth disk in the following bands (Fig. 1.2):

1. visible (0.4—1.1 μm) (VIS)

2. thermal infrared (10.5—12.5 μm) (IR)

3. infrared water vapour absorption (5.7—7.1 μm) (WV)

The visible band measures the reflected radiance of the electro-magnetic spectrum. The brightness of visible imagery is a measure of the Earth's albedo, generally displayed in picture form where white represents areas of high albedo and black represents areas of lowest reflectivity. (Reflectivity of an object is measured in albedo, which is defined as the ratio of the amount of electromagnetic radiation reflected by a body to the amount incident upon it, commonly expressed as a percent). The brightness of a cloud as seen from space depends upon the illumination of the cloud (sun angle), the angular position of the cloud in relation to the sensor and the sun, and the reflectivity of the cloud itself. Reflectivity, in turn, is related to cloud thickness, particle size distribution, particle composition (ice or water), and to the character of the upper cloud surface.

The thermal infrared band measures the long-wave radiation emitted by cloud, land and water surfaces. These measurements may be converted to temperature representative of the surfaces viewed. Radiation from these surfaces is transferred

to space through complex radiative transfer processes which involve absorption and re-emission by several atmospheric components. In infrared imagery darkest areas represent the warmest surfaces, while the brightest represent the coldest surfaces. They also shows relative cloud height.

The water vapour band measures radiation that is not controlled by absorption in the atmosphere, but by the temperature near the boundary between moist and dry air. The moister the air, the higher and colder this boundary, so the less radiation it is. On the image dark areas correspond to relatively large amounts of radiation (originating at low, warm altitudes) and brighter areas correspond to relatively small amounts of radiation (originating at high, cold altitudes).

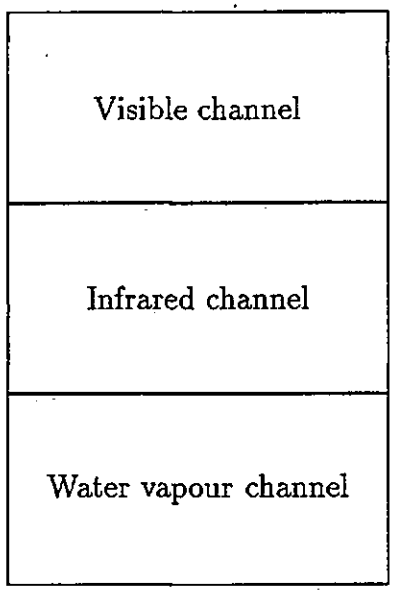


Figure 1.2: METEOSAT visible, infrared, and water vapour images.

The field of view of the visible detector is 2.5 km, at the subsatellite point (the point which has the shortest distance between the satellite and Earth), while for the other two infrared band detectors it is 5 km. The radiometer generates a new Earth image every half hour. The image is created in the direction from south to north and east to west by making use of the spin motion of the satellite and stepping the radiometer. An IR or WV image consists of 2500 lines by 2500 pixels, whilst each visible image consists of 2500 lines x 5000 pixels. There are two visible detectors which are offset by one pixel in the north-south direction, so if both visible channels are active the full visible image consists of 5000 lines x 5000 pixels. Normally the radiometer operates as (i) IR plus one VIS and (ii) IR plus one VIS plus WV. The VIS images transmitted are usually with half resolution, i.e. it is also 2500 lines x 2500 pixels the same as IR and WV images. In the operational timetable WV is only available every 60 minutes, instead of 30 minutes as for VIS and IR. VIS is only available when the sun illumination is reasonable, however IR & WV can be generated even at night-time. Figure 1.3 shows the relationship of resolution with earth surface. Due to the surface curvature the resolution decreases rapidly away from the subsatellite point (ssp).

There are in total five geostationary weather satellites covering the majority of the earth surface. Table 1.4 shown a comparison of these satellites. Figure 1.3 shows the distribution of the satellites.

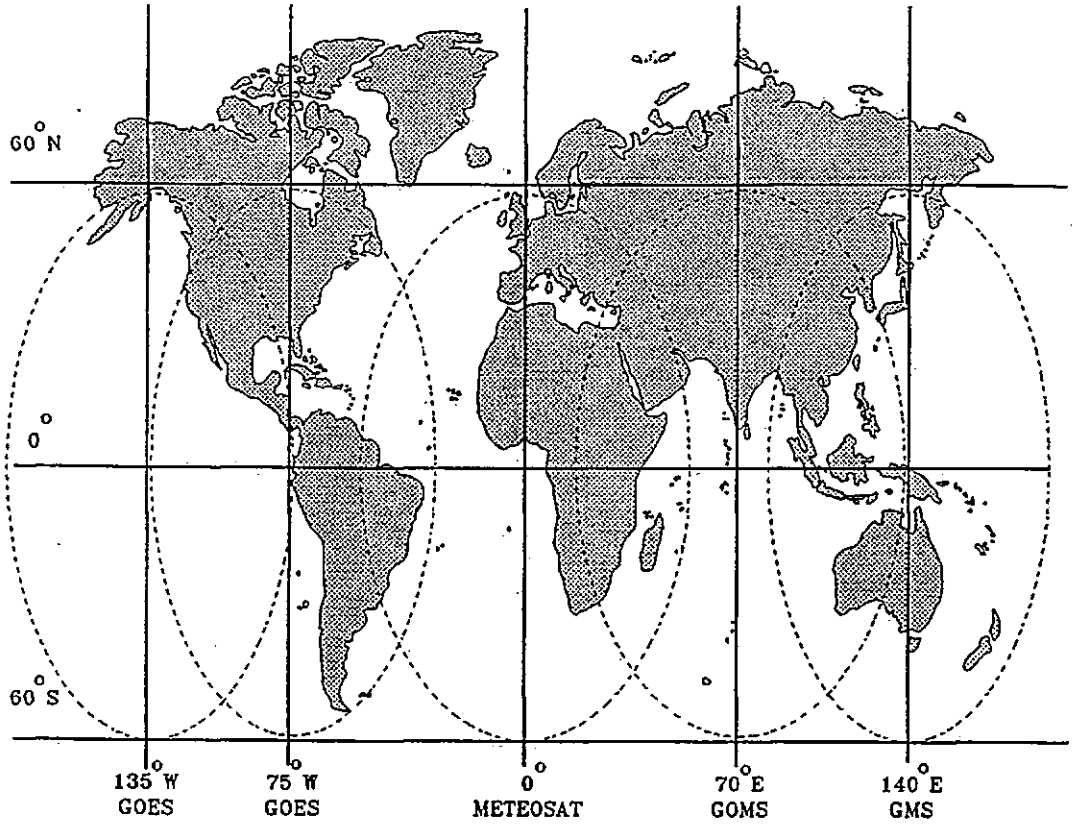


Figure 1.3: Fields of view of five geostationary satellites (Hubert 1979).

Satellite	Spectral bands	Resolution (at ssp)	Imaging period period
GOES (USA)	visible infrared	1 x 1 km 8 x 4 km	1/2 h
METEOSAT (ESA)	visible infrared water vapour	2.5 km 5.0 km 5.0 km	1/2 h
GMS (JAPAN)	visible infrared	1.25 km 5.0 km	1/2 h
GOMS (USSR)	visible infrared	1.5 km 12 km	1/2 h

Table 1.4: Comparative features of images from four types of geostationary weather satellites.

It is noted that Second Generation Meteosat Operational Programme (MOP) satellite in the late 1998 will provide image data in all spectral channels simultaneously (VIS, IR and WV), the high resolution visible (HRVIS) will have a sampling distance of 1km at nadir and image repeat time will be 15 minutes.

1.7 Chapter Outlines

The main objective of this study is to develop cloud classification algorithms and investigate their application to cloud tracking. The next chapter reviews different cloud identification algorithms and their applications. The problems associated with each algorithm are highlighted and their application discussed. Cloud wind systems will also be reviewed, some of them are for research purposes whilst others are operational.

Chapter 3 is an overview of various techniques related to clustering. Clustering is usually referred to as unsupervised learning in statistical pattern recognition literature. It is closely related to supervised learning in which many techniques can be extended and applied to clustering. Techniques related to the algorithms developed in this study will be introduced and some problems are highlighted. A breakdown of fields of study in pattern recognition is shown in Fig. 1.4.

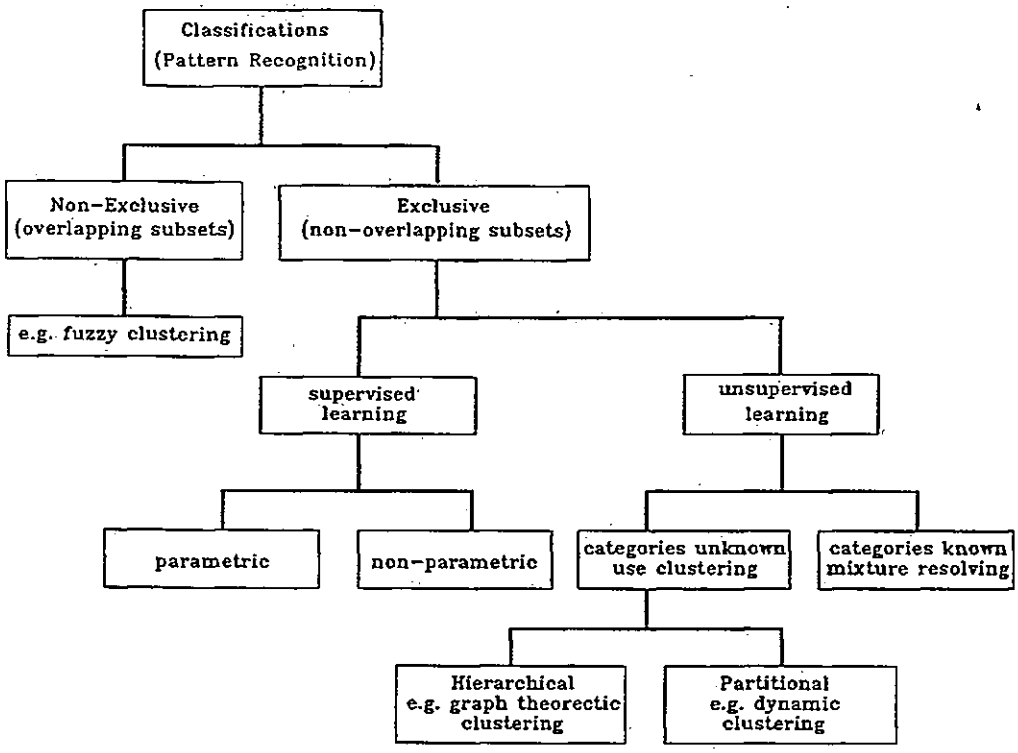


Figure 1.4: Break down of statistical pattern recognition.

A major problem in iterative clustering is the need for a set of starting centres, and these are usually selected from the image manually. This procedure is thought to be subjective and can have a profound effect on the results as shown in this work. Chapter 4 describes a Global-Local clustering scheme which partially solves this problem. The first stage of the clustering scheme consists of a non-parametric clustering algorithm which is used to obtained an initial partition of the data without any a priori knowledge of the data, however due to the nature of METEOSAT data this initial partition is usually unacceptable and therefore a second stage consists of a dynamic clustering algorithm is used to refine the partition. Several distance measures derived from a general Gaussian model are also presented. The effectiveness of the Global-Local algorithm is demonstrated using several sets of METEOSAT images.

Clustering of image data is computation intensive, the Global-Local clustering algorithm exploits the multiple occurrence of patterns, and hence is very efficient.

Spatial information is usually ignored by clustering algorithms. In Chapter 5 a Spatial-Spectral clustering algorithm is introduced. The first stage consists of a segmentation technique called Graph Theoretic Hierarchical Segmentation method which exploits the contextual information of image data. The second stage is the clustering of segments generated in the first stage. Segmentation is regarded as spatial clustering, based on this concept two stepwise optimal segmentation approaches are presented, they are the top-down and bottom-up approaches respectively. A cost function is evaluated in each step such that a criterion function is minimized, and cost functions are classified into global and local types, and it is shown that performance of global type cost functions are superior to local type functions. Segmentation results are always justified subjectively, in this study the performance of a segmentation process is monitored using the mutual information measure. Segmentation is modelled as a communication channel and the entropy loss of the system reflects how well the segments approximate the original image. The optimum number of segments are then clustered using an agglomerative method, this uses a statistical hypothesis testing as a similarity measure of segments, and the most similar pair of segments are merged in each step. The complete process integrates both contextual and spectral information and therefore generates "clean" clusters. The Spatial-Spectral clustering algorithm is compared with the Global-Local algorithms (per-pixel), and problems of this algorithm are also discussed.

Procedures to compute cloud winds is discussed in Chapter 6. They included geometric rectification of images, choice of tracking algorithms, target size, height assignment of wind vectors and target selection. The Global-Local clustering algorithm is included in an automated cloud wind generation scheme which compares traditional methods of cloud tracking using raw images and the clustering approach employed in this study. Six sets of METEOSAT images will be used and the result compared with numerical model data supplied by Meteorological Office Bracknell. An encouraging improvement using a clustering approach has been achieved. The optimal target size which yields minimum low level wind speed

error has been found to be 24 x 24 pixel at mid-latitude for METEOSAT imagery.

Finally, Chapter 7 summaries the results and highlights the advantages and disadvantages of the different techniques.

Chapter 2

Review of Cloud Analysis and Cloud Wind Systems

This chapter reviews various methods for cloud identification and cloud motion wind (CMW) systems. Satellite wind generation has been studied since the mid 1960's, and many wind systems have been developed. However, major problems still exist. Many cloud algorithms have been developed but most of them are designed for purposes other than cloud wind tracking. In general, all cloud identification algorithms are concerned with the classification of pixels in an image. Different algorithms will be discussed and we shall explain why clustering is chosen for cloud tracking in this study.

2.1 Cloud Identification Algorithms

Most of the Earth's surface is constantly covered by clouds, ice and snow, these substances dominate the reflectivity of the planet. The height and amount of cloud is an important mechanism in the control of infrared radiation emitted to space. It is clear that clouds play an important role in the mechanism that controls the planet's climate. The International Satellite Cloud Climatology Project (ISCCP) was set up in 1981 as part of the world climate research program (Rossow et al. 1985). The objectives of this project are to generate a representative data

base of satellite data and extract from this the cloud information appropriate for atmospheric modelling and climate studies. The huge number of images received from current satellite systems demand an automated technique for evaluating the cloud parameter. The objective of cloud analysis is to deduce properties such as cloud height, description, amount and type. A good cloud algorithm should be able to deduce all these features of the image. However, since many different forms of cloud exist, and because, the background can confuse observation of some cloud types (cirriform), this task can be extremely difficult. Many algorithms are heuristic and usually designed specifically for a single cloud property.

Many cloud analysis algorithms have been developed specifically for cloud cover estimation, since cloud cover is the most important parameter affecting the energy budget of the earth-atmosphere system. A small consistent perturbation in cloud cover may cause a significant change in climate. On the other hand, cloud signatures reflect the nature of atmosphere circulation and is important for the study of climate variations.

Cloud identification algorithms are used to estimate cloud cover. Satellite-based cloud identification algorithms can be grouped into three classes. 1) Threshold methods, 2) statistical procedures, and 3) radiative transfer techniques (Goodmane and Henderson-Sellers 1988).

Threshold techniques can use only a single visible channel, an infrared channel or both (Rossow et al. 1985). They assign to each pixel (or field of view) a completely clear or cloudy label according to the magnitude of the observed radiance or albedo relative to the predetermined threshold level. VIS thresholds are intended to represent the apparent temperature. The areas in the infrared images which are colder than the threshold are assumed to be clouds. There are four main methods for choosing a threshold. Firstly, the threshold for both VIS and IR are constant. Secondly, they can be derived from other weather or geographical data bases: infrared thresholds can be derived from surface reports of temperature and visible thresholds can be derived from the type of surface which is viewed. Thirdly, thresholds can be derived from the image itself if the area includes clear

areas. Clustering or Spatial Coherence methods (to be described later) can help to establish the threshold. Finally, to ensure that an area is partly clear so that the threshold represents 'clear/cloud' boundaries and not 'cloud layer' boundaries, the threshold can be determined from a series of images at different times and the extreme radiance (dark visible or warm infra-red) can be used as the threshold. Unfortunately, these techniques perform badly when pixels are partially or semi-transparent clouded. It is also difficult to define the clear sky radiance value.

In contrast, statistical techniques partition the multi-dimensional histogram into representative classes. They assume that every class has a distinct mode in the multi-spectral histogram (usually visible and infra-red). These classes are associated with relatively homogeneous emitting and reflecting surfaces, cloud types, oceans and land. There are three type of approaches based on the concept of multi-dimensional histograms, they are:

1. Gaussian histogram analysis (Platt 1983, Phulpin et al. 1983).

2. Dynamic Clustering (Desbois et al. 1982).

3. Spatial Coherence method (Coakley and Bretherton 1982).

The first method fits a Gaussian (normal) distribution function to one and two dimensional frequency histograms in order to isolate distinct clusters (see Fig. 2.1). The success of this technique depends on the decision criterion and the resulting effectiveness of the cluster definition. The European Space Agency (ESA) uses a histogram analysis of this type and applies it to 32 x 32 pixel visible and infrared window. A cluster is then identified in the histogram and classified (Bowen et al. 1979).

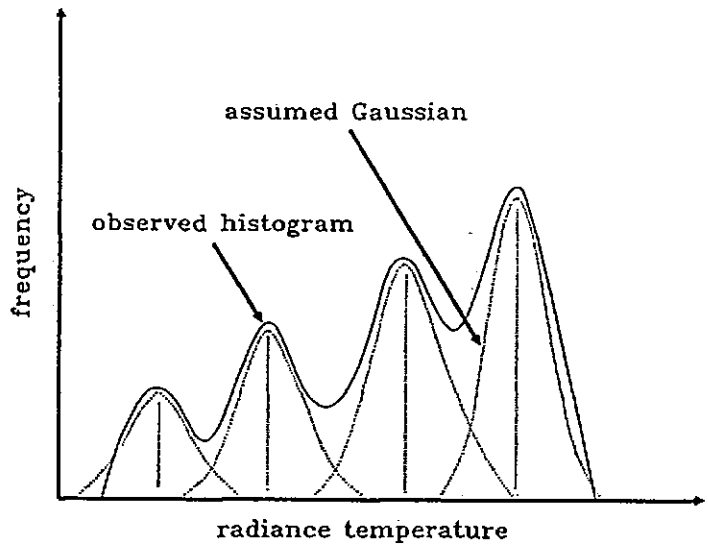


Figure 2.1: Fitting of one-dimensional Gaussian model onto a histogram.

The second method uses a clustering algorithm to indirectly partition the histogram. The maximum number of clusters to be found may be limited but the algorithms are otherwise free to find as many clusters as the image suggests. This is important because it distinguishes the clustering algorithms from classification algorithms, which use a previously determined set of clear and cloudy categories as a fixed set of choices. Clustering algorithms are known as unsupervised procedures that produce unlabelled categories. They are very flexible and useful for coping with a great variety of cloud types and backgrounds found in meteorological images. A drawback of clustering algorithms for cloud cover estimation is the inability to treat partially cloudy pixels correctly. Other limitations such as bad performance when clusters are not well defined in the histogram space. The initial centres also have a profound effect upon the number of resulting classes.

The last method called Spatial Coherence algorithms. This method studies the local spatial variance of a 3 x 3 pixel array and it relies on the assumption that over small horizontal distances the sea surface emitted radiance values from each pixel will be virtually constant (see Fig. 2.2). The emitted radiance values

of cloud tops are assumed to be variable from pixel to pixel. Over cloud-free sea surface the local standard deviation in infrared is found to be small, whereas for cloud contaminated pixels it is normally much higher. All local arrays with standard deviation higher than a predefined threshold value are assigned as cloud contaminated. The local standard deviation of 11 μm brightness temperature is plotted as a function of local mean radiating temperature. Function values with high standard variation and medium mean temperature represent overlap area in the histogram and hence partially cloud covered class. This spatial properties can be considered as a feature and be included in clustering to improve cloud boundary classification (Seze and Desbois 1987).

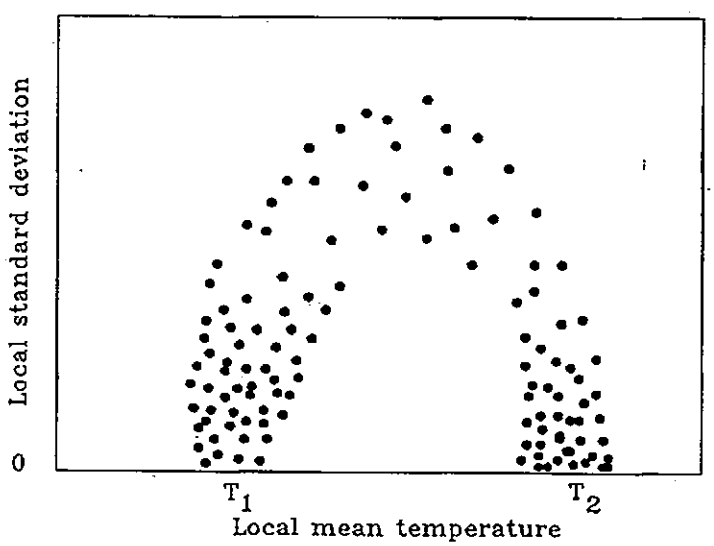


Figure 2.2: Typical plot obtain using spatial coherence method. Cluster of points at T_1 represents cloud-free scan spots, the cluster near T_2 represents cloud-covered scan spots. The points between these cluster represents partially covered field of view.

The last type of algorithm is called the radiative transfer type. This algorithm uses a cloud radiative model to simulate what the satellite would sense and what cloud properties could be retrieved if the data in different atmospheric windows

were available. The calculated radiances and corresponding cloud properties can be used in a look-up table so that measured radiances can be related to cloud properties.

The above techniques can be combined to complement each other advantages and disadvantages. Saunders and Kriebel (1988) used five tests to detect clear sky and cloudy radiances on AVHRR data. The first test applied to both daytime and night-time data is an infrared (12μm) threshold test as a check on cloud contamination. The second test is a local uniformity or spatial coherence test applied on a 3 x 3 pixel array of 11μm brightness temperatures. This test is applied over the sea during the day, as the horizontal temperature variations over cloud-free land can be significant, so this test is only suitable for small temperature variation surfaces. The third test applied during the day is a dynamic reflectance threshold test, the threshold is determined by examining a histogram of a 50 x 50 pixel window of the visible and infrared images. The fourth test used during the day, uses the ratio of near infrared reflectances to visible reflectances, this ratio is close to unity over clouds. The final test applied to both day and night time data examines the difference between 11μm and 12μm brightness temperatures. This cloud detection scheme was compared with Pairman's (1986) clustering algorithm, although no solid conclusion can be drawn due to difficulties in comparison, both schemes are useful for detection of clear sky radiance.

2.2 Supervised Cloud Classification

The main application of a cloud identification algorithm is to provide a cloud cover estimation, and so it is not directly related to the recognition of cloud types. But cloud identification algorithms such as those using clustering or histogram fitting produce unlabelled cloud types as a by product. There are many algorithms designed specifically for cloud classification, and these can have other meteorological applications, such as cloud cover estimation. Another possible application is the identification of cloud targets for cloud tracking, as suggested by Parikh (1977).

Most cloud classifications algorithms are statistical pattern recognition techniques and fall under the heading of supervised classification. Basically, the supervised technique uses a set of labelled samples to represent the typical pattern classes, then model parameters (usually Gaussian probability distribution function) are derived from these samples and hence the decision boundaries for different classes. Therefore classes have to be provided to the algorithm initially, and this is very difficult because images taken at different times of day and seasons can vary greatly due to different sun-satellite angles (although a data base can be built to compensate for these).

To avoid the problem of building a data base, an unsupervised classification approach seems highly desirable. One class of unsupervised classification is called clustering. Basically, clustering tries to partition the data set into its natural pattern, with little or no a priori knowledge of the data. A disadvantage of clustering is that spatial information is difficult to apply and therefore is often ignored. However, in Chapter 5 we will describe a new clustering algorithm which incorporates the use of spatial information.

A common type of cloud classification breaks the satellite image into small regions and classifies the cloud class of each region as a whole. The method may use many statistical and textural features to help classify each region more accurately. Due to the coarse resolution it is suitable for classifying large areas.

A study using the region approach was done by Parikh (1977), which compared the value of different features in classifying clouds. Cloud was classified into four types, low, mix, cirrus and cumulonimbus. The original application of this study is cloud motion wind determination. It was found that the visible and infrared images are good features for cloud classification. Parikh and Rosenfeld (1978) use a mixture of segmentation, thresholding and classification techniques to classify cloud types. The infrared image is first thresholded to separate cloud and background, and then the cloudy pixels are segmented using a clustering technique. Finally the clusters are classified using decision tree classifier based on the statistical features derived from each cluster.

A region approach was adopted by Garand (1988), Ebert (1987, 1989). Ebert used a maximum likelihood technique to classify four channels of Advance Very High Resolution Radiometer (AVHRR) data into eighteen cloud classes. Spectral and textural features characterizing each class is extracted from each 32 x 32 pixel cell. Eight features were chosen out of a total of sixty six features. Among them three are spectral features and the others are textural features. An iterative training procedure is used to reduce subjectiveness of initial training set. A classification accuracy of 85% on 25 classes was reported on a data set taken over the polar region.

Garand also used a maximum likelihood technique to classify bispectral GOES data into twenty cloud classes. The features used were slightly different from the usual spectral and textural features. Features such as cloud fraction at low, middle and high altitude, cloud top height and mean cloud albedo, multi-layer index, a streakiness factor connectivity indices and power spectrum were used. The use of two dimensional features can detect cloud streaks and roll. An accuracy of 79% was reported.

Despite all the difficulties of coping with the great variation of the data, an operational cloud classification model has been implemented in Sweden (Karlsson 1989). A data base of Sun elevations and air mass temperatures was built to allow diurnal and seasonal variations of radiances. The variation of data is mainly due to the following factors:

1. sun elevation at the object,

2. sun-satellite azimuth angle (i.e. the azimuth angle between the satellite viewing point and the sun measured at the object),

3. the shape of the object (e.g. if the object produces shadows, if it is transparent or if it is only partly filling the pixel),

4. differences in radiometer performance between different satellite and consecutive orbits,

5. variable radiance contributions from intercepting atmospheric water vapour and aerosols,

6. temperature variations of the object.

Karlsson's classification model included 828 class categories with 3 levels:

1. three season classes (summer, spring/autumn and winter),

2. twelve sun elevation classes,

3. twenty three object classes.

The classifier was based on maximum likelihood classification.

Another approach to classification is to segment the image based on the features being used, and then to label each of the regions or segments afterwards. Many more measurements are then derived to make a decision when labelling each of the resulting segments. Measurements such as the object shape can also be used. These are not available when individual pixels or arbitrary defined regions are being labelled. Studies by Seddon (1983) use cluster analysis to partition multi-spectral images. It should be noticed that the clustering technique assigns pixels to a cluster on an individual basis, so like the per-pixel classification, noise problems will be encountered at the boundaries between classes. Seddon used a heuristic post processing algorithm to clean up the regions before making shape measurements. Many spectral, textural and shape features were derived and the clusters were classified using decision tree classifier.

Algorithms which assign pixels individually tend to produce a rather noisy boundary between areas belonging to different classes. Studies by Kittler and Pairman (1985b) consider the use of contextual information in per-pixel classification. They argued that if the classification is a preprocessing step for a pattern recognition process, then the effect of noisy edges is undesirable for subsequent shape analysis of the classes. For instance, the result of the classification may be interpreted by a syntactic method to identify entities such as fronts, cyclones or jet streams. To do this the shapes of the objects would be examined. The

existence of many extraneous incorrectly classified pixels will make the task much more difficult. Their iterative contextual algorithm considered the classification of the neighbouring pixel, the classification rule was based on modified version of the Bayes' decision rule.

2.3 Unsupervised Cloud Classification

An area with a more direct relevance to the work described in the next three chapters is that of unsupervised classification or cluster analysis. This approach has been used in deriving cloud properties as well as cloud classification. Desbois et al.(1982) used dynamic clustering to cluster three channel Meteosat images (visible, infrared and water vapour). Dynamic clustering is a type of iterative process which tries to optimise an objective function representing the partition. The most common type of objective function is the mean square error within each cluster. Well defined types of cloud can be associated with a specified spectral signature in the multi-spectral histogram, therefore assuming every kind of cloud is represented in the spectral space by a compact cluster, decision boundaries can be drawn between different cloud types. Desbois find 5—7 clusters in a 200 x 200 pixel window. The clustering result was applied to determine the top temperature of semi-transparent cloud.

Seddon and Hunt (1985) and Pairman and Kittler (1986) also applied clustering to cloud images, and they used a similar clustering algorithm to that of Desbois. Seddon and Hunt included split and merge functions to allow clusters with too large a standard deviation to split, or clusters which are very similar to merge as in the classical ISODATA algorithm (Ball and Hall 1967). Seddon also used a linear transform (principle component analysis) to preprocess the image, and obtained better partitioning. Pairman used a modified distance metric derived from a normal distribution model. The metric allowed generalised Gaussian clusters and also account for different cluster population. Their algorithms required an initial partition to start. This initial partition can be generated randomly, but

random starting points can have a profound effect on the final result and therefore is not recommended (Seddon 1985, Kittler and Pairman 1985b). An algorithm to obtain good starting points will be discussed in Chapter 4.

Clustering produces decision boundaries that are fixed in spectral space, i.e. every pixel can only belong to one cluster. Key et al. (1989) used a new clustering approach called fuzzy clustering applied to cloud images. The fuzzy clustering algorithm assigns each observation to all clusters, with membership values as a function of distance to the cluster centre. The fuzzy set provides information on which spectral channels are best suited to the classification of particular features and can help determine likely areas (cloud boundaries) of misclassification. The ability to assign a pixel to more than one class may help to identify cloud contaminated pixels.

A supervised approach requires many resources to collect and verify a cloud data base which is not always a feasible approach. A quick start can be accomplished by using clustering, which requires little or no prior information and should perform well for all situations. Desbois (1982) found that clustering can compensate for images taken with different sun angle.

2.4 Operational and Research Wind Systems

All satellite cloud motion wind systems consist of the following process: image referencing, segment processing (target selection), wind vector determination, height attribution, manual editing, final processing (Hubert 1979).

Most early operational wind systems only generated low level winds. Green et al. (1975) described an operational model for SMS-1 low level winds. Since cross correlation is computationally expensive, they used a first guess/fast displacement algorithm to reduce the search area. They also used temperature slicing of infrared images to compute wind vectors, and then the output from the automatic procedure was presented to meteorologists for manual editing. A significant proportion of the automatic wind vectors which are rejected during manual editing represent

multi-layered or upper-level clouds.

Novak and Young (1977, 1978) gave a review of operating cloud motion system. An automatic method was used to produce only low level wind, and a manual method was used for both low and high level wind. After 1974 meteorological satellites carried infrared detectors, and this allowed cloud height to be estimated; however only low level (700~900mb) winds were computed operationally. They used temperature slicing for cross correlation. Manual analysis techniques provide the final quality control of the wind estimates produced by the automated and manual procedures.

An interesting wind system was presented Endlich et al. (1971). Here a clustering algorithm called ISODATA was used to separate cloud clusters in visible images and each cluster was represented by the centre of gravity. These centres were tracked in the image and the displacement of the centre represented the cloud motion. This clustering algorithm was found to be too time consuming and was modified by Wolf et al. (1977). The modified automatic system for cloud wind used 4km x 4km SMS-GOES images. They used a 21 x 21 pixels window and only retain pixels which are above the mean gray level of the window. These pixels are supposed to be all cloudy pixels, which are then grouped together as if they are spatially connected. Very large groups were reduced and very small groups were dropped. Two consecutive images were processed as mention above, then group centres are matched with heuristic procedures, these matches similarities produce the displacement of cloud targets. The process uses both VIS and IR information, but not simultaneously.

The operational wind system (Fig. 2.3) of the ESOC was describe by Bowen et al. (1979). This system is designed for extraction of meteorological information from METEOSAT images. The system is highly automatic and required little human intervention, although manual editing of the final result was necessary. This system also allowed manual tracking using a technique similar to a movie loop, but using cross correlation for interactive tracking. The whole image is divided into 32 x 32 pixel segments, and a cloud analysis using VIS+IR, and/or

IR+WV bi-dimensional histogram is applied to each segment. The Gaussian model is being fitted onto the IR histogram and then each fitting is regarded as a cluster. The mean radiance of each cluster in the histogram is compared with the predicted radiance and classified. If a suitable target is identified, it was tracked in the previous and next infrared image using cross correlation, therefore two vectors were obtained. If the two vectors differ by more than a threshold, then the target was assumed to produce spurious wind and be rejected. If a vector passed the test, another test was used to determined which cloud type in the segment it belongs to. This test performs cross correlation of every cloud type found in the previous histogram analysis, and the cloud type having maximum correlation was assigned to the wind vector.

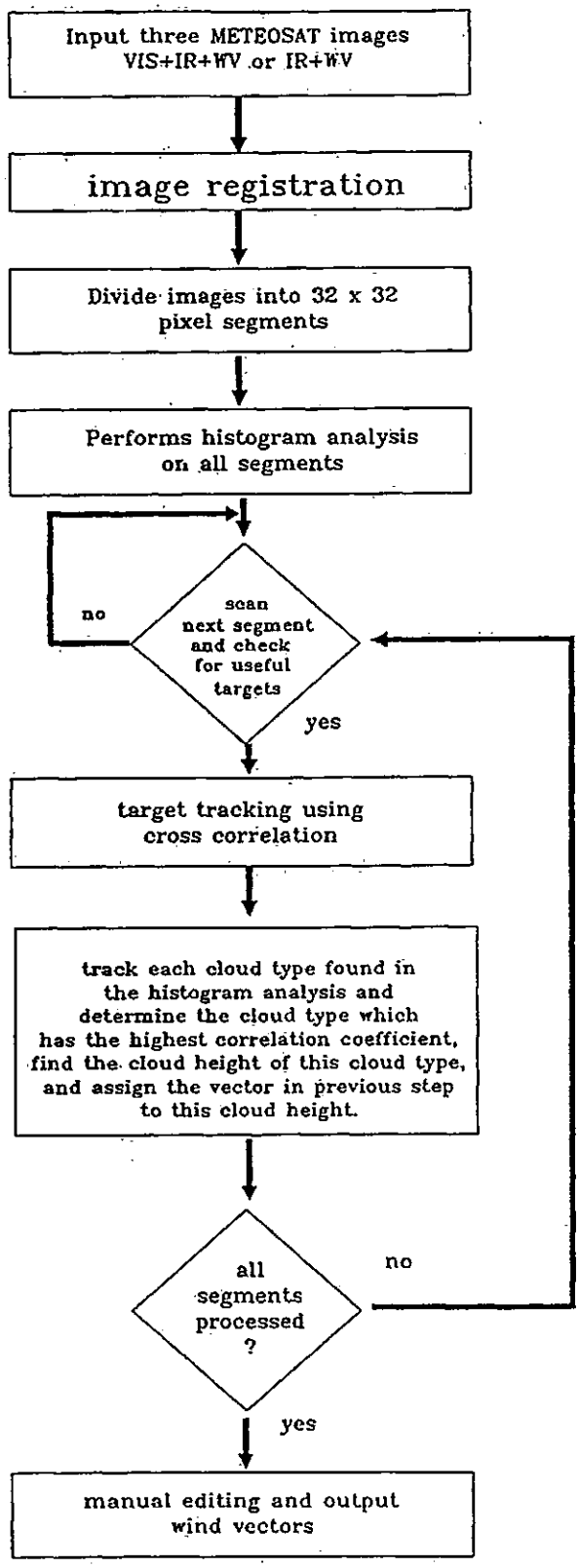


Figure 2.3: Operational cloud motion vectors system operated by Meteorological Information Extraction Center in European Space Operations Center.

Wilson (1984) described a new multi-spectral image processing system for extracting mesoscale wind fields automatically from sequences of GOES imagery. The system include a preprocessing stage, which performs the following image processing functions

1. clean up image to reduce random missing elements and lines, using nearest neighbour averaging.
2. contrast stretch to expand the image count value to the full 0-255 dynamic pixel resolution.
3. alignment of the visible/infrared scenes to account for element shift offset.
4. resample the infrared imagery to an equivalent resolution of the visible scene.
5. edge preserving filtering of the visible image.

The targets are selected by examining the visible and infrared greyscales, if the visible variance is less than a threshold T_{VIS} and infra-red variance is greater than T_{IR} then the window is assumed to contain a suitable cloud target. Wilson applied a tracking techniques called Sequential Similarity Detection Algorithm (SSDA) which was proposed by Barnea and Silverman (1972). A performance index was computed for every correlation surface and wind vectors with performance index lower than a threshold were rejected. He claimed less than 1% of wind vectors require meteorological editing.

Turner and Warren (1989) described two schemes for obtaining cloud motion vectors in the polar region using AVHRR data. The manual scheme used a movie loop to assist target selection and used cross correlation for tracking. The automatic scheme use three consecutive polar stereographic projected infrared images with a target window of 7 x 7, very flat featureless area were not selected because of difficulty in separating cloud and ice. The target and search area must have a mean temperature variation less than $5^{o}K$. Cross correlation was used and several threshold methods were used to speed up the computation. Two vectors were computed for each window and if either the speed or direction differ by more than a threshold

of 50% or 30° they are rejected. The height assignment use infra-red brightness temperature, no atmospheric correction or semi-transparency correction was applied. As in this thesis, their results were compared with numerical model results from the Meteorological Office.

Lunnon and Lowe (1990) investigated the relationship between template geometry and rms vector error using template size ranging from 4 x 4 to 32 x 32 pixels (METEOSAT infrared images) and found a minimum low level vector difference with a template size of 16 x 16 pixels. They followed the operational cloud wind extraction procedure of the European Space Operation Centre (ESOC).

Schmetz and Holmlund (1990) described the up to date version the of operational cloud motion wind scheme operated by ESOC. The wind extraction works fully automatically and is following by a manual quality control. The main features of the system remain unchanged as described by Bowen et al. (1979). The changes were

1. a radiance slicing technique for high level clouds was introduced alleviating the problem of tracking a mixture of clouds at various levels.

2. a new calibration of the METEOSAT water vapour channel based on radiative transfer calculations gave considerably higher calibration coefficient, which has lead to a better height assignment of semi-transparent clouds forming the major share of high level cloud tracers.

3. use a wind forecast to guide the cross correlation search.

4. the radiance slicing was replaced by a preprocessing and image transformation which extracts pixels belonging to the highest cloud layer in a segment area and smooths contaminated pixel values and the background.

In spite of these improvements, high level wind speed is still systematically underestimated. Bowen (1979) also suggests that satellite wind extraction using cirrus as tracer should be regarded as the mean wind of a deep layer instead of a specific level.

2.4.1 Other Cloud Wind Applications

Satellite cloud images can also be used to obtain mesoscale wind vectors. Wilson and Houghton (1979) computed 3 dimensional wind fields for a severe storm situation determined from SMS cloud images. He used the leading edge of large trade wind cumulus as tracers, and assign all winds to cloud top level. Their results show that satellite cloud wind data provided reasonable and useful information for mesoscale wind fields.

Although the infrared image is used for tracking low and high level wind, middle level wind can be obtained by tracking features in the WV channel. Endlich et al. (1981) tracked METEOSAT WV images using the wind system described by Wolf et al. (1977). However due to the flatness of the WV image, high vector density can only be obtained in and around regions of active weather phenomena. Eigenwilling and Fischer (1982) used cross correlation on METEOSAT water vapour image. The WV image was preprocessed by high pass filtering using a gradient filter. They obtained results applicable to mid-troposphere level (500 mb), and also found that WV features always can last as long as 10 hours.

2.5 Summary

There are several basic difficulties that have been encountered in automatic cloud motion tracking. The most important problems are

1. Height assignment of wind vector.

2. Target selection.

3. Multi-layer cloud confuse cross correlation tracking.

Each of the above problems can be treated as a separate research topic. In this study the primary objective is to improve the ability of cloud tracking in multi-layered regions. The necessity to separate cloud types before tracking is well recognised and has been done using simple temperature slicing (Green et al.,

1975) and by more sophisticated techniques such as histogram analysis (Bowen et al., 1979). The temperature slicing techniques does not use the visible information. While the histogram technique only uses histogram analysis for target detection, no attempt was make to track individual cloud type. Although clustering was tried by Endlich et al. (1971), it did not use both the VIS and IR information and their concept of tracking cluster centre is different from our concept of tracking the cluster pattern. In Chapter 1 it was mentioned that effective cloud analysis can only be made if both VIS and IR images are analysed simultaneously, therefore VIS+IR images are used in this study.

Based on the forgoing literature reviews, it is believe that cloud separation using multi-spectral clustering can improve cloud tracking and at least provide another approach for cloud motion tracking.

Chapter 3

Statistical Pattern Recognition and Clustering

Cloud recognition requires a partitioning of the feature space, this is obtained by pattern recognition methods. Pattern recognition can be divided into two main areas: supervised and unsupervised approach (Duda and Hart 1973). The only difference in these two approaches is whether samples with known labels are given. If samples are labelled, a supervised approach is employed, otherwise unsupervised learning is applied. The main area of study in this work is unsupervised learning (clustering), but, due to the similarity of the two approaches, many techniques are equally applicable. This chapter discuss clustering techniques which are relevant to the project.

An application of pattern recognition in image analysis is to classify or label individual pixels (Kittler and Pairman 1985a, Swain et al. 1981). Other approaches (Seddon 1983, Ebert 1987, 1989, Garand 1988) classify groups of pixels (region). Every pixel is a pattern vector in the case of a multi-spectral image. The dimension of the pattern vector is equal to the number of spectral bands only if direct observed features are used (Fig. 3.1).

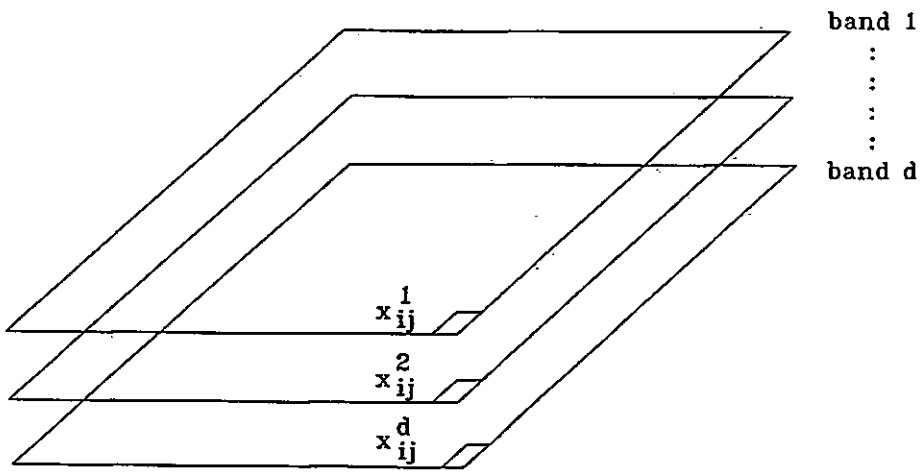


Figure 3.1: A multi-spectral image.

Each band in a multi-spectral image responds to a different window in the electromagnetic spectrum. Many more sophisticated features can be derived such as linear or non-linear combinations of bands, or different local texture measurement (Seddon 1983, Ebert 1987, 1989, Garand 1988). It is assumed that patterns that belong to the same class have similar feature values and clusters in the feature space.

The main objective of this work is the separation of cloud types before tracking cloud motion and this is to be achieved using pattern recognition techniques. METEOSAT images have three spectral bands; visible (VIS), infrared (IR), and water-vapour (WV). Most cloud types can be distinguish using VIS and IR bands (Parikh 1978). and since these bands are transmitted every 30 minutes most of the processing will use only VIS and IR images. However the algorithms derived could work with more spectral bands.

In this application the emphasis of classification is not on the accuracy of labelling patterns but rather on the homogeneity of the clusters, especially in the infrared band, because the cloud level can be inferred from infrared radiance,

and cloud at the same level should have a small temperature different. One could argue that thresholding on the IR image will have the same result as multi-spectral clustering. However, this is not the case because some cloud types may have a similar temperature but their reflectance is very difference (e.g. cumulus and stratus). Therefore a multi-spectral method is more effective than a single band method. An image pattern recognition system (Fig. 3.2) has been developed to first cluster VIS and IR images and then track clusters to ascertain cloud motion winds.

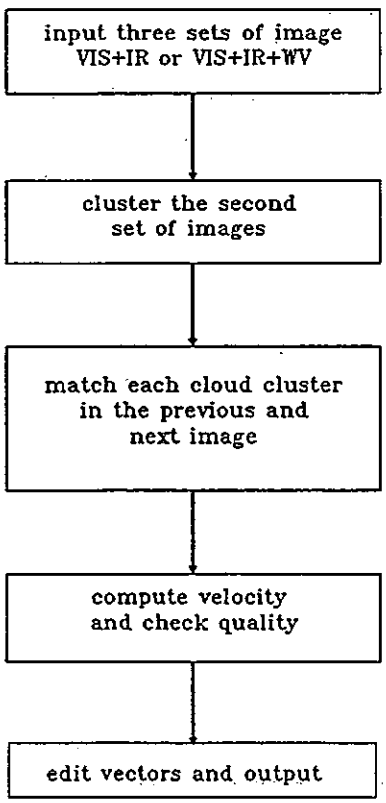


Figure 3.2: An Image pattern recognition system for cloud motion tracking.

Many wind systems reviewed in Chapter 2 do not take advantage of the full multi-spectral information; i.e. they do not separate cloud types using multi-spectral information before tracking. However, it is intuitively attractive to separate cloud types before tracking as suggested by Hubert and Whitney (1971), Parikh (1977), Parikh and Rosenfeld (1978).

A clustering approach (unsupervised learning) has been chosen in this work for cloud classification since it was not possible to build up a data base to account for cloud classes at different times of day and season (Karlsson 1989). Clustering is often refer to as "learning without a teacher", its purpose is to learn the underlying distribution of the patterns and partition the data into unimodal clusters (Devijver and Kittler 1982).

Almost all common clustering approaches have been used in this study, and so it is useful to give an overview of clustering and some pdf estimators before introducing the clustering algorithms in Chapter 4 and 5.

3.1 What is Pattern Recognition?

For our work a pattern will be defined as a vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ in multi-spectral space. The concept behind statistical pattern recognition is that if representative features can be extracted from the object to be classified, and if those features have well separated probability density functions (pdfs) in the feature space, then objects can be classified by forming decision boundaries between these densities. New objects can then be classified using the decision functions.

In case of a multi-spectral image, a pattern is usually the pixel with or without additionally derived features. Patterns that belong to a particular class are not unique, due to noise in generating the patterns, and variations among objects. The variability of patterns implies that the problem of pattern recognition is the discrimination of input data, not between individual patterns but between populations.

Figure. 3.3 shows two typical classes usually found in weather satellite images. Notice the variation in the pattern class. The land and high cloud class are disjoint clusters in the measurement space, but in fact land often overlaps with low cloud and sea, while high cloud overlaps with middle cloud. Also, most cloud classes are usually not unimodal.

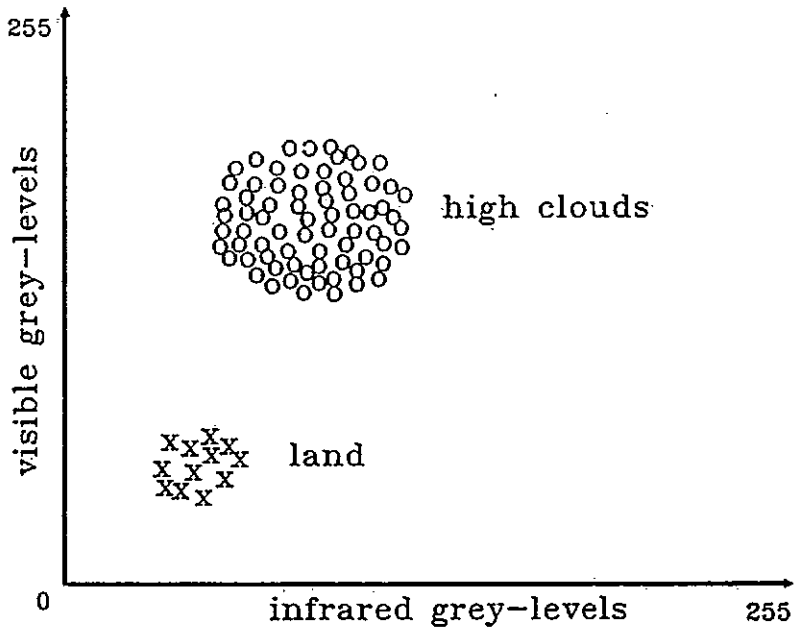


Figure 3.3: Two disjoint pattern classes.

3.1.1 Feature Selection

In most pattern recognition systems a major concern is the dimension of the measurement space. In some cases dimensions up to a hundred are not unusual. One might think that the more features the better the ability to discriminate classes, but in fact this is not true. A basic pattern recognition system is shown in Fig. 3.4. Usually certain features are common to some classes but not others, it is useful to extract or select those features which are discriminatory for each pattern class with minimum loss of information. This preprocessing of measurements is necessary not only for the reason just mention but also to reduce computational cost (Mausel et al. 1990, Sheffield 1985). When the most effective features have been selected, it is required to derive a set of classification rules from samples of classified patterns. This process is called discrimination, while new patterns are classified using the classification rules.

In this application the number of features is no more than three and it is found that feature selection is unnecessary.

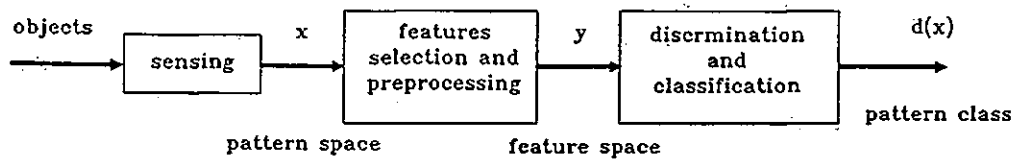


Figure 3.4: A basic pattern recognition system.

3.1.2 Decision Rules

Supervised classifiers use decision rules to classify new patterns, these rules assign a class label to a pattern based on their estimated class conditional probability. In contrast to this, decision rules based on distance measure (or dissimilarity) are used extensively in unsupervised methods.

Classification of new patterns using decision rules is based on the classical Bayesian approach. The basic concept of this approach is to minimise a decision loss function and hence the average risk. A pattern $\mathbf{x}$ is represented by a vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, where d is the dimension of the feature space and $\omega_i \quad i = 1, \ldots, C$ represent possible classes. The aim of statistical pattern recognition is to be able to determine the class membership of a given pattern with minimum probability of error by means of decision rules or discriminant functions.

Let $P(\omega_i)$ $\equiv$ the a priori probability of class ω_i

$p(\mathbf{x})$ $\equiv$ the probability that a pattern is $\mathbf{x}$

$p(\mathbf{x} \mid \omega_i)$ $\equiv$ the class conditional probability that the pattern is $\mathbf{x}$, given that it belongs to class ω_i

$P(\omega_i \mid \mathbf{x})$ $\equiv$ the a posteriori conditional probability that the pattern belongs to class ω_i, given that the pattern is $\mathbf{x}$

$P(\omega_i, \mathbf{x})$ $\equiv$ the joint probability that the pattern is $\mathbf{x}$ and that its class membership is ω_i

where $\sum_i P(\omega_i) = 1$, $\int_{R^d} p(\mathbf{x})d\mathbf{x} = 1$

$$\text{We have,} \quad P(\mathbf{x}, \omega_i) = p(\mathbf{x} \mid \omega_i)P(\omega_i) \tag{3.1}$$

$$\text{and} \quad P(\mathbf{x}, \omega_i) = p(\omega_i \mid \mathbf{x})P(\mathbf{x}) \tag{3.2}$$

Combine 3.1 and 3.2 we have

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})} \tag{3.3}$$

Ideally, a pattern $\mathbf{x}$ is assigned to class ω_i if $p(\omega_i|\mathbf{x}) > p(\omega_j|\mathbf{x})\ \forall j \neq i$. However, $p(\omega_i|\mathbf{x})$ is usually very difficult to estimate. Bayes' relation (eqn. 3.3) provides a practical means for classification, because the class conditional probability can be estimated from a set of samples with known categories.

The Bayes' classification rule becomes: assign $\mathbf{x}$ to class ω_i if

$$p(\mathbf{x}|\omega_i)P(\omega_i) > p(\mathbf{x}|\omega_j)P(\omega_j) \ \ \forall\, j \neq i \tag{3.4}$$

If it is further assumed that all classes have equal probability the rule becomes, decide $\mathbf{x} \in \omega_i$ if $p(\mathbf{x} \mid \omega_i) \geq p(\mathbf{x} \mid \omega_j)\ \ \forall\, i, j \text{and} i \neq j;\ i, j = 1, \ldots, C$. Rules 3.4 is also known as the maximum likelihood decision rule, because they use the likelihood function $\frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x}|\omega_j)P(\omega_j)}$.

The class conditional probability density function $p(\mathbf{x} \mid \omega_i)$ is not normally known. It can be estimated from a training set of correctly classified data. Although a non-parametric pdf seems appropriate, in practice a multi-variate normal

distributed pdf is usually assumed. Despite its use without any proof, this assumption performs reasonably well provided the distribution is unimodal.

The use of normal distribution is justified by the following reasons (Hand 1981):

1. The normal distribution is a good model of many naturally occurring phenomena. A possible reason for this is due to the central limit theorem.

2. The multi-variate normal distribution can be defined uniquely by its mean vector μ and a covariance matrix Σ . This property of normal distribution allows very simple and efficient computation.

3. After any non-singular linear transformation of the axes a normal distribution is still normal but with different parameters.

4. The principle of maximum entropy states that, if the probability density function characterizing a random variable is not known, the probability density function which maximizes the entropy of the random variable subject to any known constraints, and it can be shown that normal distribution is a satisfactory choice.

The class conditional pdf using a Gaussian distribution is

$$p(\mathbf{x} \mid \omega_i) = (2\pi)^{-d/2}|\Sigma_i|^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i)\right] \qquad (3.5)$$

where d is the dimension of $\mathbf{x}$, μ_i is the mean vector, Σ_i is the covariance matrix for class ω_i, and $|\Sigma_i|$ is the determinant of the covariance matrix.

Substituting 3.5 into 3.4 and taking the logarithm, the maximum likelihood rule becomes

assign $\mathbf{x} \in \omega_i$ if

$$\log|\Sigma_i| + (\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i) - \log P(\omega_i) <$$
$$\log|\Sigma_j| + (\mathbf{x}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}-\mu_j) - \log P(\omega_j) \quad \forall\, j \neq i \qquad (3.6)$$

The second stage of the Global-Local clustering algorithm in Chapter 4 clusters patterns based on eqn. 3.6.

A simple decision function for two pattern classes is shown in Figure 3.5, where ϵ_1 and ϵ_2 is the class error probabilities $d(\mathrm{x}) = 0$ is the decision boundary.

$$\epsilon_1 = \int_{\Omega_2} p(\mathrm{x} \mid \omega_1) d\mathrm{x} \tag{3.7}$$

$$\epsilon_2 = \int_{\Omega_1} p(\mathrm{x} \mid \omega_2) d\mathrm{x} \tag{3.8}$$

where Ω_1 such that $p(x|\omega_1) - p(x|\omega_2) \geq 0$

Ω_2 such that $p(x|\omega_1) - p(x|\omega_2) \leq 0$

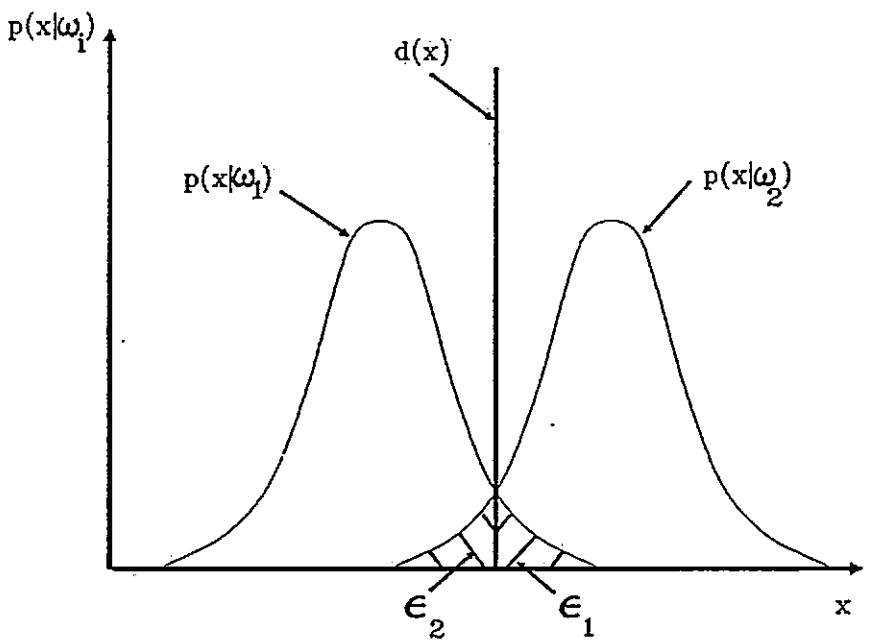


Figure 3.5: A simple decision function for two pattern classes.

Notice that in Figure 3.5 the maximum likelihood decision surface is chosen to minimize the sum of error. A full treatment of error rate can be found in Devijver and Kittler (1982).

A criterion function based on minimum error probability is used in the second stage of the Global-Local clustering algorithm presented in Chapter 4.

3.1.3 Distances

The maximum likelihood decision rule (eq. 3.4) classifies patterns based on the ratio of two probabilities. An alternative way to classify patterns, is to use a distance based discrmination. Consider two sample sets in d dimensional space, the greater the difference between the two populations ω_1 and ω_2, the greater will be the separation between the two groups. A pattern $\mathbf{x}$ would be allocated to the population to whose training set it is "nearer".

It is natural to assign $\mathbf{x}$ to class ω_i if the distance of $\mathbf{x}$ in the feature space is such that

$$\delta(\mathbf{x}, \omega_i) \leq \delta(\mathbf{x}, \omega_j) \quad \forall i \neq j \tag{3.9}$$

where $\delta(\mathbf{x}, \omega_i)$ is a distance function of pattern $\mathbf{x}$ and class ω_i. The distance function provides a measure of dissimilarity (or similarity) between patterns. This concept of classification is illustrated in Figure 3.6.

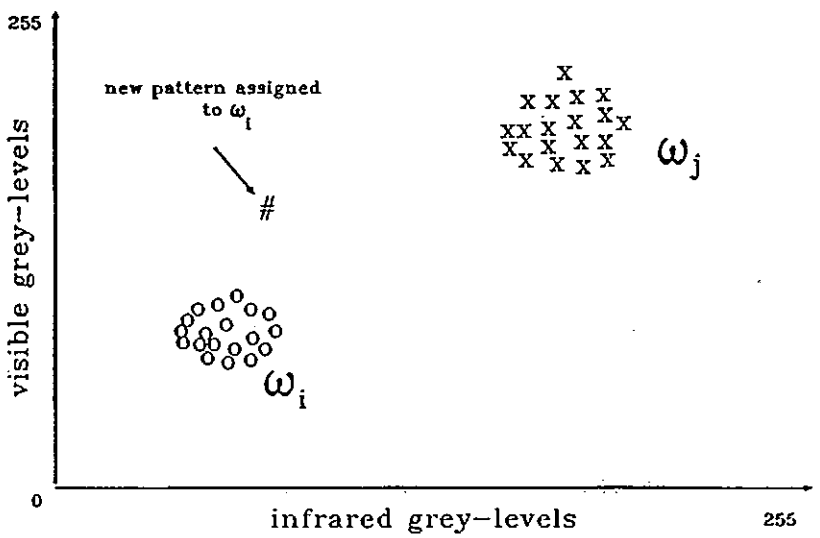


Figure 3.6: Pattern classification using distance.

The most familiar distance measure is probably the Euclidean metric. However, there are many other measures which are more suitable in certain situations,

e.g. when the knowledge about the cluster structures is known beforehand. We shall discuss problems of using distance measures in clustering, and suggest an approach to derive a distance measure suitable for METEOSAT images which is being used in the Global-Local clustering algorithm presented in Chapter 4.

3.1.4 Pattern Classifier

There are many methods to derive decision rules. So far we have assumed that $p(\mathbf{x} \mid \omega_i)$ came from a known family of distributions, leaving only the parameters to be estimated. Other methods are *non-parametric*, this applies when we can not make simplifying assumptions about the pdfs or decision surfaces.

Besides the division of parametric and non-parametric methods, pattern recognition can further divided into two main areas, namely supervised classification and unsupervised classification. Suppose we are given a set of samples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ in which the samples are independent and identically distributed random variables with continuous pdf $p(\mathbf{x})$. This sample set is *labelled* if the classification is supervised, while if the classification is unsupervised the sample set is *unlabelled.*

Both supervised and unsupervised learning may be formulated as a classical estimation problem. Unsupervised learning however, causes the problem solution to generally be much more complex than when there is supervised learning. To simplify the unsupervised problem it is often necessary to apply engineering intuition. In weather images most pattern classes are time varying, so the unsupervised approach seems appropriate because supervised learning usually can not detect multi-modal classes and requires training samples which are difficult to obtain.

Clustering is a common approach for unsupervised classification and it is very often used to generate training samples for supervised classification. Figure 3.7 shows that clustering makes sense only in terms of a priori knowledge used, and this is partly due to the inability to define the term cluster. The uncertainty in the number of clusters in the weather images are further confused by high overlap

of cloud classes. Very often using visible and infrared features can not identify semi-transparent cloud, because the background radiance always interferes with the cloud radiance (Desbois et al. 1982).

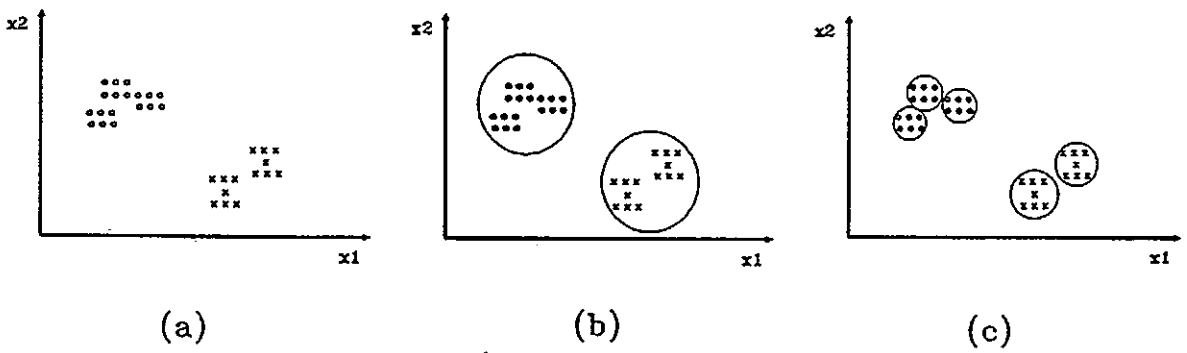


Figure 3.7: a) Samples in measurement space, b) Possible partition with two classes, c)Possible partition with five classes.

We shall introduce two density estimation methods which are used in the clustering algorithms in Chapter 4 and 5. The first is parametric and the second is non-parametric, both methods are very popular in supervised and unsupervised classification. The first method is called maximum likelihood estimator and the second is the histogram estimator.

3.2 Parametric Density Estimator

The major goal of supervised learning is to estimate $P(\omega_i)$ and $p(\mathbf{x} \mid \omega_i)$ and use Bayes decision rule to classify input patterns. Usually estimation of $P(\omega_i)$ presents no problem, but not the conditional densities. The general problem in estimating the conditional density is the small number of samples and large dimension of feature vectors. If a parametric model of a pdf can be assumed (usually Gaussian), then the problem can be simplified to estimation of mean vector μ and Σ_j of $p(\mathbf{x} \mid \omega_j)$. We will assume patterns are normally distributed throughout this section.

3.2.1 Maximum Likelihood Estimator

The Maximum likelihood estimator is probably the most popular method for parameter estimation due to its simplicity. Less common methods such as distance minimisation, Bayes method, and sequential methods can be found in Hand (1981), Duda and Hart (1973) and Tou and Gonzalez (1974).

Maximum likelihood estimation of parameters requires construction of a criterion function which is a function of the unknown parameters and the samples. Those parameter values are then found by optimizing this function. The maximum likelihood estimator treats the parameters as quantities whose values are fixed but unknown. The estimator selects the parameter vector which has largest a posteriori probability (Wilks 1962).

Suppose there are C sets of samples $\mathbf{X}_1, \ldots, \mathbf{X}_C$ with the sample in $\mathbf{X}_j$ having been drawn independently and randomly according to the probability law $p(\mathbf{x} \mid \omega_j)$, assume that $p(\mathbf{x} \mid \omega_j)$ has a known parametric form, and is therefore determined uniquely by the value of a parameter vector θ_j. The Maximum likelihood estimator uses the information provided by the samples to obtain a good estimator for the unknown parameter vectors $\theta_1, \ldots, \theta_C$, for which the probability of obtaining the observed samples is a maximum. To simplify the problem, it is assumed that the class conditional densities are independent. This assumption

allows one to work with each class separately. The traditional way of using the information in the sample set $\mathbf{X}$, is to use an estimate $\hat{\theta}_i$ in $p(\mathrm{x} \mid \theta_i)$ (Aitchison et al. 1977).

Suppose the set $\mathbf{X}$ contains n samples, $\{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$. Then since the samples were drawn independently,

$$p(\mathbf{X} \mid \theta) = \prod_{k=1}^{n} p(\mathrm{x}_k \mid \theta) \tag{3.10}$$

$p(\mathbf{X} \mid \theta)$ is called the likelihood function of θ with respect to the set of samples. The maximum likelihood estimator of θ is, by definition, the value $\hat{\theta}$ that maximizes $p(\mathbf{X} \mid \theta)$ (Figure 3.8). The classical approach is to differentiate $p(\mathrm{x} \mid \theta)$ with respect to θ, equate $\partial p(\mathbf{X} \mid \theta)/\partial\theta = 0$, and solve for $\hat{\theta}$.

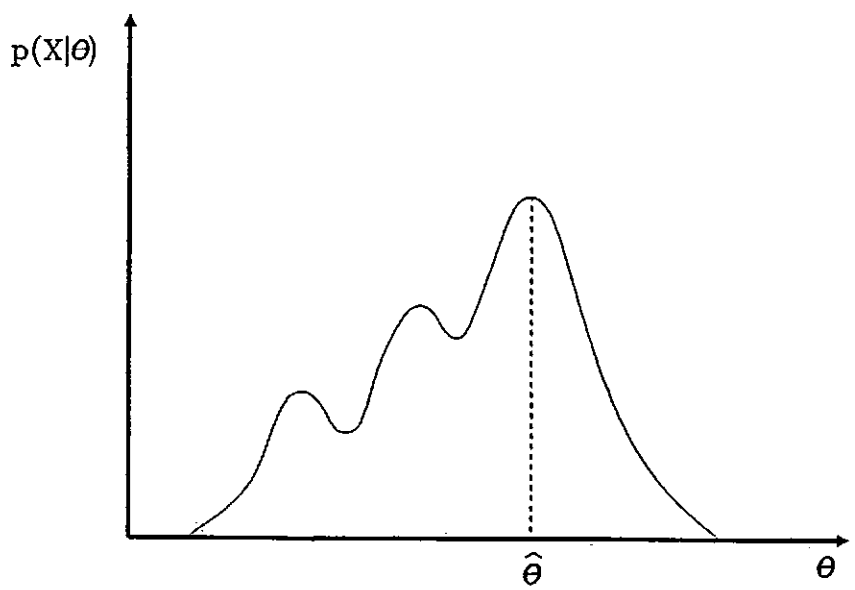


Figure 3.8: The maximum likelihood estimate for a parameter θ.

Suppose the sample set $\mathbf{X}$ is from a normal distribution with parameter $\theta = (\mu, \Sigma)$, the maximum likelihood estimate of the mean μ and covariance matrix Σ is obtained by substituting

$$p(\mathrm{x} \mid \theta) = (2\pi)^{-d/2}|\Sigma|^{-1/2} \exp\left[-\tfrac{1}{2}(\mathrm{x}-\mu)^T\Sigma^{-1}(\mathrm{x}-\mu)\right]$$

which
into eqn. 3.10, then gives the likelihood function

$$p(\mathbf{X} \mid \theta) = (2\pi)^{-dn/2}|\Sigma|^{-n/2} \exp\left[-\tfrac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i - \mu)^T\Sigma^{-1}(\mathbf{x}_i - \mu)\right] \tag{3.11}$$

It is more convenient to work with the logarithm of the likelihood function than with the likelihood function itself,

$$\begin{aligned} L' &= \log p(\mathbf{X} \mid \theta) \\ &= \frac{n}{2}\log|\Sigma|^{-1}| - \frac{1}{2}\mathrm{tr}\Sigma^{-1}V \\ &\quad -\frac{n}{2}\mathrm{tr}\Sigma^{-1}(\mathbf{m}_n - \mu)(\mathbf{m}_n - \mu)^T + \frac{dn}{2}\log 2\pi \end{aligned} \tag{3.12}$$

Setting $\frac{\partial L'}{\partial \mu}$ and $\frac{\partial L'}{\partial \Sigma}$ to zero (see Appendix A for proof)

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \tag{3.13}$$

$$\hat{\Sigma} = \frac{1}{n}\sum_{j=1}^{n}(\mathbf{x}_j - \hat{\mu})(\mathbf{x}_j - \hat{\mu})^T \tag{3.14}$$

The Maximum likelihood estimator is used to estimate parameters of cluster models in the second stage of the Global-Local algorithm described in Chapter 4, it is also used to obtain parameters of mutual information model described in Chapter 5.

3.3 Non-Parametric Density Estimator

In some cases if the probability densities can not be approximated by a general parametric form pdf, we have to use the sample points to estimate the densities with non-parametric methods. The assumption that the forms for the underlying density function can not be characterised by parametric form is not uncommon. For example, when the density function of classes are multi-modal.

There are three major type of non-parametric estimators: 1) the histogram method, 2) the kernel method, and 3) the k-nearest-neighbour (k-nn) method

(Duda and Hart 1973). We shall only look at the histogram estimator, and the other two are essentially a generalization of the histogram estimator.

3.3.1 Histogram Estimator

Histograms are the conceptually simplest method of estimating a pdf. The generalisation of the histogram from one dimension to many is simply to partition the whole space into disjoint cells of equal volume.

Consider a small region R^d with volume V (Fig. 3.9) about the point x where the density $p(\mathrm{x} \mid \omega_i)$ is to be estimated. Since we are working with single class, we shall drop the class subscript. Given n independent samples $\mathrm{x}_1, \ldots, \mathrm{x}_n$ the probability P that k_j of these n samples fall into this region is

$$P = \int_{R^d} p(\mathrm{x}')d\mathrm{x}' \tag{3.15}$$

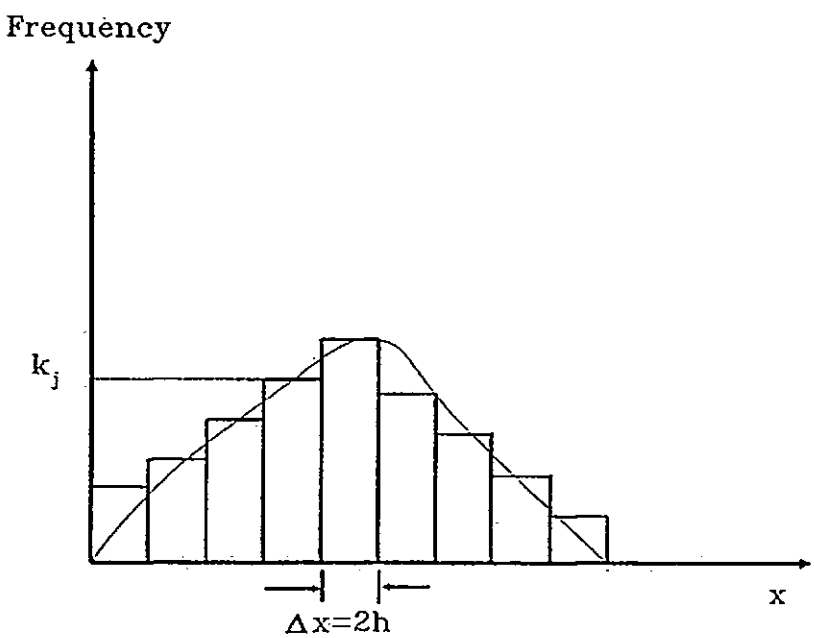


Figure 3.9: The approximation of probability density function by histogram, where h determine the volume of a cell.

If the region R^d is sufficiently small, we can write

$$\int_{R^d} p(\mathbf{x}')d\mathbf{x}' \approx p(\mathbf{x})V \tag{3.16}$$

$$\text{so, } p(\mathbf{x}) \approx \frac{P}{V} \tag{3.17}$$

It is required to estimate P from the set of samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Since the samples are independent, the probability of finding any k of these samples in R^d is given by the binomial distribution

$$P_k = p(k \mid n, P) = \frac{n!}{k!(n-k)!} P^k (1-P)^{n-k} \tag{3.18}$$

The maximum likelihood estimation of P_k can be found by differentiating P_k with respect to k and setting dP_k/dk to zero (Duda and Hart 1972),

$$\begin{aligned} \frac{dP_k}{dk} &= \begin{pmatrix} n \\ k \end{pmatrix} P^{k-1}(1-P)^{n-k-1}[k(1-P) - P(n-k)] \\ &= 0 \end{aligned} \tag{3.19}$$

Solving for P, the estimate is given by

$$\hat{P} = \frac{k}{n} \tag{3.20}$$

The local estimated density of $p(\mathbf{x})$ is given by

$$\hat{p}(\mathbf{x}) = \frac{k}{nV} \tag{3.21}$$

If we consider the region R^d as a cell, then the histogram estimation of $p(\mathbf{x})$ is given by 3.21. The histogram estimate for cell b_j is

$$\hat{p}_j(\mathbf{x}) = \frac{k_j}{nV} \tag{3.22}$$

where k_j is the number of points lie in cell b_j and V is same for all cells.

This method has the advantage that the points themselves do not need to be stored after the estimate has been made. Only statistics describing the cell location, number of points need be retained. The histogram approach is largely

limited by the dimension of the feature space. If the space is divided into N intervals then the number of cell is equal to N^d.

In some cases there may be another problem occurring due to discontinuity between cell boundaries. The estimate given by 3.22 is applied to the volume occupied by a cell, so there is an abrupt change of level between two adjacent cells. The Spline line function can be used to smooth the boundaries (Ichida and Kiyono 1975).

The first problem, with the histogram method is the prohibitively large number of cells needed even in relatively low dimensional problems, this can be alleviated by storing non-empty cells only (Shlien and Smith 1975). Another method is to let the data somehow determine the cell locations, number of points as suggested by Sebestyen and Edie (1966).

Local density estimate are commonly used in mode seeking clustering algorithm which shall be discussed later. A practical method to construct multi-dimensional histograms was described by Narendra and Goldberg (1977) and Wharton (1983). This method is incorporated in the first stage of the Global-Local clustering algorithm which partitions a multi-dimensional histogram (see Chapter 4).

3.4 Unsupervised Learning

Unsupervised learning is to classify samples without any prior knowledge, such as sample labels, parameters and forms characterise the underlying distribution.

If no prior knowledge of the class pdf is given, the problem becomes one of decomposing a mixture of distributions into their components. A mixture pdf for C classes of samples is given by (Everitt and Hand 1981)

$$p(\mathbf{x} \mid \theta) = \sum_{j=1}^{C} p(\mathbf{x} \mid \omega_j, \theta_j) P(\omega_j) \tag{3.23}$$

Assume the form of the class condition pdf $p(\mathbf{x} \mid \omega_j, \theta_j)$ is known. All that is unknown is the values for the C parameter vectors $\theta_1, \ldots, \theta_C$. The conditional densities $p(\mathbf{x} \mid \omega_j, \theta_j)$ are called the component densities, and the a priori probabilities $P(\omega_j)$ are called the mixing parameters. If the parameters can be estimated from the samples, we can decompose the mixture into its components.

3.4.1 Unsupervised Maximum Likelihood Estimation

We introduce clustering using the maximum likelihood approach for densities decomposition, this approach is very restrictive since a parametric model and number of clusters are assumed known. The maximum likelihood method can be used to learn the parameters of a mixture density (Wolfe 1970, Hasselblad 1966).

Assumed the following:

1. The samples come from a known number C of classes.

2. The mixing parameters $P(\omega_j)$ for each class are known, $j = 1, \ldots, C$.

3. The form for the class conditional pdf $p(\mathbf{x} \mid \omega_j, \theta_j)$ are known, $j = 1, \ldots, C$.

4. All that is unknown are the values for the C parameter vectors, $\theta_1, \ldots, \theta_C$.

Given a set of unlabelled sample $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ drawn independently from the mixture density

$$p(\mathbf{x} \mid \theta) = \sum_{j=1}^{C} p(\mathbf{x} \mid \omega_j, \theta_j) P(\omega_j)$$

Assume independence of samples, we have

$$p(\mathbf{X} \mid \theta) = \prod_{k=1}^{n} p(\mathbf{x}_k \mid \theta) \tag{3.24}$$

It is required to find the estimate $\hat{\theta}$ that maximise the mixture $p(\mathbf{X} \mid \theta)$. The logarithm of the likelihood function is

$$
\begin{aligned}
L &= \sum_{k=1}^{n} \log p(\mathbf{x}_k \mid \theta) \\
&= \sum_{k=1}^{n} \log \left[\sum_{j=1}^{C} p(\mathbf{x}_k \mid \omega_j, \theta_j) P(\omega_j) \right] \qquad (3.25)
\end{aligned}
$$

Wolfe (1970) shown that the estimate which maximizes eqn. 3.25 are given by:

$$
\begin{aligned}
\hat{P}(\omega_k) &= \frac{1}{n} \hat{P}(\omega_k \mid \mathbf{x}_i, \theta) \qquad (3.26) \\
&\quad k = 1, \ldots, C-1 \\
\hat{\mu}_k &= \frac{1}{n\hat{P}(\omega_k)} \sum_{i=1}^{n} \hat{P}(\omega_k \mid \mathbf{x}_i, \theta) \mathbf{x}_i \qquad (3.27) \\
&\quad k = 1, \ldots, C \\
\hat{\Sigma}_k &= \frac{1}{n\hat{P}(\omega_k)} \sum_{i=1}^{n} \hat{P}(\omega_k \mid \mathbf{x}_i, \theta)(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T \qquad (3.28) \\
&\quad k = 1, \ldots, C
\end{aligned}
$$

It is found that the maximum likelihood estimators for the unsupervised case (mixture) are analogous to those in the supervised case (single distribution). While in the unsupervised case the sample points are weighted by the posteriori probability, so all samples are contribute to the estimation.

Equation 3.26, 3.27 and 3.28 do not give θ explicitly, they must be solved using some type of iterative (hill climbing) procedure. One iterative technique which is commonly used has two stages (Wolfe 1970, Hasselblad 1966). The first stage estimates the membership probability for the kth component 3.26 and the second stage uses these membership probability estimates to update the estimates of $\hat{P}(\omega_k), \hat{\mu}_k, \hat{\Sigma}_k$. These two steps are then repeated iteratively. All iterative techniques are not guarantied to yield the global maximum, but the convergences can be improved by a set of good initial values of θ. Duda and Hart (1973 Ch.6) use the maximum likelihood estimator on a simple one dimensional, two components normal mixture. He demonstrated that the solution depends on the initial estimate, and multiple solutions always exist using a clustering technique. Clustering techniques which shall be discussed later are popular and practical

approach for unsupervised learning. Although it is suboptimal, the simplicity and efficiency always out grows their disadvantages.

Since unsupervised maximum likelihood estimation requires many assumption and some of them are unrealistic in many applications, so this is usually replaced by other clustering methods. In the rest of this chapter the most commonly used clustering techniques and the problem of in choosing a distance measure are discussed. Concepts of these techniques are used in clustering algorithms developed in Chapter 4 and 5.

3.5 Clustering

In the last section clustering was introduced as a mixture decomposition approach. Clustering also finds many applications outside the pattern recognition area. It is generally known as a tool for exploratory data analysis (Jain and Dubes 1988). Cluster analysis attempts to organise data into their natural structure such that patterns in the same cluster are more similar than patterns in different clusters. This organisation can be a partitioning of the data set into non-overlapping subsets, or it can be a hierarchy of groups (see Fig. 1.2 for various methods).

The concept of distance was introduced in section 3.1.3, it is a measure of similarity of two patterns. The use of similarity in clustering is analogous to human recognition of objects. We tend to group objects based on their similarity, so it is natural to use this concept in clustering. Similarity (dissimilarity) is the heart of clustering, but from the first moment when a clustering method is chosen a structure is imposed on the data which is somehow conflicting with the original goal of clustering (Hartigan Ch.2 1975). For example, the hierarchical methods "single linkage" is suitable for clustering enlongated clusters, if this method is applied to normal distributed data, the original structure may not be recovered. So if a priori knowledge about the data is sparse, different clustering methods should be tried before any results can be accepted.

A general problem with clustering is to solve the problem of determining the

number of clusters. Usually this problem is scale dependent as shown in Fig. 3.7. This problem is referred as cluster validation (Duda and Hart Ch.6 1973, Dubes and Jain 1979, Jain and Moreau 1987). If the data can not be projected into two or three dimensions such that its structure can be viewed, then it becomes necessary to test the validity as an objective measure of the clustering results. In clustering of imagery data, these problem is caused by the uncertainty between objects boundaries. In this study cluster validity is not a serious problem since the approximate number of clusters can be obtained by inspecting the original image, and for most situations (same image size) the number of clusters only differ by one or two.

3.5.1 Dissimilarities

The dissimilarity between the ith and kth patterns is denoted $\delta(i,k)$ and must satisfy the following four properties (Anderberg 1973):

1. $\delta(\mathbf{x}_i, \mathbf{x}_i) = 0, \quad \forall\, i$

2. $\delta(\mathbf{x}_i, \mathbf{x}_k) = \delta(\mathbf{x}_k, \mathbf{x}_i), \quad \forall\, i,k$

3. $\delta(\mathbf{x}_i, \mathbf{x}_k) \geq 0, \quad \forall\, i,k$

4. $\delta(\mathbf{x}_i, \mathbf{x}_k) \leq \delta(\mathbf{x}_i, \mathbf{x}_m) + \delta(\mathbf{x}_m, \mathbf{x}_k), \quad \forall\, i,k,m$

A distance measure only needs to satisfy 1–3, while for a distance metric it must satisfy 1–4. Assumption 1 implies that an object is zero distance from itself and that two points zero distance apart are identical. Assumption 2 implies symmetry of distance, assumption 3 prohibits negatives distances, and assumption 4 is known as the triangle inequality, this requires that the length of one side of a triangle be no longer than the sum of the lengths of the other two sides.

The Euclidean metric is a special case of the Minowski metric

$$\delta_r(\mathbf{x}_i, \mathbf{x}_k) = \left(\sum_{j=1}^{d} |\mathbf{x}_{ij} - \mathbf{x}_{kj}|^r \right)^{1/r} \quad \text{where } r \geq 1 \tag{3.29}$$

where x_{ij} is the jth variable of the ith pattern. If $r = 2$ we have the Euclidean metric. Other special cases of the Minkowski metric are the City block and sup-norm metric

$$\delta_1(\mathbf{x}_i, \mathbf{x}_k) = \sum_{j=1}^{d} |x_{ij} - x_{kj}| \tag{3.30}$$

$$\delta_\infty(\mathbf{x}_i, \mathbf{x}_k) = \sup_{1 \le j \le d} |x_{ij} - x_{kj}| \tag{3.31}$$

Sibson (1972) argues convincingly, order relationships are more important than numerical values. A dissimilarity measure *need not be a metric*, for example the squared Euclidean distance is usually used to replace the Euclidean distance for more efficient computation. However the squared Euclidean distance is not a metric, because it does not satisfy the triangle equality.

3.5.2 Problems of Measuring Dissimilarities in Clustering

If prior information about the data can be obtained, it allows us to choose an appropriate distance measure based on the model of the data structure. For example, most weather image data can be modelled by multi-variate normal mixture (Pairman and Kittler 1986). In this case, a distance function can be derived from the normal distributed model which takes into account of the cluster size and population.

When prior information is not available, a common approach is to normalise the data such that Euclidean distance can be used to produce better clustering. For example in Figure 3.10, if Euclidean distance is used the point $\mathbf{x}$ will be assigned to the wrong cluster ω_1. This problem may be solved by taking account of the variance of the clusters. Ideally we would like to normalise the data so that within clusters variances are approximately equal. This is essentially a normalization of the features variables in *each cluster*. Let the mean of the jth variable of cluster ω_k be $m_{jk} = \frac{1}{n_k} \sum_{i=1}^{n_k} x_i \quad \forall x_i \in \omega_k$, and s_{jk}^2 be the variance of the jth variable of cluster ω_k,

$$s_{jk}^2 = \frac{1}{n_k}\sum_{i=1}^{n_k}(\mathrm{x}_{ij} - \mathrm{m}_{jk})^2 \quad j = 1, \ldots, d \tag{3.32}$$

Unfortunately, this normalization can not be justified if the clusters are not known a priori.

Another reason for normalization of data is the scaling effect (Hartigan Ch.2 1975). If the values of some variables are particularly large, these variables will dominate the distance measure. To equalise the importance of each feature variable, each variable could be scaled by dividing by its sample standard deviation

$$s_j^2 = \frac{1}{n}\sum_{i=1}^{n}(\mathrm{x}_{ij} - \mathrm{m}_j)^2 \quad j = 1, \ldots, d \tag{3.33}$$

where $\mathrm{m}_j = \frac{1}{n}\sum_{i=1}^{n}\mathrm{x}_i \ \forall \mathrm{x}$ is the mean of the ith variables.

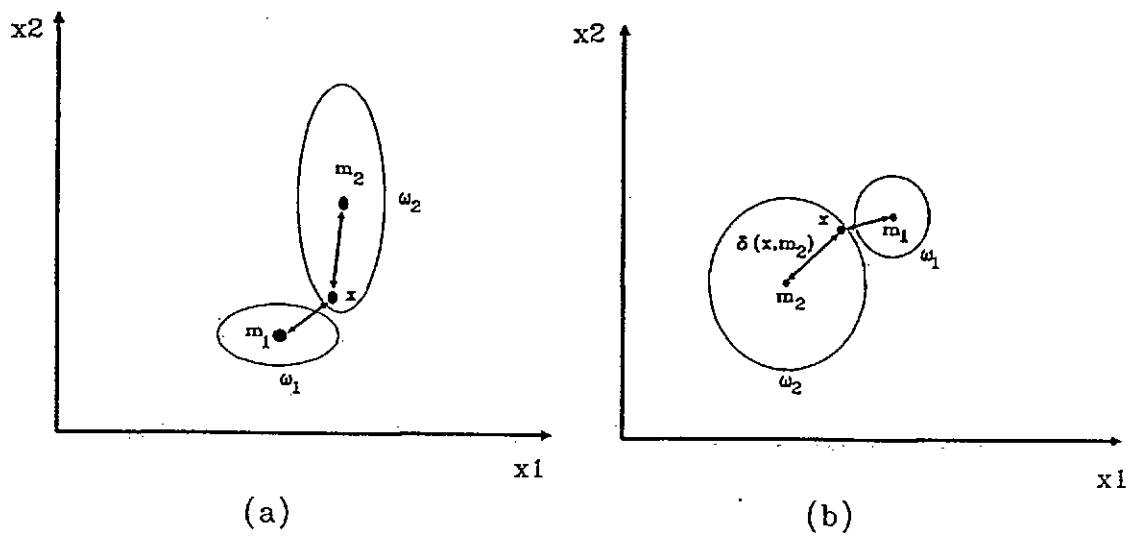


Figure 3.10: A point x in these cases should belong to ω_2, although $\delta(\mathrm{x}, \mathrm{m}_2) > \delta(\mathrm{x}, \mathrm{m}_1)$, where $\delta(\mathrm{x}, \mathrm{m}_1)$ is the Euclidean distance between cluster centre m_1 and point x.

Equation 3.33 rescales all the variables to have unit variances with respect to the *whole* data set. A problem with this normalization method is that after normalization those variables with relatively large Between Cluster Variances will be reduced in importance, as their scaling factors are larger. This means that the overall between cluster variance will be reduced relative to the within cluster variance and the clusters will become less distinct (Hartigan Ch.2 1975).

An alternative to normalizing the data and using Euclidean distance is to use some kind of normalized distance, such as Mahalanobis distance

$$\delta(\mathbf{x}_i, \mu_j) = \left\{(\mathbf{x}_i - \mu_j)\Sigma^{-1}(\mathbf{x}_i - \mu_j)^T\right\}^{1/2} \tag{3.34}$$

where Σ^{-1} is the covariance matrix of cluster ω_j.

Generally, we would prefer to use 3.32 for normalization, because 3.33 reduces the distinctness of clusters. So normalization is just one method to deal with the variation of cluster size and shape. Clearly, the effect of normalization is more difficult to access when applied to non-linearly separable clusters. Hartigan (Ch.2 1975) highlighted the difficulty of normalization as a basic circularity:

1. In order to cluster patterns, it is necessary to propose a measure of distance between patterns.

2. In order to define distance, it is necessary to weight the variables.

3. In order to weight the variables, it is necessary to know the clusters of objects so within-cluster variances can be equalised.

There is no doubt that no distance measure is universal and the choice of a suitable measure should be obtained from prior information of the data whenever possible.

In spite of all the criticism about normalization, Fukunaga and Koontz (1970) proposed a normalizing transformation for clustering and showed that the result using this transformation for clustering Gaussian data was improved. Suppose we wish to partition a data set $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of d-dimensional vectors into C clusters

with population $n_1, \ldots, n_C$. The scatter matrices are then defined as

$$\text{Total scatter } T \stackrel{\text{def}}{=} \sum_{k=1}^{n} \mathbf{x}_k \mathbf{x}_k^T \tag{3.35}$$

where the data set has zero mean.

$$\text{Within group scatter } W_j \stackrel{\text{def}}{=} \sum_{\mathbf{x}_k \in \omega_j} (\mathbf{x}_k - \mathbf{m}_j)(\mathbf{x}_k - \mathbf{m}_j)^T \tag{3.36}$$

$$\text{where } \mathbf{m}_j = \frac{1}{n_j} \sum_{\mathbf{x}_k \in \omega_j} \mathbf{x}_k$$

$$\text{Total within group scatter } W \stackrel{\text{def}}{=} \sum_{j=1}^{C} W_j \tag{3.37}$$

$$\text{Between group scatter } B \stackrel{\text{def}}{=} \sum_{j=1}^{C} n_j \mathbf{m}_j \mathbf{m}_j^T \tag{3.38}$$

$$\text{and } T = W + B \tag{3.39}$$

The eigenvalues of $W^{-1}B, \lambda_1, \ldots, \lambda_d$ are invariant under non-singular linear transformations of the data set.

Three clustering criteria (discuss later) have been defined in terms of these scatter matrices. They are

$$J_0 = \text{tr}W = \sum_{j=1}^{C} \sum_{\mathbf{x}_k \in \omega_j} \|\mathbf{x}_k - \mathbf{m}_j\|^2 \text{ (minimize)} \tag{3.40}$$

$$J_1 = |T|/|W| = \prod_{i=1}^{d} (1 + \lambda_i) \text{ (maximize)} \tag{3.41}$$

$$J_2 = \text{tr}W^{-1}B = \sum_{i=1}^{d} \lambda_i \text{ (maximize)} \tag{3.42}$$

J_0 was proposed by Casey and Nagy (1968) and McQueen(1967), which is not invariant under non-singular linear transformations of the data set. A non-singular linear transformation of a positive definite scatter matrix T proves that a non-singular matrix A exists such that $ATA^T = I$. J_1 and J_2 was proposed by Friedman and Rubin (1967), these two criteria are invariant to non-singular linear transformation. Fukunaga and Koontz (1970) showed that J_1 and J_2 are superior to J_0 in the sense of the performance requirement. However only J_0 can be directly optimized using the K-means type algorithm (MacQueen 1967). J_0 is

computationally more efficient than J_1 and J_2, so Fukunaga and Koontz (1970) derived a normalizing transformation and applied on J_0 to produce

$$J_0' = \sum_{i=1}^{d} \frac{1}{1+\lambda_i} \tag{3.43}$$

which is invariant under non-singular linear transformation of the xś. Fukunaga and Koontz (1970) showed that for two group Gaussian data, J_0' yields the same optimum partition as J_1 and J_2 with greater efficiency.

Lumelsky (1982) argued that the weighting (another form of normalization) of variables should be done within the clustering stage and not as a preprocessing step. A clustering transformation was incorporated into a K-means type clustering algorithm. The criterion to be minimized is the average within group square error

$$J = \sum_{k=1}^{C} \frac{1}{n_k(n_k-1)} \sum_{i,j\in\omega_k} \sum_{p=1}^{d} w_p^2(\mathrm{x}_{ip} - \mathrm{x}_{jp})^2, \quad i \neq j \tag{3.44}$$

where w_p is the weight for pth variables and is

$$\begin{aligned} w_q &= d\frac{1/c_q}{\sum_{p=1}^{d} 1/c_p}, \quad q = 1, \ldots, d \\ \text{where } c_q &= 2\sum_{k=1}^{C} \frac{1}{n_k(n_k-1)} \sum_{i,j\in\omega_k} (\mathrm{x}_{iq} - \mathrm{x}_{jq})^2 \end{aligned} \tag{3.45}$$

The clustering results compared favourably with algorithm using Mahalanobis distance and equal variance normalisation.

In general normalization is an attempt to equalize the variation of clusters, it should be used carefully to assist clustering, rather than taken for granted. It is also believed that normalization usually improves clustering of hyperspherical or hyperellipsoidal clusters, although a different clustering method is suitable to identify either linearly or non-linearly separable clusters.

It was mentioned that most weather image data can be modelled by a normal mixture. With this prior information, Kittler and Pairman (1985b) derived a distance function using Bayes minimum error criterion (section 3.1.2). The distance measure is suitable for clustering of normal mixture and produces better results

than either the Euclidean or Mahalanobis distance. In Chapter 4 a Global-Local clustering algorithm has been developed based on this distance function.

3.5.3 Partitional Clustering

Partitional clustering algorithmscan be divided into two categories, they are iterative and non-iterative algorithms. Iterative partitional clustering algorithms allow patterns to be transferred from one cluster to another to optimize some criterion function. The choice of a criterion function and a distance measure are most important in iterative clustering algorithm. Jain and Dubes (Ch.3 1988) suggested that criteria can be classified as global or local. A global criterion represents each cluster by a prototype (usually the cluster centroid) and assigns patterns to clusters according to the most similar prototype. Many iterative clustering algorithms (ISODATA, K-means) use the global criterion. A local criterion forms clusters by utilizing local structure in the data which is most popular in mode seeking type algorithms. For example, clusters can be formed by identifying high density regions in the pattern space (Narendra and Goldberg 1977, Torn 1976, Ince 1981, Wharton 1983) or by assigning a pattern and its k-nearest neighbours to the same cluster (Gowda and Krishna 1978, Urquhart 1982, Jarvis and Patrick 1973).

Our objective is to partition n patterns into C groups such that a criterion is optimized. The solution to this partitional problem is straight forward, this can be accomplished by searching for all possible combination and selecting the best one. If iterative clustering algorithms are used, a criterion must be chosen. Criteria are highly dependent on problem parameters (Jain and Dubes Ch.3 1988). Some criteria have been mentioned in section 3.5.2. We repeat those criteria here for convenience.

$$J_0 = \mathrm{tr}W \tag{3.46}$$

$$J_1 = |T|/|W| \tag{3.47}$$

$$J_2 = \mathrm{tr}W^{-1}B \tag{3.48}$$

$$J_3 = \mathrm{tr}W \cdot \mathrm{tr}B \tag{3.49}$$

The most popular criterion is J_0

$$\mathrm{tr}W = \sum_{k=1}^{C} \mathrm{tr}W_k = \sum_{k=1}^{C} \sum_{i=1}^{n_k} (\mathbf{x}_{ik} - \mathbf{m}_k)^T (\mathbf{x}_{ik} - \mathbf{m}_k) \tag{3.50}$$

which is equal to the sum of variance of all clusters, and it is equivalent to the sum of square error criterion. The $\mathrm{tr}W$ criterion is invariant under orthogonal transformations, such as rotations, but is not invariant under non-singular linear transformations. That is, the minimum square error partition may change if the coordinate axes are scaled (Fukunaga and Koontz 1970). However, J_1 and J_2 (Friedman and Rubin 1976) are invariant under non-singular linear transformations. Friedman and Rubin described a two passes algorithm for optimizing J_1 and J_2. The first pass is a hill-climbing pass, it changes the cluster label of an object only to improve the criterion function. K-means (MacQueen 1967) is a popular algorithm for performing such task. The K-means algorithm is as follow:

Step 1. Select C initial cluster centres.

Step 2. Assign patterns to the closest centre using a distance measure.

Step 3. Update the centres using the new partition.

Step 4. If the centres have not changes then terminate,
otherwise goto Step 2.

Usually the iterations stop when the number of patterns that changed label are insignificant, or simply specify a maximum number of iterations. The second pass of Friedman and Rubin's algorithm is a forcing pass, it perturbs the partition to avoid getting trapped at a local minimum of the criterion function. All patterns in a cluster are transferred to other clusters and the criterion function is recalculated after each test. The best partition found is retained, and the forcing pass is repeated for the next cluster. This process is repeated until convergence is obtained.

Coleman and Andrews (1979) use the criterion $J_4 = \mathrm{tr}B \cdot \mathrm{tr}W$ (maximize) for image segmentation. It was showed that J_4 attains a maximum within the

upper and lower bound of number of clusters (Fig. 3.11), and the maximum of J_4 represent the intrinsic number of clusters. A large number of features were derived using the Sobel edge operator with window sizes of 3 x 3, 7 x 7, 15 x 15. Then the best features were selected by comparing values of the criterion function, and clustering of the best features were accepted as the best results.

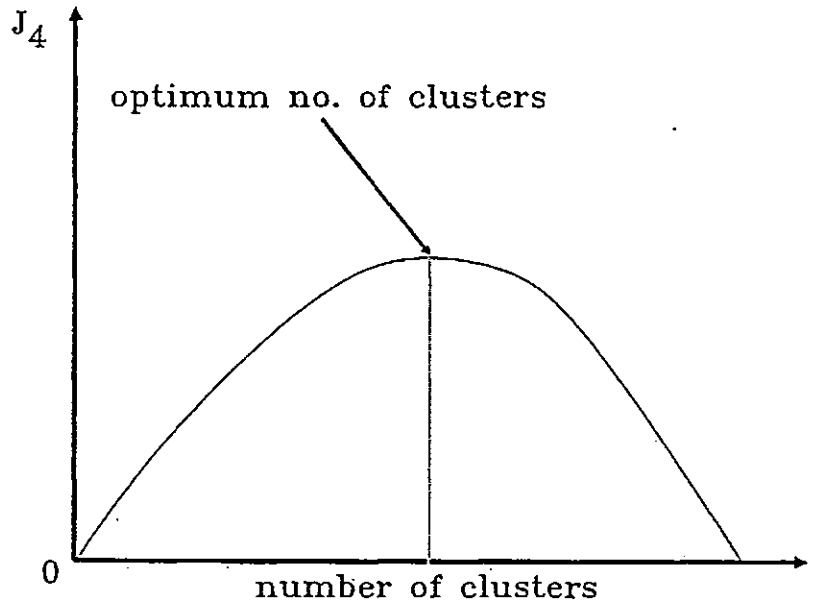


Figure 3.11: General shape of the criterion function J_4.

Koontz and Fukunaga (1972) derived a family of criteria based on minimisation of the error committed in estimating distances between pairs of patterns. Koontz and Fukunaga then showed that the iterative algorithm to minimize the criterion is in fact an iterative use of a decision rule, and the criterion has a very important property that it is valley seeking, so non-linearly separable clusters can also be found. Therefore different criterion functions can produce very different results.

The second problem of iterative clustering is how to limit the search space such that a solution can be found. The K-means algorithm forms the basis of many variations in iterative clustering. Ball and Hall (1967) generalize the K-means algorithm by adding split and merge capabilities. This algorithm is known as

Iterative Self Organizing Data Analysis Techniques A (ISODATA). Those iterative partitional clustering techniques can be generalized as (Jain and Dubes Ch.3 1988):

Step 1. Select an initial set of C cluster centres.

Step 2. Generate a new partition by assigning each pattern to its closest cluster centre.

Step 3. Compute new cluster centres as the centroids of the clusters.

Step 4. Repeat Step 2 and 3 until an optimum value of the criterion function is formed.

Step 5. Adjust the number of clusters by merging and splitting existing clusters or by removing small, or outlier clusters. If any split or merge has occurred goto Step 2 otherwise stop.

The structure of the ISODATA algorithm forms the basis of the second stage of the Global-Local clustering in Chapter 4.

The detailed implementation of these steps involves heuristic procedures (Fromm and Northouse 1976, see also section 4.2.4) and the performance of the algorithm also depends on the choice of distance measure and input parameters which decide the split and merge of clusters. Furthermore, Step 2 and 3 can be changed such that the centres are updated after a pattern has been transferred. However this procedure is susceptible to being trapped at a local minimum, and a further disadvantage of making the results depend on the order of pattern being clustered (Duda and Hart Ch.6 1973). These kind of hill climbing procedures in general do not guarantee global optimization. Koontz et al.(1975) proposed a branch and bound algorithm to limit the search space, by eliminating searches which are unnecessary, this algorithm generates global optimum result. However for large data sets, this approach is still impractical.

In Step 1 an initial set of centres or partitions is need to initialise the iterative clustering algorithm, and it is well know that changing the initialisation generates different results (Wolfe 1970). Good results can only be obtained with initial

partition close to the global optimal. The Global-Local clustering algorithm in Chapter 4 generates an initial partition using a very efficient mode seeking clustering algorithm.

Dynamic Clustering

So far we have only considered using the centre to represent a cluster, and this representation allows us to minimise the sum of square error using very efficient hill climbing techniques. However, it is also possible to use cluster representations other than the centre such that appropriate distance measures can be derived, for example, using a parametric models of the kernel. Diday (1974) generalised the representation of a cluster ω_j by a kernel $K_j = K(\mathbf{x}, V_j)$ with V_j denoting a set of parameters defining K_j. The kernel can be the centre as before, a set of points which are close to the cluster centre (Milgram et al. 1977), or a parametric model (Kittler and Pairman 1985b). The idea of dynamic clustering is also based on iterative optimization of a criterion function. Letting $d(\mathbf{x}, K_j)$ be a similarity measure between the pattern $\mathbf{x}$ and kernel K_j, the criterion function is given by

$$J = \sum_{j=1}^{C} \sum_{i=1}^{n_i} d(\mathbf{x}_i, K_j) \tag{3.51}$$

The algorithm for dynamic clustering is given as:

Step 1. Choose an initial partition of the data set, $\omega_j, j = 1, \ldots, C$,
determine kernel $K_j, j = 1, \ldots, C$ for each cluster.

Step 2. Assign each point $\mathbf{x}_i$ to that cluster ω_j
if $d(\mathbf{x}_i, K_j) = \min_k(\mathbf{x}_i, K_k)$

Step 3. Update the kernel using the new partition,
if kernels $K_j, \forall j$ remain unchange, terminate
the algorithm otherwise goto Step 2.

The kernel function allows a better representation of the cluster. Kittler and

Pairman (1985b) used a multi-variate normal kernel and applied it to cloud images with good results compared with algorithms using Euclidean and Mahalanobis distances. This approach allows the use of a Gaussian model to represent a cluster, and it is used in the Global-Local algorithm in Chapter 4.

Clustering by Mode Seeking

The mixture decomposition method introduced in section 3.4 which assumed a Gaussian model, most mode seeking methods are non-parametric counter part of it. The objective of mode seeking is to identify a unimodal cluster of points in the feature space. Unlike the iterative clustering algorithm, mode seeking algorithms are usually non-iterative and because they are non-parametric only local density estimates are used for clustering.

The simplest way to identify modes in the data is to construct a histogram by partitioning the feature space into a number of non-overlapping regions or cells. Cells with relatively high frequency counts are probably modes and the valleys of the histogram represent the boundaries between clusters. Since these methods are non-parametric they can identify clusters with any shape. This approach is incorporated in the first stage of the Global-Local clustering algorithm presented in Chpater 4, which identifies unimodal clusters by partitioning the multi-dimensional histogram to generate an initial partition for the iterative clustering algorithm in the second stage.

Kittler (1976) proposed a mode seeking algorithm using a Parzen window estimate of the density function with a hypercubic kernel function. This algorithm essentially tries to map the multi-dimensional histogram into a one dimensional sequence of density estimates. A pattern is chosen randomly and corresponds to the first point in the sequence. The second point in the sequence is that pattern which has a maximum density in a hypercubic window around the first pattern. The pattern with the maximum density in the region which is the union of the windows around the first two patterns is selected for the third point. This chain of hypercubes in the data set which will eventually reach the local peak of the

pdf. When the sequence has reached the first peak, the points selected will be with lowest probability density until the valley $\mathbf{x}_v$ is reached. Since all points from the first peak with $p(\mathbf{x}) > p(\mathbf{x}_v)$ have already been selected, the following point $\mathbf{x}_r$, with $p(\mathbf{x}_r) > p(\mathbf{x}_v)$ will belong to the second mode of the pdf. The process continues until all modes have been included in the sequence.

3.5.4 Hierarchical Clustering

In contrast to partitional clustering, in hierarchical clustering, patterns are not transferred between clusters once they are processed. Hierarchical clustering produces a sequence of partitions in which each partition is nested into the next partition in the sequence. If n patterns are partitioned into C clusters, we shall say we are at level k in the sequence when $c = n - k + 1$. Given any two patterns $\mathbf{x}$ and $\mathbf{x}'$ if they are in the same group at level k then they will remain in the same group for all higher levels (Jain and Dubes 1988). An agglomerative algorithm for hierarchical clustering starts with n clusters and each cluster has one pattern. The clusters are merged pair by pair using some distance measure until all patterns are contained in one cluster. The result of this can be represented using a tree structure (Fig. 3.12) which is usually known as a *dendrogram.*

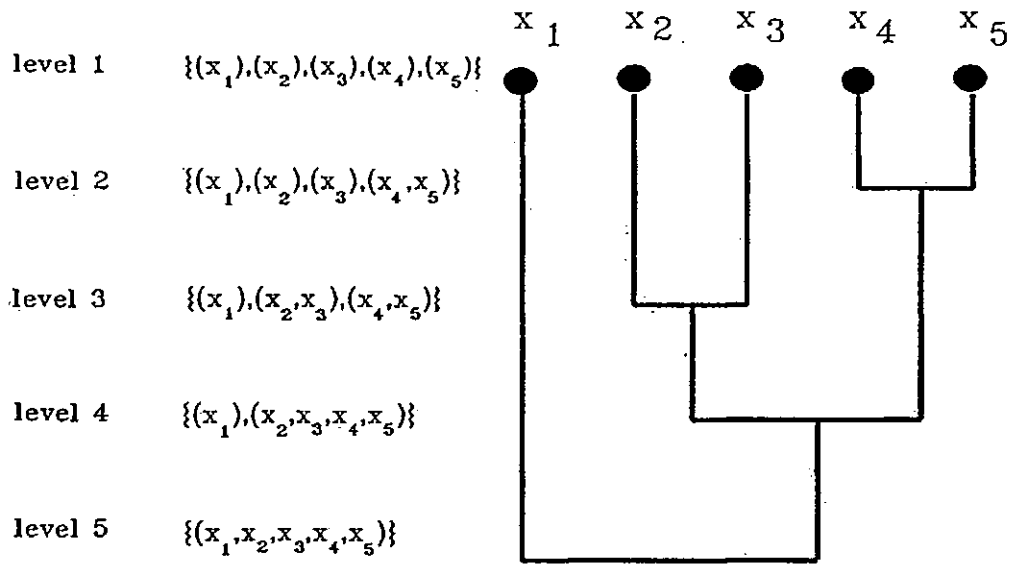


Figure 3.12: Example of dendrogram.

Most hierarchical clustering starts by constructing the similarity matrix, for n patterns there are $n(n-1)/2$ pairs of measure. The size of the similarity matrix limits the number of patterns which can be clustered. So hierarchical clustering algorithms are only limited to clustering a relatively small number of patterns. Anderberg (Ch.6 1973) provided three approaches for implementation of hierarchical agglomerative clustering algorithms, they are the stored matrix, the stored data, and sorted data approach. These approaches all aim to solve the problem of clustering a large data set. Traditionally, hierarchical clustering has found little application on multi-spectral image data, probably because of the storage problem and the complexity in comparing large number of pixels. There are many hierarchical clustering methods, they can be divided into three categories and they are predominantly agglomerative approaches (Anderberg Ch.6 1973):

1. linkage method.

2. centroid method.

3. error sum of square or variance method.

Linkage Methods

The linkage method is conceptually the simplest of all clustering methods. The basic agglomerative clustering method is (Duda and Hart 1973):

Step 1. Let $g = n$ and $\omega_i = \{\mathbf{x}_i\}, i = 1, \ldots, n$

Step 2. If $g \leq C$, stop (C is the number of clusters)

Step 3. Find the nearest pair of distinct clusters, say ω_i and ω_j

Step 4. Merge ω_i and ω_j, remove ω_j and decrease g by one

Step 5. goto Step 2

The following are the most popular distance measures for linkage methods:

$$\delta_{\min}(\omega_i, \omega_j) = \min_{\mathbf{x}\in\omega_i, \mathbf{x}'\in\omega_j} \delta(\mathbf{x}, \mathbf{x}') \tag{3.52}$$

$$\delta_{\max}(\omega_i, \omega_j) = \max_{\mathbf{x}\in\omega_i, \mathbf{x}'\in\omega_j} \delta(\mathbf{x}, \mathbf{x}') \tag{3.53}$$

$$\delta_{\text{avg}}(\omega_i, \omega_j) = \frac{1}{n_i n_j} \sum_{\mathbf{x}\in\omega_i} \sum_{\mathbf{x}'\in G_j} \delta(\mathbf{x}, \mathbf{x}') \tag{3.54}$$

$$\delta_{\text{mean}}(\omega_i, \omega_j) = \delta(\mathbf{m}_i, \mathbf{m}_j) \tag{3.55}$$

$$\text{where } \mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x}\in\omega_i} \mathbf{x} \text{ is the mean of group } \omega_i$$

Each of these four distance measures produces a different clustering method.

Single Linkage Method

If $\delta_{\min}$ is used, the method is called single linkage. The patterns are regarded as nodes and edges are used to connected these nodes in the merging process. When $\delta_{\min}$ is used to measure the distance between two groups the edge that satisfies $\delta_{\min}$ will connect two nodes which are nearest neighbours. Every time an edge is added two distinct clusters are connected, if the process is allowed to continue until there is only one cluster the result is a graph which does not contain any closed loops; in graph theory this procedure generates a tree. If a weight is assigned to an edge and is equal to the distance between the two nodes to which it is connected, then the sum of weight is minimum and so it is known as the a Minimal Spanning Tree

(MST) or Shortest Spanning Tree (SST). Figure 3.13 gives some example of single linkage clustering.

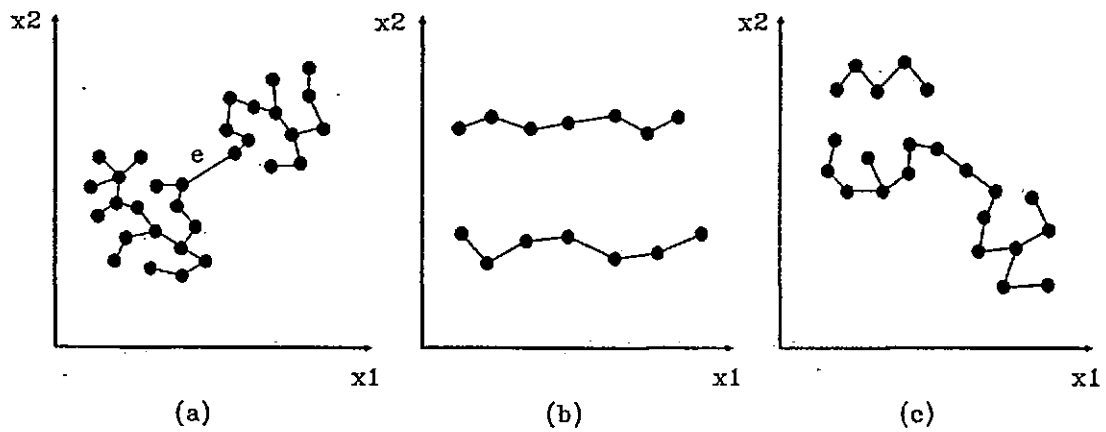


Figure 3.13: Single linkage clustering example.

In Fig. 3.13a there are two compact clusters and the edge e connecting them is the longest, so if the edge is removed we discover the number of clusters exactly, however if these clusters move closer, they can not be discovered by removing the longest edge. So single linkage is only suitable for well separated clusters. On the other hand single linkage is capable of detecting enlongated clusters as in Fig. 3.13b. Unfortunately this property leads to a problem that two very different patterns may be assign to the same group as in Fig 3.13c. This behaviour is often called the "chaining effect".

Complete Linkage Method

When $\delta_{\max}$ is used to measure the distance between clusters, the growth of enlongated clusters is discouraged. This method is called complete linkage because all nodes within a group are linked to each other at some maximum distance. Such a cluster is called a "complete subgraph" in graph theory. Figure 3.14 shows an example of complete linkage clustering. While a single linkage method concentrates on seeking clusters which are isolated from each other, paying no attention to their cohesion; the complete link method concentrates on the internal cohesion of clusters.

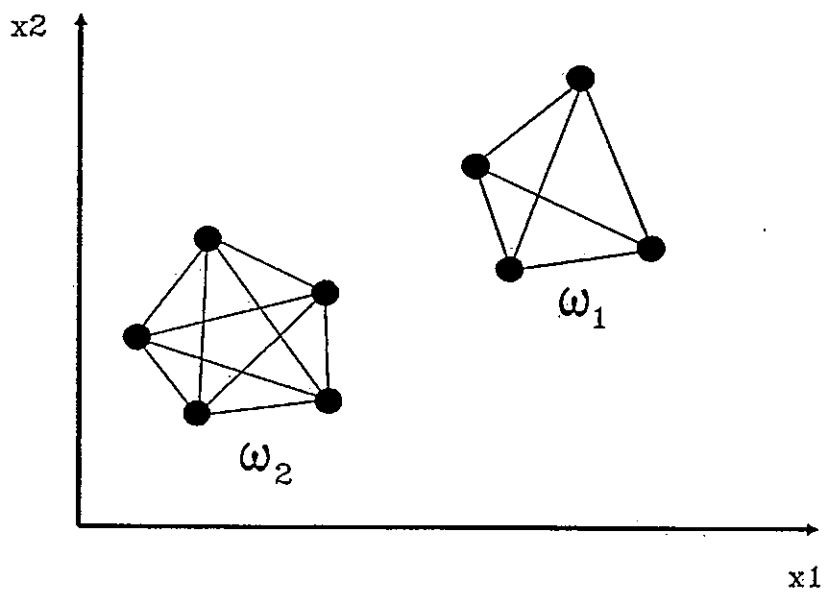


Figure 3.14: Example of complete linkage clustering.

Average Linkage Method

The single linkage and complete linkage methods rely on minimum and maximum distance measure. Because the use of distance can affect the cluster structure to be detected, it is natural to use δ_{avg} and δ_{mean} in the hope that some of the problems such as the chaining effect can be reduced.

The Centroid Method

The Centroid method was proposed by Sokal and Michener (1958). Groups are depicted to lie in the Euclidean space, and are replaced on formation by the coordinates of their centroid. The distance between groups is defined as the distance between the group centroids. The algorithm merges groups according to the distance between their centroids, the groups with the smallest distance being fused first.

Error Sum of Square Method (Variance Method)

Ward (1963) suggested a hierarchical clustering method in which the merges at each stage are chosen so as to maximize an criterion function. The choice of a objective function is a specific problem, since it is well known that there is no universal criteria (Fukunaga and Koontz 1970). Ward used the error sum of squares criterion function, at each stage those two clusters whose merger gives the minimum increase in the total Within Group Error Sum of Squares. Wishart (1969) showed that the Ward algorithm could be implemented by updating a stored matrix of squared Euclidean distances between cluster centroids.

Graph Theoretic Methods

So far we consider hierarchical agglomerative clustering as a process to merge patterns and transform the similarity matrix into a dendrogram, then a clustering is obtained by cutting the dendrogram horizontally. An alternative way for clustering can be obtained by graph theory which is based on a visual perceptual model of clusters.

Graph theoretic clustering is very similar to hierarchical agglomerative clustering. The major difference between the linkage method and the graph theoretic method is how the tree is presented. The dendrogram is presented in an hierarchical order of the merging of objects or groups, and the tree completely spans the data set. In graph theoretic clustering, the tree may or may not span the data set completely, and the tree does not necessary contain information on the merging

order of objects. However, they share a common characteristic that clusters are formed in an hierarchical order.

It was mentioned that the single linkage method is equivalent to the generation of the minimal spanning tree (MST). Efficient algorithms exist for generation of the MST (Prim 1957, Kruskal 1956). Given a MST we can find the clustering produced by the single linkage algorithm. Removal of the longest edge forms two clusters, removal of the next longest edge produces one more cluster and so on. In this way we first merge all patterns into one cluster and divide the cluster into subclusters, so we can perform a divisive hierarchical procedure.

Instead of the removal of an edge based solely on its weight, we can select an edge to remove by comparing the length of edges in its neighbourhood. For example, if the length of an edge is much longer than the mean length of its neighbour, removal of this inconsistent edge may produce two distinct clusters. Zahn (1971) produced an excellent discussion on various method to locate inconsistent edges in spanning tree. Urquhart (1982) used a Gabriel graph and a relative neighbourhood graph as an extension to Zahn's (1971) MST method. A review of applications of graph theory to clustering was given by Hubert (1974).

The property of the spanning tree ability to identify clusters of arbitrary shape has seldom been exploited on image data. If a spanning tree is constructed with the constraint of a pixels' spatial relationship, the resulting segments are found to be superior to other thresholding segmentation techniques in the sense that the segments are less noisy (Morris et al. 1986). This noise insensitive property is particularly important for subsequent shape analysis of the segmentation (Kittler and Pairman 1985a). The new Spatial-Spectral clustering algorithm in Chapter 5 follows this principle and is able to produce clusters with clean boundaries.

Most hierarchical agglomerative and graph theoretic clustering methods can be regarded as stepwise optimal procedures. At every step in the process the two most similar pairs of groups or objects are merged. A criterion function can be derived such that the sum of cost is minimized. In clustering, the cost of merging a pair of groups is often the distance between them. This stepwise

approach form the basis of the Spatial-Spectral clustering algorithm presented in Chapter 5. In the algorithm Graph theoretic clustering has been generalized to the clustering of the spatial space (segmentation) and four distances commonly used in hierarchical clustering have been used to demonstrate the idea of spatial clustering. The principle of agglomerative clustering (stepwise optimization) is also used in the final stage of the Spatial-Spectral clustering algorithm , which clusters segments generated by the Spatial-Spectral clustering.

3.6 Summary

This Chapter has reviewed statistical pattern recognition as a problem of class conditional probability density function estimation. The difference between supervised learning and unsupervised learning is only the presence or absence of labels of the samples. Non-parametric estimators are also introduced which can be used when the pdf to be estimated can not assumed a particular model.

While decision rules based on probability are used to classify patterns in supervised learning applications, the distance function plays a similar role in clustering. Distance is a measure of dissimilarity between patterns. Euclidean distance is a popular distance but it is not capable of discovering clusters with different within cluster variances. So the data is usually normalized such that the variance of each cluster is approximately equal. Since normalization does not always produce the desired effects, the choice of distance would be much easier if a prior knowledge of the structure of the data is available.

Partitional clustering, especially iterative hill climbing type techniques which minimize sum of square error usually imposed a hyperspherical model on the clusters. However, if a suitable distance function is used, clusters can assume different sizes, shapes, covariances and populations. All hill climbing techniques do not guarantee a global optimum and usually only a local optimum is found. Design of hill climbing techniques involves two steps, the first is to define a criterion for optimization, the second is the iteration procedure to optimize the criterion

function. Different criteria will lead to very different algorithms. The sum of square error is the most popular criterion function, but this criterion implies the cluster to be recovered are hyperspherical and does not always give good results.

Another problem of hill climbing techniques is that it requires a set of initial starting points or partitions. Since hill climbing algorithms can be trapped in a local optimum, a set of initial starting points or partition close to the true solution is essential for quick convergence and optimum result.

Clustering by mode seeking is usually non-parametric and therefore suitable for detecting clusters which are irregularly shaped. Local density estimation is generally noisy, so it is practically impossible to establish whether some of the peaks in the estimates correspond to the actual modes in the data. So for the local estimate to be more reliable, a large set of data is required.

The new Global-Local clustering algorithm presented in Chapter 4 uses a cascade of a mode seeking clustering algorithm and an iterative clustering algorithm such that their advantage can be combined. The first algorithm is an efficient histogram clustering (mode seeking) algorithm which generates an initial partition, and the second algorithm is an iterative clustering algorithm which refines the partition using an optimized cluster model.

Hierarchical clustering is closely related to graph theoretic clustering, some of them are capable of detecting non-linearly separable clusters (probably due to the chaining effect). Graph theoretic clustering has a valuable property that it can exploit spatial information within a data set. This property forms the basis of the Spatial-Spectral clustering presented in Chapter 5. However hierarchical clustering techniques have a general disadvantage: since the clustering is constructed in one pass, they can not recover from a poor initial clustering.

Chapter 4

A Global-Local Clustering Algorithm for METEOSAT Imagery

This chapter presents a new and optimised Global-Local clustering algorithm which is a cascade of two clustering algorithms. The first stage of the algorithm generates an initial partition by clustering the multi-dimensional histogram into unimodal regions and the second stage is the optimization of the initial partition using a dynamic clustering algorithm. The objective of the Global-Local clustering algorithm is to eliminate the manual or random selection of an initial partition which is required for all iterative partitional clustering algorithms. It is true that manual selection of an initial partition is time consuming and subjective, on the other hand random selection as shown later in this chapter, always produces sub-optimal results. Therefore it is desirable to generate initial partitions which are close to the optimum partition objectively.

Dynamic clustering is chosen because it is stable for most data types and can be implemented with good efficiency for clustering of image data. The large data set normally found in remote sensing imagery makes it natural to think of clustering as partition of the multi-spectral histogram into unimodal regions.

Section 2.3 reviewed that dynamic clustering is an efficient and reliable method

for unsupervised classification of weather images (Desbois et al. 1982, Seddon and Hunt 1985, Kittler and Pairman 1985b). Among many clustering methods partitional clustering is found to be the most popular in application to image classification (Jain and Dubes 1988).

As mentioned previously many more features (e.g. textural) can be derived from the raw image. However, this work concentrates on the separation of homogeneous cloud clusters which represent layers of cloud corresponding to different altitude or pressure. Results in this work are based on VIS and IR data only, since the goal is to identify clusters correspond to a single level of cloud, and not to try to identify every possible type of cloud. It should be noted that more features in addition to the visible and infrared features are necessary for classification of *all* cloud types (Seddon and Hunt 1985). It is therefore possible that a cluster represents more than one cloud type. This is not a serious problem when the application is mesoscale cloud motion tracking since different cloud types at the same level tend to move with similar speed and direction.

However, additional features are often necessary, because most low cloud tracers are cumulus type and most high cloud tracers are cirrus type (Hubert 1971). An extension of this work can be an investigation of additional features which provide better class discrimination between cloud types appearing at the same level and hence towards the goal of better satellite wind accuracy and automatic target selection. Cloud classification using features in addition to raw data can be found in work done by Parikh & Rosenfeld (1978), Seddon (1983) and Pairman (1985).

4.1 Initial Partitions

A major problem common to all iterative clustering algorithms is the requirement of an good initial partition or centres to converge to a local optimal solution (see section 3.5.3), and the final result is highly dependent on the initial conditions. Since dynamic clustering is a generalization of iterative clustering methods, it

therefore requires a set of initial points or partitions.

Iterative clustering can be used as an approximate estimation of the mixture components (Wolfe 1970). It is well known that multiple solutions exist for all kinds of iterative optimization methods (Ball 1967, Friedman and Rubin 1967). These solutions are due to trapping of the criterion function in a local minimum. Wolfe (1970) suggested that the updating of cluster parameter is responsible for the cause of multiple solutions. So with different initial conditions the algorithm will converge to different local minimum, and one can only try different initial conditions and select one which he thinks is the best solution.

It is interesting to note that the inability of hierarchical clustering methods to transfer pattern vectors is regarded as an disadvantage, while the ability of an iterative clustering algorithm to transfer pattern vectors leads to the problem of multiple solutions. Studies by Desbois et al. (1982), Seddon and Hunt (1985), Kittler and Pairman (1985b) all used iterative clustering algorithms to classify cloud images. Although their algorithms perform satisfactorily, the starting partition was selected manually from the image. This procedure is thought to be tedious and prone to human error, if a large number of images have to be clustered.

Anderberg (Ch.7 1973) reviewed some methods to eliminate the manual selection of initial centres or partition. The initial centres are called seed points because the subsequent result depends on these starting points. If no prior knowledge is given about the data set, the starting points are usually obtained in a random manner. Suppose k seed points are to be chosen, it could be done in one of the following ways:

1. Choose the first k patterns in the data set (MacQueen 1967).

2. Label the patterns from 1 to n and choose the patterns corresponding to k different random numbers in the range 1 to n (McRae 1971).

3. Take any desired partition of the patterns into k mutually exclusive groups and compute the group centroids as seed points (Forgy 1965).

4. Choose seed points which span the data set, that is, most patterns are relatively close to a seed point but the seed points are well separated from

each other (Astrahan 1970).

5. Subjectively choose k subsets of representative patterns from the data set. (Desbois et al. 1982, Seddon and Hunt 1985, Kittler and Pairman 1985b).

An initial partition can be obtained as follows:

1. For a given set of seed points, assign each pattern to the cluster built around the nearest seed point. The seed points remain stationary throughout the assignment of the full data set (Forgy 1965).

2. Assign patterns to the nearest seed points, after a pattern is assigned to a cluster, update the centroid so that it is the true mean vector for all the patterns currently in that cluster (MacQueen 1967).

3. Use a hierarchical clustering to produce an initial partition (Wolfe 1970).

The most promising method to generate initial partition is to use a clustering algorithm. This has the advantage of reducing convergence time and generating better solutions. A hierarchical clustering method can be used for such purpose, but it is impractical, for example, when the data set is an 256 x 256 pixel multi-spectral image. An efficient clustering algorithm with very few or no control parameters is therefore highly desirable.

The most important factors in using an iterative clustering algorithm is the choice of a criterion function, a distance measure and a set of initial centres or partitions. The Global-Local algorithm uses a cascade of two clustering algorithms which combine the advantages of the two algorithms such that the results is better than using either of them alone. The first algorithm is mode seeking and avoids the subjective manual selection of a set of initial partitions by clustering the multi-dimensional histogram. The second algorithm is iterative and uses a criterion which minimize the average error probability and hence a distance measure is derived using a Gaussian model.

The primary task of the first stage of the Global-Local algorithm is to estimate the initial cluster configuration. Given a d band multi-spectral image I, the goal

of clustering is to organize the data into C non-overlapping subsets ω_i , $i = 1,\ldots,C$, such that a clustering criterion $J(\Omega;\mathbf{X})$ evaluated over the partition Ω is optimized. The image I is regarded as a data set $\mathbf{X} = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ where $\mathbf{x}_i$, $i = 1,\ldots,n$ are d-dimensional vectors representing the image points (pixel).

These vectors can include additional features if better cloud classification is required. In this study the dimension d is limited to two since only VIS and IR data are used, although the algorithm is applicable for $d > 2$.

4.2 The Global-Local Clustering Algorithm

The components in the Global-Local clustering algorithm are in Fig. 4.1. The scheme includes a global clustering algorithm which generates an initial partition in a semi-automatic and objective manner, followed by a local clustering algorithm (a classic iterative algorithm) which uses a clustering model tailored to the data to be clustered. The initial partition is then optimized locally in order to converge to a local optimum of the criterion function. This Global-Local approach was also used by Eigen et al. (1974).

The initial partition is generated using a very efficient and simple histogram clustering algorithm. The clustering algorithm is non-parametric and does not require specification of the number of clusters a priori. It was original designed to cluster LANDSAT images, which are very different from METEOSAT images in the area of coverage is much smaller (with a 75m resolution) and therefore class boundaries in the image are rather distinct. While the METEOSAT images have fuzzy boundaries along all cloud types except land and sea in general. In other words, objects in LANDSAT images are represented by pdfs with little overlap while in METEOSAT cloud classes are represented by highly overlapping pdfs. Results show that this algorithm performs unsatisfactorily with METEOSAT images due to the fuzziness of pdfs boundaries. However, some classes (in particular land and sea) which have pdfs with little overlap can always be identified successfully. It is also found that the clustering although far from optimum does serve as very good

initial estimates for the iterative clustering algorithm.

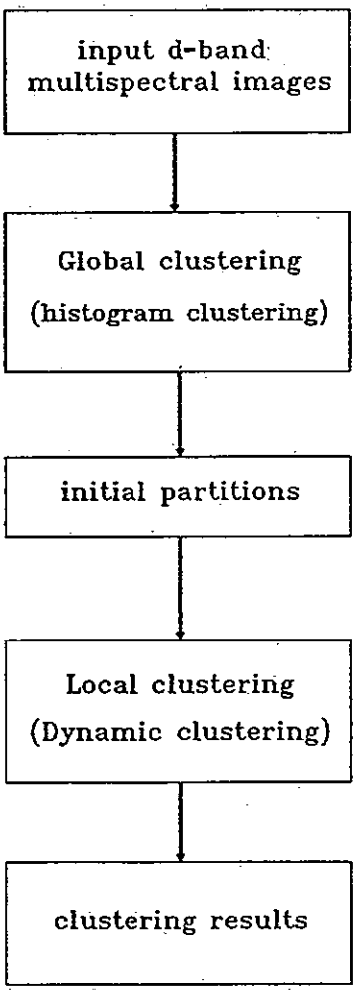


Figure 4.1: A two stages Global-Local clustering algorithm which eliminates manual selection of initial partitions.

4.2.1 The First Stage of the Global-Local Clustering Algorithm

The first stage of the Global-Local clustering algorithm is to generate an initial partition of the multi-dimensional histogram using the histogram clustering algorithm of Narendra and Goldberg (1977). This includes the construction of a multi-dimensional histogram and a non-parametric clustering algorithm which partitions the histogram.

The histogram clustering scheme is very efficient because it clusters the multi-spectral histogram which is usually compressed by varying the cell size (to be discussed later). For most remotely sensed data the number of distinct vectors in the pattern space is usually much less than the number of possible vectors. In case of METEOSAT imagery, the ratio of distinct vector to possible vector is lower than LANDSAT imagery since most cloud clusters have large variance. For example a 256 x 256 two dimensional (VIS & IR) histogram of METEOSAT data (full resolution) has an average possible vector to distinct vector ratio of 6.

There are several characteristics of the histogram clustering scheme which make it suitable for either independent use or generating initial partitions.

1. The number of computations needed to identify clusters in the histogram is much less than that for clustering individual pixels.

2. No parametric assumptions about the underlying probability density.

3. The program is more or less automatic, only two parameters are required, the smoothing and compression parameter (the compression can be fixed if data sets are from the same batch).

4. Number of clusters does not need to be specified a priori.

5. The scheme is very efficient and non-iterative.

Figure 4.2 is the flowchart of the histogram clustering algorithm used by the author.

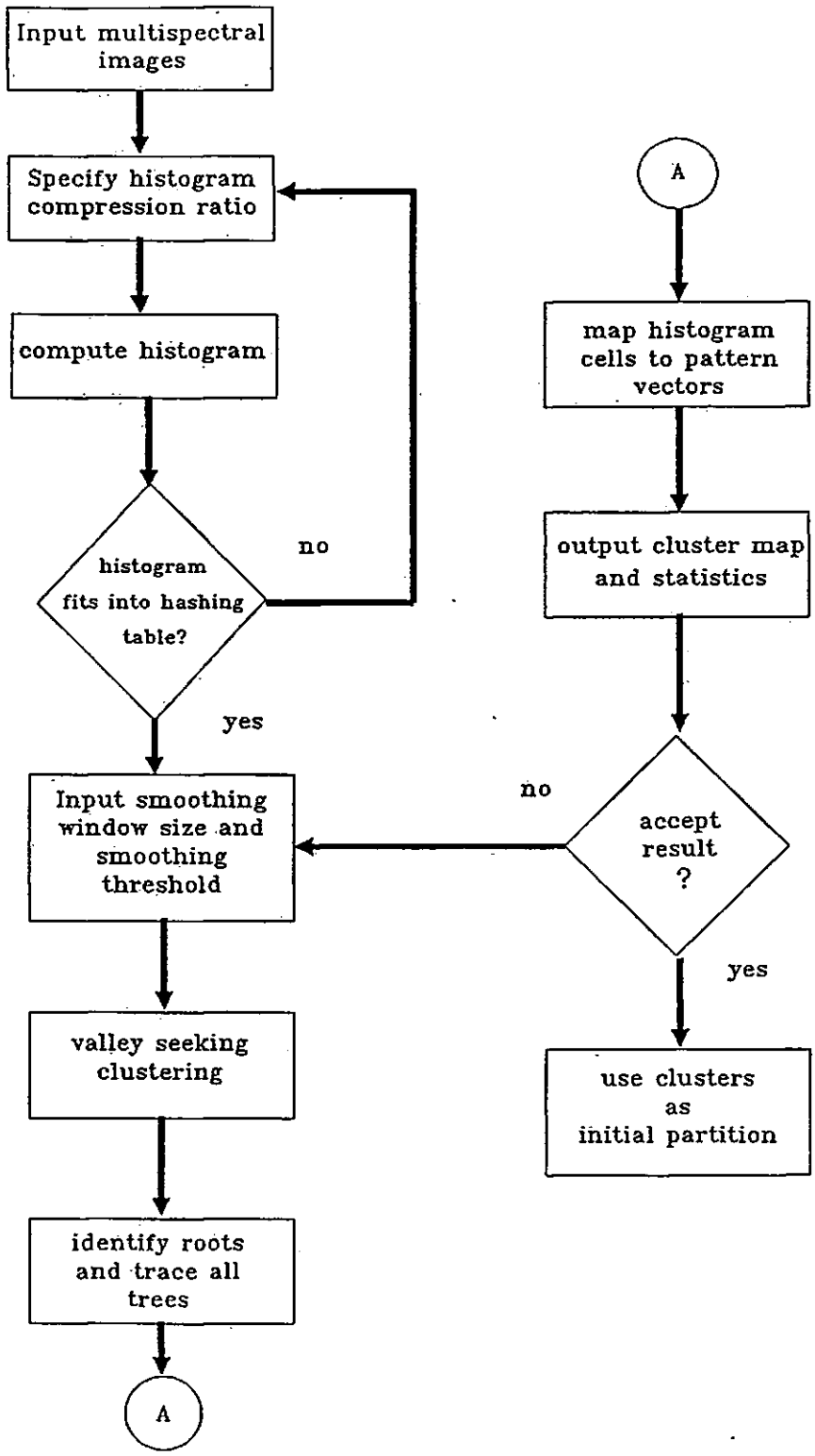


Figure 4.2: Flowchart of the histogram clustering scheme for generation of initial partitions (first stage of the Global-Local clustering algorithm).

Multi-dimensional Histogram

A histogram is a very good estimate of the mixture density provided that the pixel to distinct vector ratio is high (see section 3.3). The histogram can be used directly with a non-parametric clustering algorithm that seeks to partition the histogram into unimodal regions.

The first problem is to construct a multi-dimensional histogram. Although only VIS+IR images are used in this study, the histogram clustering algorithm used is capable of handling more than two bands such that WV band can be used when available. The number of possible vectors is equal to g^d where g is the number of quantization levels and d is the dimension of the pattern space. The number of grey-levels in METEOSAT is $2^8 = 256$, and the maximum number of spectral bands are three, so it requires 256^3 cells to store the histogram. This amount of memory obviously make the histogram construction impractical. Since the number of distinct vectors is much less than the number of possible vectors, a huge saving in memory can result if only the distinct vectors are stored. Then the problem becomes one of how to store and access the distinct vectors in an efficient way.

An efficient way to store a multi-dimensional histogram was suggested by Shlien and Smith (1975). The method is based on a computing technique called *hashing function* $k = h(y)$, where k is the key to the location where vector y is stored and $h(y)$ is the function that maps every vectors to a storage location. The hashing function can be a simple division of a number determined by the distinct vector by a prime number. The prime number is usually equal to the length of the scatter table used to stored the distinct vectors. The scatter table therefore contains all the distinct vectors of the multi-dimensional histogram such that items in the table representing a distinct vector. The table must be slightly bigger that the number of distinct vectors in order to have efficient access. Figure 4.3 is a schematic illustration of how the hashing function operates.

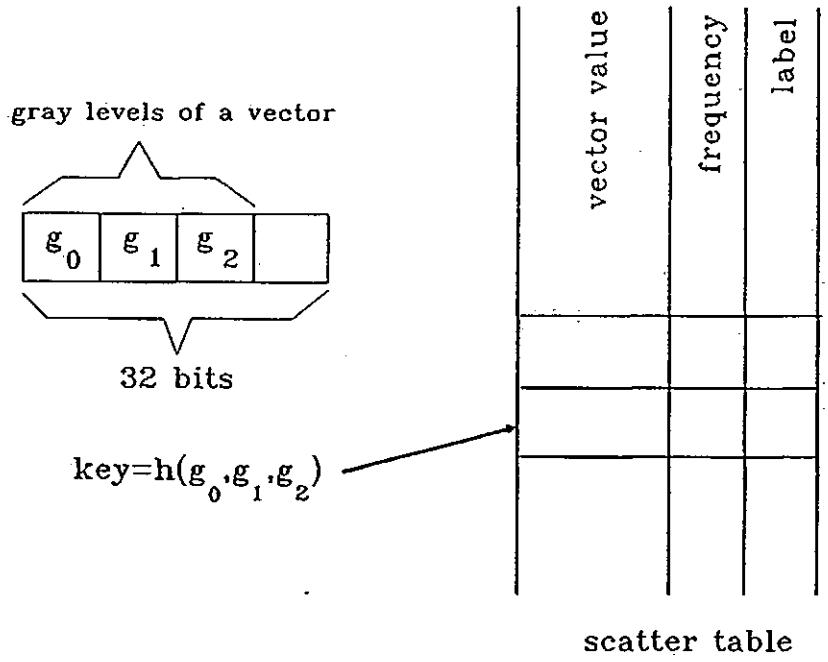


Figure 4.3: Schematic diagram showing how vectors are stored and accessed using a hashing function (Narendra & Goldberg 1977).

In this study the pixel intensity of the three bands (VIS, IR, and WV) are stored in the first three bytes of an 32-bit integer. The key is the remainder after the division by the length of the scatter table. In order that the key are scattered evenly along the table the length of the table is usually chosen to be a prime number.

Wharton (1983) generalised the hashing function such that an arbitrary number of dimensions is allowed. Wharton's hashing function performs a series of remainder operations, one for each feature. The hashing function first completes the remainder of the first feature. This remainder is concatenated with the second feature. The hashing function then computes the remainder of this result, and concatenates it to the third features. This process continues until all the features have been considered. The final remainder is used as the location key.

It is possible that more than two different vectors will be mapped to the same location, in this case a collision occurs. The problem is easily solved by a trial and

error process which iteratively applies a different hashing function until either a vector match or an empty location is found. As the scatter table is gradually filled the chance of collision increases, so the scatter table should be bigger than the number of distinct vectors (the actual size depends on hashing function efficiency).

A non-parametric Valley Seeking Clustering Algorithm

The multi-dimensional histogram is clustered using the valley seeking algorithm described by Koontz et al.(1976). This algorithm is non-iterative and non-parametric, it requires no starting classification, is valley seeking and is capable of detecting generally shaped clusters. The algorithm uses both graph theory and local density estimation to cluster the histogram cells.

The cells will be clustered by constructing directed trees on them. A directed graph is a set of nodes $\{V\}$ and edges (arcs) $\{E\}$, each edge connects an initial node v to a final node v'. A directed path is a set of edges $e_1, \ldots, e_n$ from v to v', if v is the initial node of e_1, v' is the final node of e_n and the final node of e_k is the initial node of e_{k+1} for $k = 1, 2, \ldots, n-1$.

A directed tree is a directed graph with a unique node v, called the root such that (Fig. 4.4):

1. Every node $v \neq r$ is the initial node of exactly one arc.

2. r is the initial node of no arc.

3. There is no directed path from a node v to itself (i.e. no cycles).

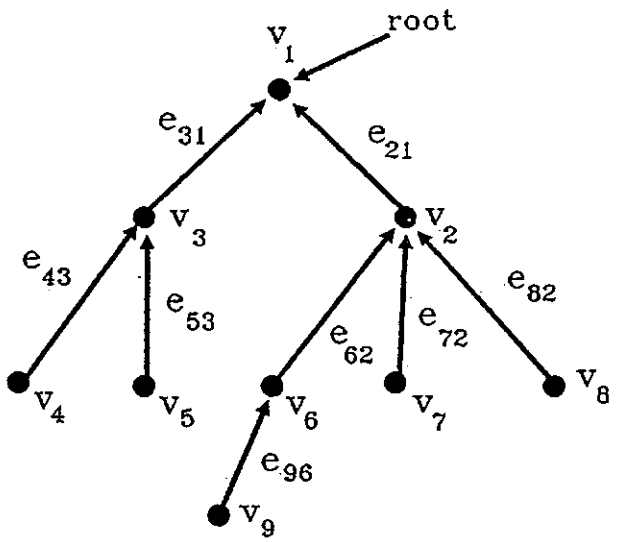


Figure 4.4: Example of a directed tree.

The final node of an edge is called the parent node of the initial node. Therefore the root node has no parent node.

The pattern set (distinct vectors) $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is regarded as a set of nodes, i.e. $v_1 = \mathbf{x}_1, \ldots, v_n = \mathbf{x}_n$. The goal is to construct one or more directed trees on the set of nodes. The clustering procedure is governed by a set of rules for assigning a parent node to each $\mathbf{x}_i$. Any nodes without a parent node become roots. And no cycle should exists in the directed graph. The number of roots will determine the number of directed tree's hence the number of clusters. Every $\mathbf{x}_i$ will belong to only one directed tree (non-overlapping subsets).

The nodes are linked by an edge according to the density gradient between two nodes. The density gradient between two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as

$$g_{ij} = \frac{p(j) - p(i)}{d_{ij}} \tag{4.1}$$

where $p(j)$ is the local density at $\mathbf{x}_j$ and d_{ij} denote the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. $p(j)$ can be estimated using any non-parametric methods. In the author's algorithm, the local density is given by the histogram estimator. Koontz et al.

used a Parzen estimator with a rectangular window,

$$p(i) = \frac{1}{n}\sum_{j=1}^{n} K(\mathbf{x}_i, \mathbf{x}_j) \tag{4.2}$$

where

$$K(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} (2h)^{-d} & \text{if } d(\mathbf{x}_i, \mathbf{x}_j) \leq \theta \\ 0 & \text{otherwise} \end{cases}$$

where $2h$ is the width of the window, n is the total number of patterns. Let k_j be the number of patterns falling within the window centred on $\mathbf{x}_j$. Since $\frac{1}{2hn}$ is constant, and only relative value is important, we can denote $p(j) \stackrel{\text{def}}{=} k_j$. The window width is determined by an input parameter θ such that $h = \theta$. The effect of changing the value of θ is such that the larger the value of θ the smoother the estimates are, and the best value is found by trying several settings. The relative density is then estimated by counting the number of patterns $\mathbf{x}_k$ which satisfy $d_{ik} \leq \theta$ and $k \neq i$. Define a set which represent the neighbourhood of vector $\mathbf{x}_k$, $\eta_\theta^i = \{\mathbf{x}_k | d_{ik} \leq \theta, k \neq i\}$, and $k_i = \#\{\eta_\theta^i\}$ The density gradient is then given by

$$g_{ij} = \frac{k_j - k_i}{d_{ij}} \tag{4.3}$$

Koontz et al. also used the k-nn estimator for estimation of $p(j)$, and in this case the parameter is the number of k nearest neighbour and the density is given by the volume enclosing the k nearest neighbours.

The multi-dimensional histogram is clustered using the following procedure:

Step 1. If $k_i = 0, \mathbf{x}_i$ is a root.

Step 2. If $k_i > 0$, compute g_i^* where

$$g_i^* = \max_{\mathbf{x}_j \in \eta_\theta^i} g_{ij}$$

Step 3. If $g_i^* < 0, \mathbf{x}_i$ is a root.

Step 4. If $g_i^* > 0$, the parent node of $\mathbf{x}_i$ is $\mathbf{x}_k$,
where $g_i^* = g_{ik}, \mathbf{x}_k \in \eta_\theta^i$
(ties are resolved arbitrarily)

Step 5. If $g_i^* = 0$, consider the set $\Pi_i = \{\mathbf{x}_j | \mathbf{x}_j \in \eta_\theta^i, g_{ij} = 0\}$.
Eliminate from Π_i any node $\mathbf{x}_j$ such that there is
a directed path from $\mathbf{x}_j$ to $\mathbf{x}_i$. If the resulting
Π_i is empty, then $\mathbf{x}_i$ is a root. Otherwise, the parent
node of $\mathbf{x}_i$ is $\mathbf{x}_k$ such that $d_{ik} = \min_{j \in \Pi_i} d_{ij}$.
(ties are resolved arbitrary)

This procedure of assigning parent nodesensure that if $\mathbf{x}_j$ is uniform within a region such that $g_i^* = 0$ and Π_i has more than one elements, then we have to make sure no cycleswill result by making $\mathbf{x}_j$ a parent node of $\mathbf{x}_i$. This is achieved by eliminating all nodes in Π_i that have a directed path to $\mathbf{x}_i$ to the closest neighbour in Π_i. If $g_i^* > 0$ we are certain that no directed path exists from $\mathbf{x}_k$ to $\mathbf{x}_i$ since $p(k) > p(i)$.

The directed trees obtained have a uniquely identified root, and all the nodes belonging to a tree can be identified by tracing from the root. Since any node $\mathbf{x}_i$ can only associate with one directed tree, the cluster defined by the directed tree is non-overlapping. Consider any two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ connected by a directed edge. Suppose $\mathbf{x}_i$ is the parent node of $\mathbf{x}_j$ then $p(i) \geq p(j)$, if we delete from the tree some node $\mathbf{x}_l$ such that $p(l) < t$ for some $t \geq 0$. If any nodes remain, they also constitute a directed tree whose root is identical to that of the original tree, thus these trees are unimodal directed trees. It is also true that every root has the highest density in the tree it belongs to. Thus, each cluster is uniquely associated with a mode.

Koontz et al. also shown that the asymptotic properties of the algorithm, if $\nabla^2 p(\mathbf{y})$ is bounded and $p(\mathbf{y}) > 0$ for all $\mathbf{y}$ such that $||\mathbf{x}_i - \mathbf{y}|| \leq \theta$ then

$$\lim_{\theta \to 0} \left\{ \lim_{n \to \infty} g_i^* \right\} \longrightarrow || \nabla p(\mathbf{x}_i)|| \tag{4.4}$$

where $\nabla p(\mathbf{x}_i)$ is the density gradient at $\mathbf{x}_i$.

This property means that the ancestors of $\mathbf{x}_i$ follow the line of steepest ascent, and therefore is a hill climbing property. The other property of the algorithm is that the cluster boundaries pass through the valleys in $p(\mathbf{x})$, thus the algorithm is also termed a valley seeking algorithm.

Implementation of the Histogram Clustering Algorithm

It is now clear that the histogram estimator can be used to substitute the Parzen or k-nn estimator in the valley seeking algorithm. In the case of histograms, every distinct vector $\mathbf{x}_j$ is represented by a histogram cell and the frequency in the cell is the local density estimates $p(\mathbf{x}_j)$ of $\mathbf{x}_j$. We have already obtained the density estimate of every pattern, all we need is to construct a directed tree on the histogram cells. The distinct vectors will be denoted as nodes in the tree. Because of the way distinct vectors are arranged in the histogram, it is only necessary to consider the nearest neighbours when comparing the density gradient. An example using a two dimensional histogram is shown in Figure 4.5.

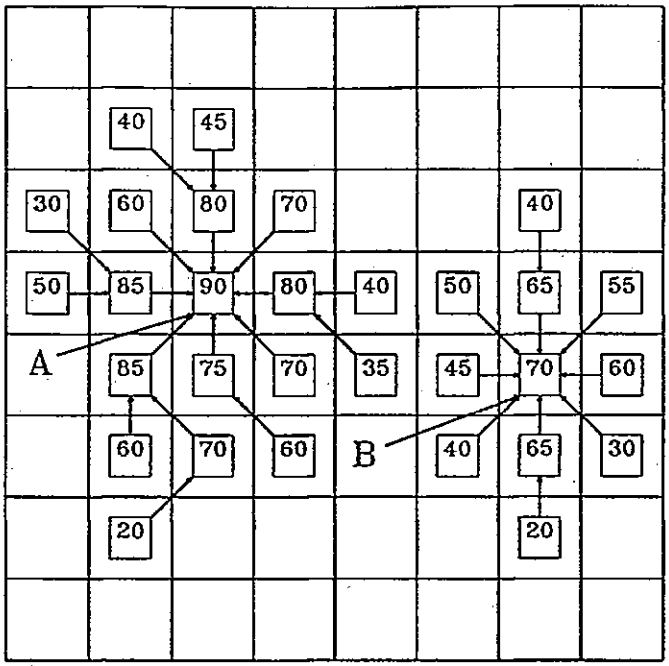


Figure 4.5: A two dimensional illustration of the histogram clustering scheme. Cell A and B are roots.

The original valley seeking algorithm requires a distance threshold parameter because the Parzen estimator was used. For histograms this parameter is no longer required because the density estimate only depends on the frequency of a cell. Thus, theoretically, no clustering parameter is necessary.

In Fig. 4.5 every cell has eight nearest neighbours, and for d-dimensional histogram the number of nearest neighbour is equal to $3^d - 1$. For $d = 3$ there are 26 nearest neighbours and an edge can only be constructed after density gradient of all neighbours have been computed, which require $n(n-1)/2$ operation to search for the neighbours. This constitutes the major computational effort involved in valley seeking clustering.

The number in the box of Fig. 4.5 represents the frequency of that cell. A directed edge is placed between each vector and the immediate neighbour which is in the direction of the maximum positive density gradient. The density gradient is given by

$$g_{ij} = \frac{f_j - f_i}{d_{ij}} \tag{4.5}$$

$$d_{ij} = \sum_{k=1}^{d} |x_{ik} - x_{jk}| \tag{4.6}$$

where f_j is the frequency of cell j and d_{ij} is the City Block distance between cell i and j.

Fig. 4.5 also shows clearly the idea of a unimodal directed tree with their roots representing the maximum local density, and two trees are separated by the boundary between the valley of two adjacent densities. Fig. 4.6 shows a case when the local density in a region is uniform.

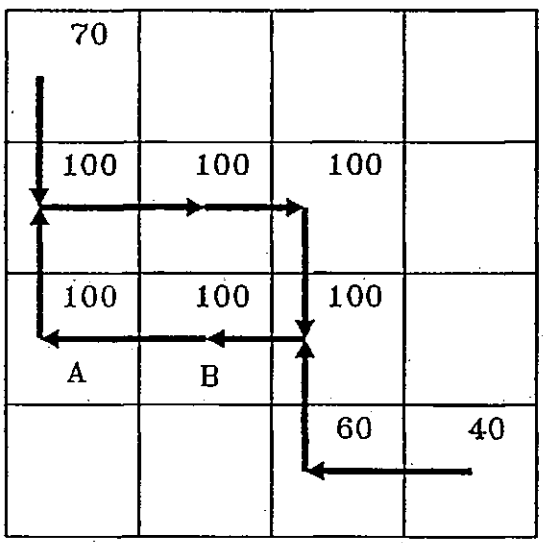


Figure 4.6: Illustration of how to avoid a directed cycle in a region of uniform density. If A and B are linked then a directed cycle results.

When regions of uniform density occur, the maximum density gradient will be zero, and directed cycle can be formed therefore in Step 5 of the algorithm, we have to find out all nodes that have a directed path connected to the current node

$\mathbf{x}_i$ and make the neighbour node that has no directed path as the parent node of $\mathbf{x}_i$, $\mathbf{x}_i$ is a root. Since ties are resolved arbitrary, the directed tree generated is not unique and depends on the order of the node begin processed.

Once the directed trees have been generated the nodes within a tree can be traced from the root using computing techniques of graph theory. There are two graph traversing algorithms, namely the depth-first search and breath-first search, depth-first search is used by the author. After all nodes in a tree are traced they are labelled as in the same cluster. The graph traversing algorithm is very efficient and the search depends linearly on the number of nodes.

Neighbourhood Computation

Narendra and Goldberg (1977) suggested a scheme for finding the nearest neighbours which required p x $2n$ instead of $n(n-1)/2$ operations, where p is the number of nearest neighbours. Their scheme is to first linearly order the concatenated values of the vectors (See Fig. 4.3). It is noted that all the possible neighbours can be represented by an offset vector O with respect to the current vector. For example (0,0,1,0) denotes the neighbour which differs with the one vector only in the third position. Taking advantage that the ordered nature of the list, the ith neighbour of $\mathbf{x}_{k+1}$ (i.e. $O_i + \mathbf{x}_{k+1}$) if it exists, must occur lower down in the ordered list than the ith neighbour of $\mathbf{x}_k$ (i.e. $O_i + \mathbf{x}_k$). Hence it is only necessary to search the ordered list once to find the ith neighbours of all n vectors. The list is stored in a file on the bulk storage device, it has been found that this does not perform as efficient as it was supposed, so a different method is adopted.

The scheme used by the author is to generate a list of offset vectors. The ith neighbour of $\mathbf{x}_k$ (i.e. $O_i + \mathbf{x}_k$) can be found directly using the hashing function, this scheme is simpler to implement and requires only n x p operations when compared to Narendra's scheme.

Wharton (1983) suggested a even more efficient scheme using a K-dimensional binary tree, he showed that the actual number of neighbours A is much less than

the number of nearest neighbours per cell p. By constructing a $K-d$ tree of the distinct vectors the search time can be further reduced. However, it is noted that for dimension $d \leq 4$ the number of actual neighbour is about 50% but considering the overhead of constructing the tree, a simple scheme such as the one used by the author should perform equally well.

Histogram Smoothing and Compression

The success of the histogram method for density estimation depends on a large pixel to vector ratio, and existence of distinct valleys. However, even with a large data set, the resulting estimate is locally noisy. This may due to the sensor noise, quantization error, atmospheric interference and other problems in remote sensing. If the histogram is clustered without any preprocessing, trivial clusters may be generated. This situation is illustrated in Fig. 4.7.

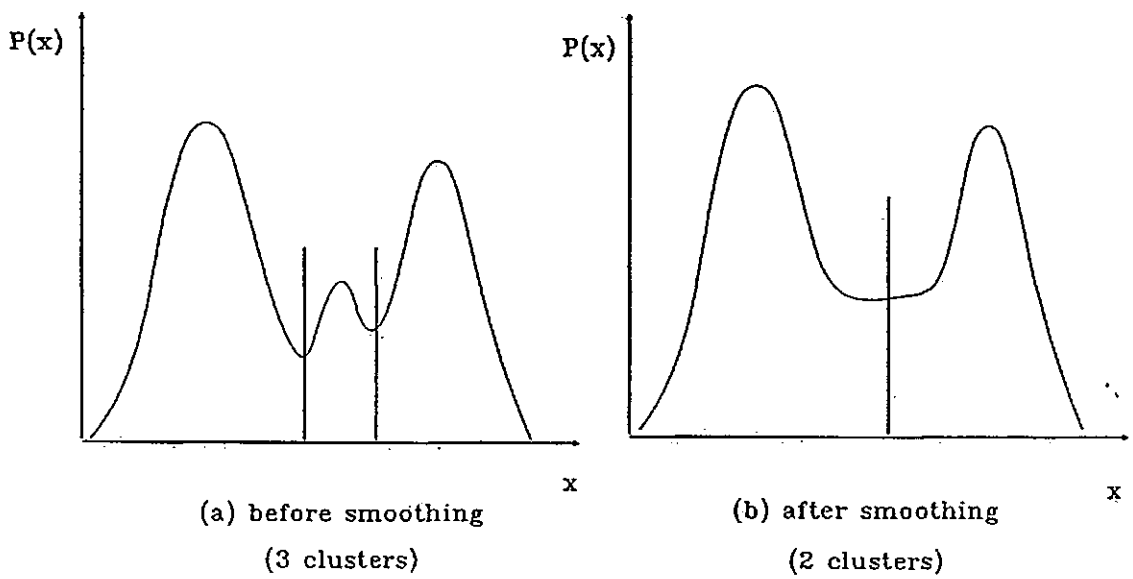


Figure 4.7: The effect of noise in the unsmoothed histogram will lead to generation of trivial clusters.

Two methods of histogram smoothing is possible. The histogram discussed so far is constructed using the highest resolution of the grey level. The first

possible smoothing method is to requantize the grey levels. For example, we can compress 256 grey levels to just 128 levels thus achieving a compression of factor 2^d. This compression of grey level has two effects. Firstly the effective pixel to vector ratio has increased, which improves the smoothness of the local density estimates. Secondly, the number of distinct vectors have also been reduced, which in turn allows the histogram to be clustered with less time. Unfortunately, a problem arises due to requantization. Since volume of the cell increase by 2^d, if we chose to double the compression ratio each time. The higher the compression ratio, the more patterns fall within a cell, if any cell is classified as ω_i then all patterns within it will have the same label, obviously the total error will increase with the compression ratio. However, if the clustering is used to provide initial estimates the error can be corrected in the local clustering stage.

The second method for histogram smoothing is by replacing a cell estimate by the local average. The average window is adjustable, and the possible window sizes are 3 x 3, 5 x 5,.... Therefore every cell estimate is replaced by the average of all neighbours inside the window. The neighbours are searched using the offset vector list. It is noted that the resulting histogram depends on the order of the histogram cell being processed. By choosing different window sizes we can control the number of clusters obtained.

Both requantization and averaging methods are employed in the Global-Local clustering scheme. It is found that clusters corresponding to high and middle cloud have relatively low pixel to vector ratio and large variance. Those cloud clusters are difficult to partition using the valley seeking algorithm, because the boundaries are not well defined in these regions. Regions with well defined peaks have more reliable estimates, so smoothing should only applied to region with low density, where the estimates are more prone to noise.

4.2.2 Starting Partition

There are two methods to obtain an initial partition from the histogram clustering. Given a partition generated using the histogram clustering algorithm, the first

method is to use the mean of each initial cluster and assigns pixel to the nearest centre using squared Euclidean distance, and then parameters of the Gaussian models are estimated using these partitions, this is the method employed in this study. The second method uses the initial partition directly to estimated the Gaussian model's parameters. It is noted that the histogram may generates some trivial clusters which are ignored.

The second stage of the Global-Local clustering algorithm can split or merge clusters according to some control parameters. If split or merge occurred, the newly formed cluster parameters have to be estimated, they are obtained by re-assigning all pixel to the nearest cluster mean, alternatively we can only estimate parameters of the newly formed clusters by reassigning only pixels previously in the split or merged clusters without changing the other cluster models.

In this study all pixels are reassigned after split or merge. This method is chosen because it is found that clustering has better stability than using the second method. This can be explained that some of the clusters generated by the histogram clustering algorithm is very different from the Gaussian model used in the second stage, therefore the estimates do not fit the model well and require several iterations to recover. Assigning all pixels to their nearest centres provide a better estimate of the cluster models because it assumes all clusters having a Gaussian model with unity variance and same a priori probability. However, one disadvantage of the first method is that clusters which are adjacent and with small variance (e.g. different type of land or sea) are assigned to the same cluster. Fortunately, this confusion is not a serious problem, because land and sea clusters are not used in tracking of cloud motion.

4.2.3 The Second Stage of the Global-Local Clustering Algorithm

The second stage of the Global-Local clustering algorithm is to optimize the initial partition generated by the histogram clustering. This is achieved by using a dynamic clustering algorithm. The complete Global-Local clustering scheme is

shown in Figure 4.8.

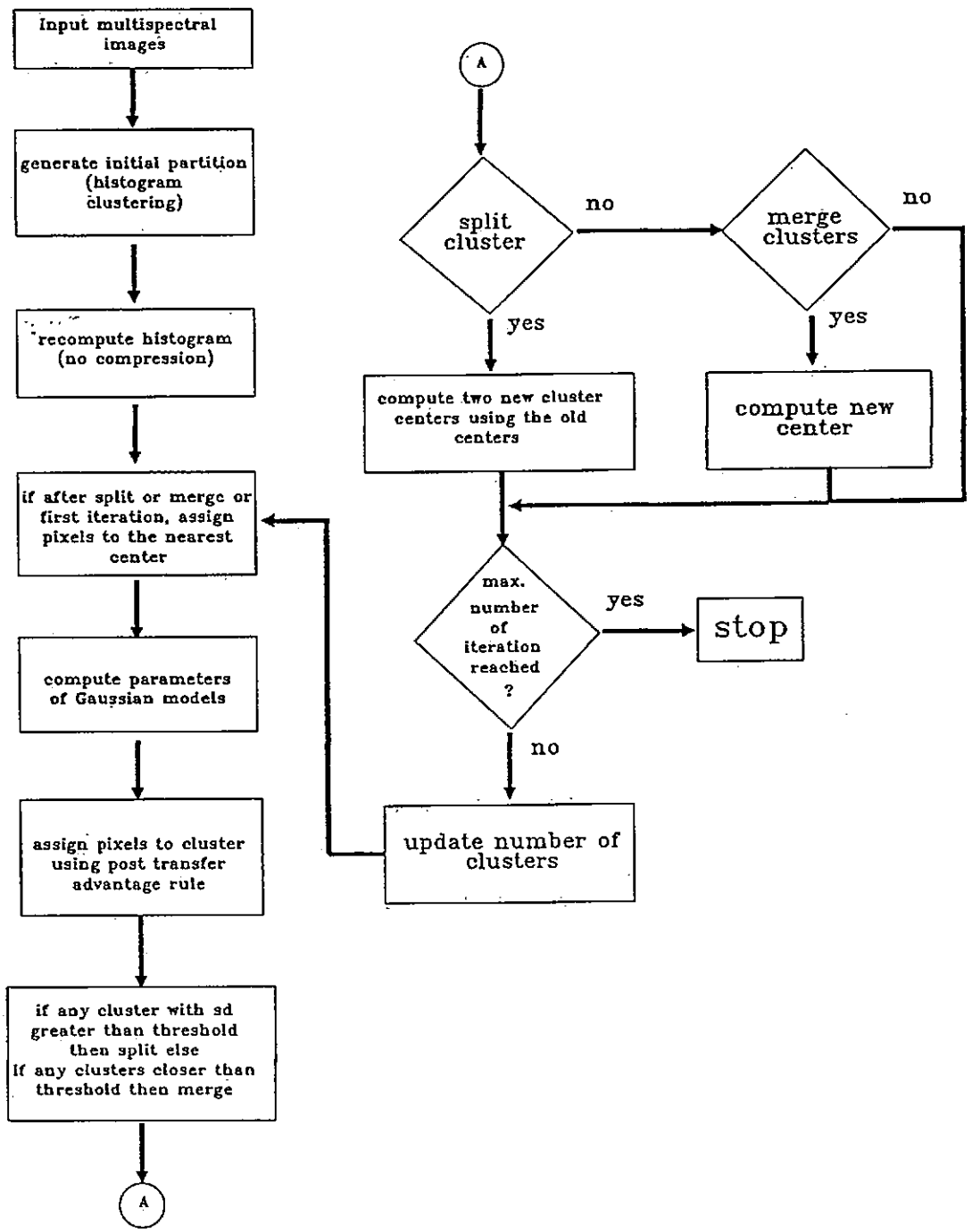


Figure 4.8: The Global-Local Clustering scheme.

It was mentioned in section 3.5.3 that the most important factors in using an iterative clustering algorithm are the choice of a criterion and a distance function. A criterion based on minimizing the average error probability and a distance function derived from a general Gaussian model described by Kittler and Pairman (1985b) have been adopted in the second stage of the Global-Local algorithm.

Dynamic Clustering

Dynamic clustering was introduced in section 3.5.3, which allows clusters to be represented using parametric models. In this study a general Gaussian model has been chosen and found to perform reliably on METEOSAT images.

Problems on Choosing a Distance Measure for METEOSAT Images

It can be seen in histogram of METEOSAT imagery (Fig. 1.1) the size and variance of each cluster in the feature space is very different. The squared Euclidean distance assumes all clusters to have equal covariance and is obviously not satisfied by the data. A better model is the Gaussian model which allows clusters to have different variance. Another important point that can be observed in the histogram is that most clusters have very different size, which is equivalent to different a priori probability $P(\omega_i)$. It is well known that most distance measure are not able to identify clusters with great difference in cluster *variance* and *size* (Symons 1981). Some of the problem on using distance was discussed in section 3.5.2 where only well separated clusters were considered. It was suggested that difference in cluster variance can be compensated by normalising the data and then using Euclidean or Mahalanobis distance for clustering. However, all of these methods (Kittler and Pairman 1985b) implicitly assume that clusters to have the same a priori probability.

The problem of using squared Euclidean distance is mainly due to the lack of a priori knowledge of the underlying data structure. It is then safe to assume all clusters have similar population and variance, therefore clusters generated using the cluster mean model tends to have equal size and variance. This point was cited

by Kittler and Pairman (1985b) i.e. "If a similarity measure becomes the key point of a clustering philosophy, then indeed it is inappropriate to argue that a data point has a greater affinity to one cluster than another, just because the former one is larger in size." Similar comment was also given by Symons (1981). This idea is illustrated in Figure 4.9. Two clusters with equal variance but different size, the optimum decision boundary is the one which minimizes the Bayes error probability.

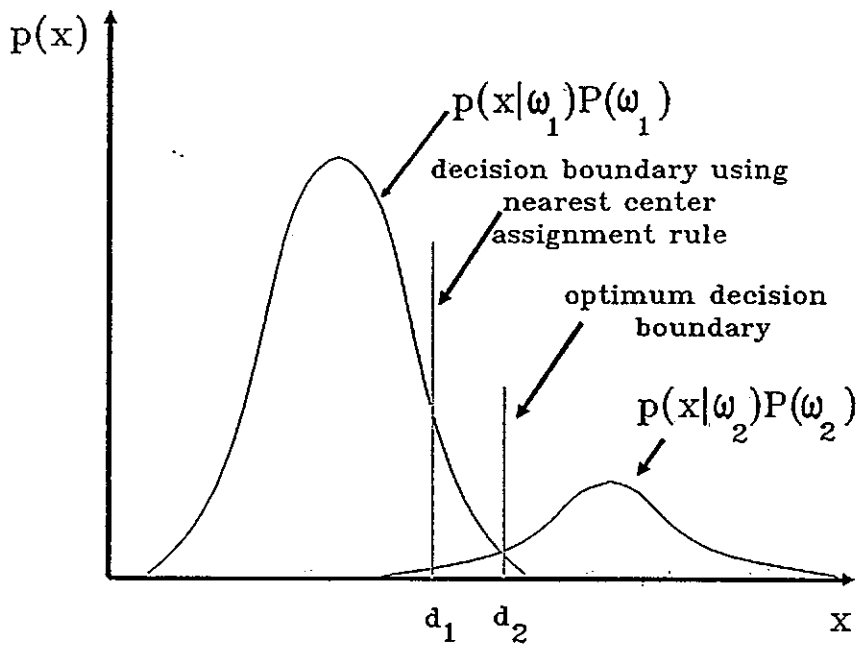


Figure 4.9: Use of the cluster mean model to cluster data with equal variance but different population.

When using the cluster mean model the decision boundary d_1 is well within the larger cluster in order to satisfy the equal variance assumption. This problem can be solved by a distance measure which will produce a decision boundary close to the optimum.

A Model that Allows Different Cluster Size and Variance

A clustering criterion based on probability of classification error should allow a better distance measure to be derived for a specific cluster model. We have already seen that the squared Euclidean distance has an underlying pdf model which is normally distributed with the identity covariance matrix. The advantage of using a clustering criterion based on minimizing classification probability error allows cluster sizes to be taken into account.

A criterion based on the concept of minimizing the average error probability is used (Kittler and Pairman 1985b) :

$$J = \left[\prod_{i=1}^{C} \prod_{j=1}^{n_i} p(\mathbf{x}_j \mid \omega_i) P(\omega_i) \right]^{\frac{1}{n}} \tag{4.7}$$

Similarly we can define a criterion without population weighting as

$$J' = \left[\prod_{i=1}^{C} \prod_{j=1}^{n_i} p(\mathbf{x}_j \mid \omega_i) \right]^{\frac{1}{n}} \quad \text{where } \mathbf{x}_j \in \omega_i \tag{4.8}$$

Assume the class conditional pdf is modelled by a Gaussian pdf.

$$p(\mathbf{x}) = (2\pi)^{-d/2} |\Sigma_i|^{-1/2} \exp\left[-\tfrac{1}{2} (\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) \right] \tag{4.9}$$

where μ_i is the mean of class ω_i.

Four clustering criteria can be obtained from eqn. 4.7 and 4.8 assuming that $p(\mathbf{x} \mid \omega_i)$ is a Gaussian pdf with a identity or general covariance matrix.
Neglecting the constant $(2\pi)^{-\frac{d}{2}}$ and substitute eqn. 4.9 gives

$$J = \left[\prod_{i=1}^{C} \prod_{j=1}^{n_i} \frac{n_i}{n} |\Sigma_i|^{-\frac{1}{2}} \exp\left[-\frac{1}{2} (\mathbf{x}_j - \mu_i)^T \Sigma_i^{-1} (\mathbf{x}_j - \mu_i) \right] \right]^{\frac{1}{n}} \tag{4.10}$$

The first two criterion assume all clusters have the same size, i.e. $\frac{n_1}{n} = \frac{n_2}{n} = \ldots = \frac{n_C}{n}$ and $\sum_{i=1}^{C} \frac{n_i}{n} = 1$. We have

$$J' = \left[\prod_{i=1}^{C} \prod_{j=1}^{n_i} |\Sigma_i|^{-\frac{1}{2}} \exp\left[-\frac{1}{2} (\mathbf{x}_j - \mu_i)^T \Sigma_i^{-1} (\mathbf{x}_j - \mu_i) \right] \right]^{\frac{1}{n}} \tag{4.11}$$

taking the minus logarithm we have

$$J_1 = \frac{1}{n}\sum_{i=1}^{C}\left[\sum_{j=1}^{n_i}(\mathbf{x}_j-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_j-\mu_i)+n_i\log|\Sigma_i|\right]$$
$$= \frac{1}{n}\sum_{i=1}^{C}\sum_{j=1}^{n_i}\left[(\mathbf{x}_j-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_j-\mu_i)+\log|\Sigma_i|\right] \quad (4.12)$$

J_1 is called the Gaussian model with equal a priori probability. If all covariances are equal to the identity matrix, then J_1 becomes

$$J_2 = \frac{1}{n}\sum_{i=1}^{C}\sum_{j=1}^{n_i}(\mathbf{x}_j-\mu_i)^T(\mathbf{x}_j-\mu_i) \quad (4.13)$$

J_2 is the well known nearest mean model.

For the next two criteria it is assumed all classes have unequal a priori probability, and taking the minus logarithm of J gives:

$$J_3 = \frac{1}{n}\sum_{i=1}^{C}\left[\sum_{j=1}^{n_i}(\mathbf{x}_j-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_j-\mu_i)+n_i\log|\Sigma_i|-2n_i\log\frac{n_i}{n}\right]$$
$$= \frac{1}{n}\sum_{i=1}^{C}\sum_{j=1}^{n_i}\left[(\mathbf{x}_j-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_j-\mu_i)+\log|\Sigma_i|-2\log\frac{n_i}{n}\right] \quad (4.14)$$

J_3 is using the population weight general Gaussian model.

If all covariance matrix are equal to the identity matrix I, we have

$$J_4 = \frac{1}{n}\sum_{i=1}^{C}\sum_{j=1}^{n_i}\left[(\mathbf{x}_j-\mu_i)^T(\mathbf{x}_j-\mu_i)-2\log\frac{n_i}{n}\right] \quad (4.15)$$

J_4 is called the population weighted nearest mean model.

By comparing with eqn. 3.51 four distance measures could be derived from the minimum error probability concept based on Gaussian model. They are

$$d_1 = (\mathbf{x}_j-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_j-\mu_i)+\log|\Sigma_i| \quad (4.16)$$

$$d_2 = (\mathbf{x}_j-\mu_i)^T(\mathbf{x}_j-\mu_i) \quad (4.17)$$

$$d_3 = (\mathbf{x}_j-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}_j-\mu_i)+\log|\Sigma_i|-2\log\frac{n_i}{n} \quad (4.18)$$

$$d_4 = (\mathbf{x}_j-\mu_i)^T(\mathbf{x}_j-\mu_i)-2\log\frac{n_i}{n} \quad (4.19)$$

where Σ_i is given by eqn. 3.14

Both $(\mathbf{x}_j - \mu_i)^T \Sigma_i^{-1} (\mathbf{x}_j - \mu_i)$ and $\log |\Sigma_i|$ terms are used very often in clustering Diday and Simon (1976) and they enforce a minimum variance solution, the term $-2 \log \frac{n_i}{n}$ would tends to allocate all points into one cluster, it enforces a minimum entropy solution. Therefore the population weighted Gaussian model is composed of three terms and a solution correspond to a compromise between the three terms. It is well known that d_1 and d_2 favour equal cluster sizes because they neglect cluster size and designed to minimize the total variance. The use of d_4 alleviate the problem of different cluster size only to an extend that the cluster satisfy the identity covariance matrix exactly. Clustering using d_3 perform best and allows cluster to have different size and variance.

It is noted that finding a suitable distance measure is a very difficult task. This is partly due to the fact that every distance imposes its own model on the data to be clustered, and if the data does not satisfy the assumption the distance measure fails to recover the data structure. To be on the safe side one should always uses as much a priori knowledge as possible when selecting the distance measure.

The estimates of the parameters μ_i, $\Sigma_i$and$P(\omega_i)$ are obtained by the maximum likelihood method, and are the standard ones

$$P(\omega_i) = \frac{n_i}{n} \tag{4.20}$$

$$\mu_i = \mathrm{m}_i = \frac{1}{n_i} \sum_{n_i} \mathbf{x} \tag{4.21}$$

$$\Sigma_i = \frac{1}{n_i} \sum_{\omega_i} (\mathbf{x}_j - \mathrm{m}_i)^T (\mathbf{x}_j - \mathrm{m}_i) \tag{4.22}$$

where $\forall \mathbf{x} \in \omega_i$

So far four distance functions have been derived using a Gaussian model with different degree of simplification. The performance of d_1, d_2, d_3, and d_4 has been studied by Kittler and Pairman (1985b), and it was found that d_3 is the best when used to cluster cloud images. Therefore d_3 (i.e. J_3) is used in the Global-Local clustering algorithm.

Assignment Rules

In Figure 4.8 it is shown that the second stage of the Global-Local algorithm is based on a classical ISODATA frame work. After a distance function and a cluster model have been determined pixels are assigned to kernels using an assignment rule.

There are two assignment rules which are used extensively. The first rule (cluster affinity rule) is traditionally used in the ISODATA algorithm, the second rule (post transfer advantage rule) is used in the K-means algorithm (see section 3.5.3). The major difference between them is the model parameters updating order. The first rule updates the model parameters after all points have been assigned to their nearest cluster, while the second rule update the model as soon as a point is transferred. The second difference is that the second rule is a stepwise optimal assignment rule which guarantee the transfer of any points will decrease the criterion function's value if the criterion is to be minimized.

Specifically, the cluster affinity rule is

$$\text{assign } \mathbf{x}_i \in \omega_k \text{ to } \omega_j \text{ if}$$

$$d(x_i, K_j) = \min_r d(x_i, K_r) \tag{4.23}$$

the post transfer advantage rule is

$$\text{assign } \mathbf{x}_i \in \omega_k \text{ to } \omega_j \text{ if}$$

$$\frac{n_j}{n_j + 1} d(\mathbf{x}_l, \mu_\mathrm{j}) = \min_{r \neq i} \frac{n_r}{n_r + 1} d(\mathbf{x}_l, \mu_\mathrm{r}) < \frac{n_i}{n_i - 1} d(\mathbf{x}_l, \mu_i) \tag{4.24}$$

where n_r is the number of data points currently belongs to cluster ω_r.

Kittler and Pairman (1988) argue that using the point to cluster affinity rule may not guarantee even reaching a local minimum of the criterion function. So they suggest using the post transfer advantage rule (eqn 4.24) as a possible solution. However, Duda and Hart (Ch.6 1973) said that the post transfer advantage

rule with immediate parameters updating is more susceptible to being trapped at a local minimum, and it has the further disadvantage of making the results depend on the order in which the points are selected.

Regardless of the above statement Kittler and Pairman (1988) proposed a post transfer advantage rule for the population weighted Gaussian model. The reassignment rule is based on the nearest mean post transfer advantage rule (eqn. 4.24). Usually patterns in imagery data will have multiple occurrence, as for multispectral image being shown in previous section. The single point assignment rule should be modified to take account of the multiple occurrence of patterns. Assignment rule eqn. 4.24 becomes

$$\text{assign } \mathbf{x}_l \in \omega_i \text{ into } \omega_j \text{ if}$$

$$\frac{n_j}{n_j + k} d(\mathbf{x}_l, \mu_j) = \min_{r \neq i} \frac{n_r}{n_r + k} d(\mathbf{x}_l, \mu_r) < \frac{n_i}{n_i - k} d(\mathbf{x}_l, \mu_i) \tag{4.25}$$

where k is the frequency of $\mathbf{x}_l$.

The post transfer advantage rule is a stepwise optimal rule such that the criterion function is guarantee to reduce by an amount of ΔJ. The assignment rule for J_3 (eqn. 4.14) is given by

$$\text{assign } \mathbf{x}_l \in \omega_i \text{ to } \omega_j \text{ only if}$$

$$\begin{aligned} d(\mathbf{x}_l, K_j) &= \min_{r \neq i} d(\mathbf{x}_l, K_r) < \log |\Sigma_i| \\ &\quad - \frac{n_i - k}{k} \log \left[1 - \frac{k}{n_i - k} \Delta(\mathbf{x}_l, K_i) \right] \\ &\quad - 2 \log \frac{n_i}{n} - (d+2) \frac{n_i - k}{k} \log \frac{n_i}{n_i - k} \qquad (4.26) \\ \text{where } d(\mathbf{x}_l, K_r) &= \log |\Sigma_r| + \frac{n_r + k}{n} \left[1 + \frac{k}{n_r + k} \Delta(\mathbf{x}_l, K_r) \right] \\ &\quad - 2 \log \frac{n_r}{n} + (d+2) \frac{n_r + k}{k} \log \frac{n_r}{n_r + k} \qquad (4.27) \\ \text{and } \Delta(\mathbf{x}_l, K_i) &= (\mathbf{x}_l - \mu_i)^T \Sigma_i^{-1} (\mathbf{x}_l - \mu_i) \qquad (4.28) \end{aligned}$$

when the cluster size n_r is large eqn. 4.26 can be approximated by

assign $\mathbf{x}_l \in \omega_i$ to ω_r if

$$\log|\Sigma_r| + \Delta(\mathbf{x}_l, K_r) - 2\log\frac{n_r}{n} = \min_j \left\{\log|\Sigma_j| + \Delta(\mathbf{x}_l, K_j) - 2\log\frac{n_j}{n}\right\} \quad (4.29)$$

It is noted that eqn. 4.29 is equivalent to the maximum likelihood decision rule (eqn. 3.6) for normally distributed data.

Therefore the point to cluster affinity rule can be regarded as the approximation of the post transfer advantage rule. It is noted that the post transfer advantage rule can be used without immediate updating as well. The immediate updating formulas is given in Appendix B.

It is difficult to compare the two rules without resort to a reference cluster map. Since the post transfer advantage rule has not been implemented by Kittler and Pairman, and implementation of this rule requires a multi-dimensional histogram to be constructed, it is therefore used conveniently in the Global-Local algorithm.

4.2.4 Other Features of the Global-Local Clustering Algorithm

Most clustering algorithms do not take advantage of the multiple occurrence of patterns found in images. This property has been exploited in the Global-Local algorithm such that the efficiency is much higher than ordinary algorithms. In Figure 4.8, after the histogram clustering has completed, the histogram is recomputed without compression, so if a distinct vector are assigned to a cluster the other copies will follow.

It is found that in multi-spectral image data the pixel to vector ratio can be very large, i.e. the number of distinct pattern vectors is significantly less than the total number of pixels. The clustering efficient can be greatly increased if the multiple occurrence can be exploited. A histogram approach can do just that,

once a full resolution histogram of the data is constructed and the application of any assignment rule is straight forward. If accuracy is to be sacrificed for efficiency the histogram can be compressed as discussed in section 4.2.1.

After the histogram is constructed the distinct vectors are clustered using the population weighted Gaussian model (J_3) with split and merge capabilities. The split and merge routine is essential because the number of clusters usually is not known a priori and the initial partition generated by the histogram clustering algorithm may not be optimal if heavily overlap clusters exist. So a group having large variance is probably a group of two or more clusters and should be split, and groups which is very close probably belongs to the same cluster.

The frame work of the Global-Local clustering algorithm is based on the classical ISODATA algorithm and is given in section 3.5.3. Split and merge capabilities are the main features of ISODATA, and these features were originally designed to tackle the problem of bad initial centres (randomly chosen centres).

Parameters which control the histogram clustering are:

1. β_c = histogram compression ratio.
2. β_s smoothing threshold, cell with frequency higher than this will not be smoothed.
3. w_s = smoothing window size.

Parameters which control the dynamic clustering are:

1. K = number of cluster desired.
2. θ_n = minimum number of vectors in a cluster, if a cluster has number of vectors less than this it is removed and vectors are reassigned to the nearest cluster mean.
3. θ_s = maximum standard deviation of a cluster allowed.
4. θ_c = minimum distance allowed between two clusters.
5. I = maximum number of iterations.

The basic program flow of the Global-Local clustering algorithm is shown in Fig. 4.8. The details implementation of the algorithm is:

Step 1. Input d band images.

Step 2. Specify the parameters to control the dynamic clustering (defined above).

Step 3. Specify the histogram dimension and compression ratio.

Step 4. If the histogram does not fit into the scatter table, goto Step 3 and double the compression ratio β_c.

Step 5. Specify the smoothing threshold β_s and window size w_s.

Step 6. Smooth the histogram adaptively using moving average method.

Step 7. If operator does not accept clustering result goto Step 5 and adjust the value of the smoothing threshold, and window size.

Step 8. Ignore trivial clusters generated by histogram clustering using a threshold n_t, clusters with points less than n_t will not be included in the initial partition.

Step 9. Recompute histogram without compression.

Step 10. Set $iter = 1$.

Step 11. If $iter = 1$ compute the mean of each cluster in the initial partition, assign distinct vectors to the nearest mean, else use the mean of current clusters, assign distinct vectors to the nearest mean.

Step 12. Compute model parameters of each cluster (covariance matrices, mean vectors and a priori probability).

Step 13. Assign distinct vectors to kernels using the post transfer advantage rule (eqn. 4.26).

Step 14. Test and remove any clusters attract less than θ_n points, if any cluster is removed goto Step 11.

Step 15. Compute the intra cluster distance (see Appendix C) of each cluster, and then compute the mean intraset distance.

Step 16. Compute the inter cluster distance d_{ij} between ω_i and ω_j as $\frac{n_i n_j}{n_i + n_j} \sum_{k=1}^{d} \frac{(\mathrm{m}_{ik} - \mathrm{m}_{jk})^2}{\sigma_{ijk}}$

n_i is the number of points in ω_i and σ_{ij}

is the common variance of ω_i and ω_j

All pairwise distance are computed.

Step 17. If *iter* = I stop, else continue.

Step 18. If *iter* is even or number of clusters greater than 2 x K, test if there are any clusters to be merged.

Merge any pair of clusters if their inter cluster distance is less than θ_c.

Suppose ω_i and ω_j is to be merged,

the new mean is $\mathrm{m}_l = \frac{1}{n_i + n_j}[n_i \mathrm{m}_i + n_j \mathrm{m}_j]$.

If merge occurred reduced number of cluster by one,

increment *iter* by 1 and goto Step 11.

Step 19. Test the standard variation of each cluster, if one of its variable is greater than θ_s and intra cluster distance is greater than the mean intra cluster distance or current number of clusters greater then $K/2$

split the cluster into two.

Suppose the kth variable in ω_l exceeds the threshold,

the two new means m_i and m_j are

given by $\mathrm{m}_{ik} = \mathrm{m}_{lk} + \gamma \mathrm{m}_{lk}$

and $\mathrm{m}_{jk} = \mathrm{m}_{lk} - \gamma \mathrm{m}_{lk}$, where

$0 < \gamma < 1$ and $\gamma = 0$ for variables do not exceed

θ_s. If split occurred, increment *iter* by 1 and

goto Step 11.

Step 20. If *iter* $< I$ goto Step 12, else stop.

4.3 METEOSAT Data Used for Algorithm Evaluation

Six sets of METEOSAT (VIS+IR) images consisting of different weather systems have been chosen to test the algorithms developed. All images are a 512 x 512 pixel subframe (data window) within a B format image. The coordinates of the top left corner is (330, 60) (pixel, line) respectively. The processing window is a 256 x 256 pixel window locates at (100, 0) in the data window (see Figure 4.10—4.15). The dates on which the data were taken are:

No.	Date	raw image	clustered result
1.	5th March 1991	Fig. 4.10	Fig. 4.16
2.	8th March 1991	Fig. 4.11	Fig. 4.17
3.	11th March 1991	Fig. 4.12	Fig. 4.18
4.	15th March 1991	Fig. 4.13	Fig. 4.19
5.	18th March 1991	Fig. 4.14	Fig. 4.20
6.	20th March 1991	Fig. 4.15	Fig. 4.21

Each set of images contains three pairs of VIS+IR images. The first pair of images were received at 11:30 GMT, and the time separation of each pair was 30 minutes. The numerical weather predication of wind fields (850mb, 500mb, 250mb with temperature) covering the process windows are provided by the Meteorological Office Bracknell. The east-west and north-south resolution of the wind fields were 0.9375° and 0.75° respectively.

The longitude and latitude of the process window being (moving from the top right hand corner in a clockwise direction) $61°N$ $5.5°E$, $40°N$ $3.4°E$, $40°N$ $10.7°W$, $61°N$ $17.1°W$. It covers the whole of the United Kingdom, Ireland, France Netherlands, Belgium and Northern Spain.

4.3.1 Description of the Imagery

The surface chart of the above images can be found in Appendix G. The images for the 5th March show frontal cloud associated with an occluding depression centred to the north of Scotland. The associated cold front stretches from the North Sea across southern England, with relatively cloud free air behind. The deep frontal cloud is embedded in a predominantly southwesterly flow.

On the 8th March a well-occluded low pressure is centred over Cornwall and the METEOSAT images show a classic spiral cloud pattern.

A major low pressure complex is situated in mid-Atlantic on the 11th March and there is a south-westerly turning north westerly flow in the upper troposphere. Ahead of the warm and occluded front it is possible to identify the south-easterly winds associated with the polar trough. The clouds here are middle level and therefore are moving under the polar front cloud.

An occluded cold front is found on 15th March, the system is in its developing stage. The cold air mass can be clearly identified in the relatively cloud free area behind the front. The front stretches from northern Scotland down to southern France.

A developing frontal systems is found on 18th March, a occluded front is stretching from north of Ireland down to Bay of Biscay. A relatively cloudless

area is found in the cold air area behind the front. A large area (top right) of thick cirrus was driven by the polar front which can be clearly identified in the infrared image. Behind this is a big lump of warm frontal cloud including a mixture of cumulonimbus and nimbostratus.

A warm and cold front are found on 20th March. The warm front stretches from southern Ireland down into the Alantic Sea and the cold front stretches from southern Ireland across to Germany.

4.4 Clustering Results

The six sets of images described above have been used to test the clustering approaches. Three clustering algorithms have been compared:

[A1] Global-Local clustering algorithm (A cascade of histogram clustering and dynamic clustering algorithms (using J_3 eqn. 4.14)).

[A2] Dynamic clustering (using J_3 eqn. 4.14 and a randomly selected initial partition).

[A3] Histogram clustering algorithm.

The author has tried a procedure using Astrahan's (1970) method: k seed points are selected such that they are evenly spanned in the pattern space, but without considering the density of the patterns chosen. The random initial partition is generated by specifying a distance d_t, then the first center C_1 is chosen as the overall mean of the data, the second center C_2 is chosen such that $d(C_1, C_2) \geq d_t$ and for the ith center $d(C_i, C_j) \geq d_t$, $j = 1, \ldots, n, i \neq j$, where n is the required number of starting partition. The initial partition is generated by assigning every pixel to the nearest center $C_i, i = 1, \ldots, n$.

In order to compare the two comprehensive algorithms (A1 and A2), the same number of clusters (6 to 7) are generated in each case. The clustering statistic is shown in Table 4.1— 4.6. The clustered images and different projections of the two dimensional histogram are shown in Figure 4.16— 4.21, each colour in

the histogram represents a cluster. Clustering algorithms (A1, A2) are allowed to run for 11 iterations. It is noted that the split and merge capabilities allow a bad initial partition to recover, and this is true for both approaches (A1 and A2) because the histogram clustering does not always give the best initial partition.

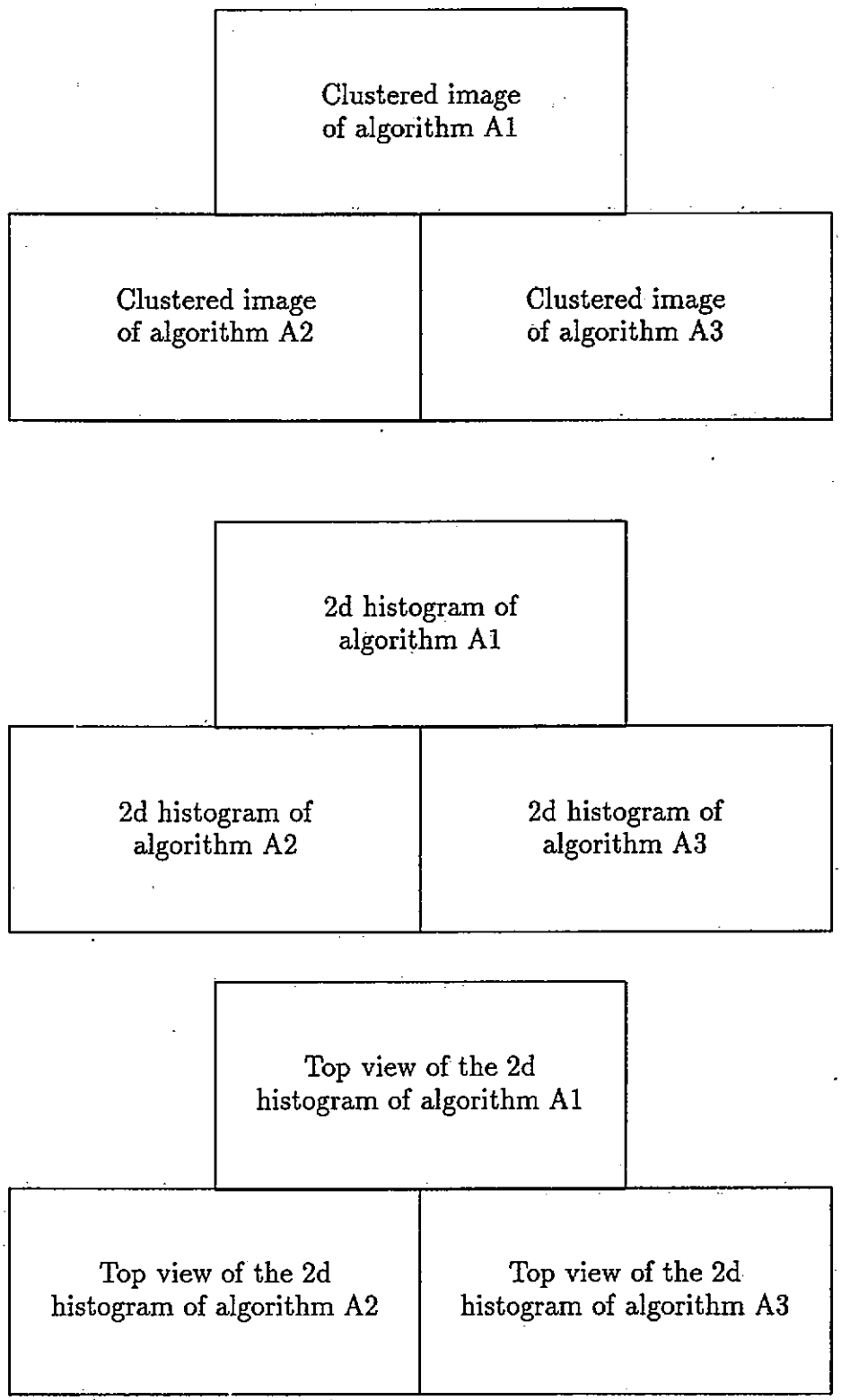


Figure 4.16: Clustering results of 5th March images

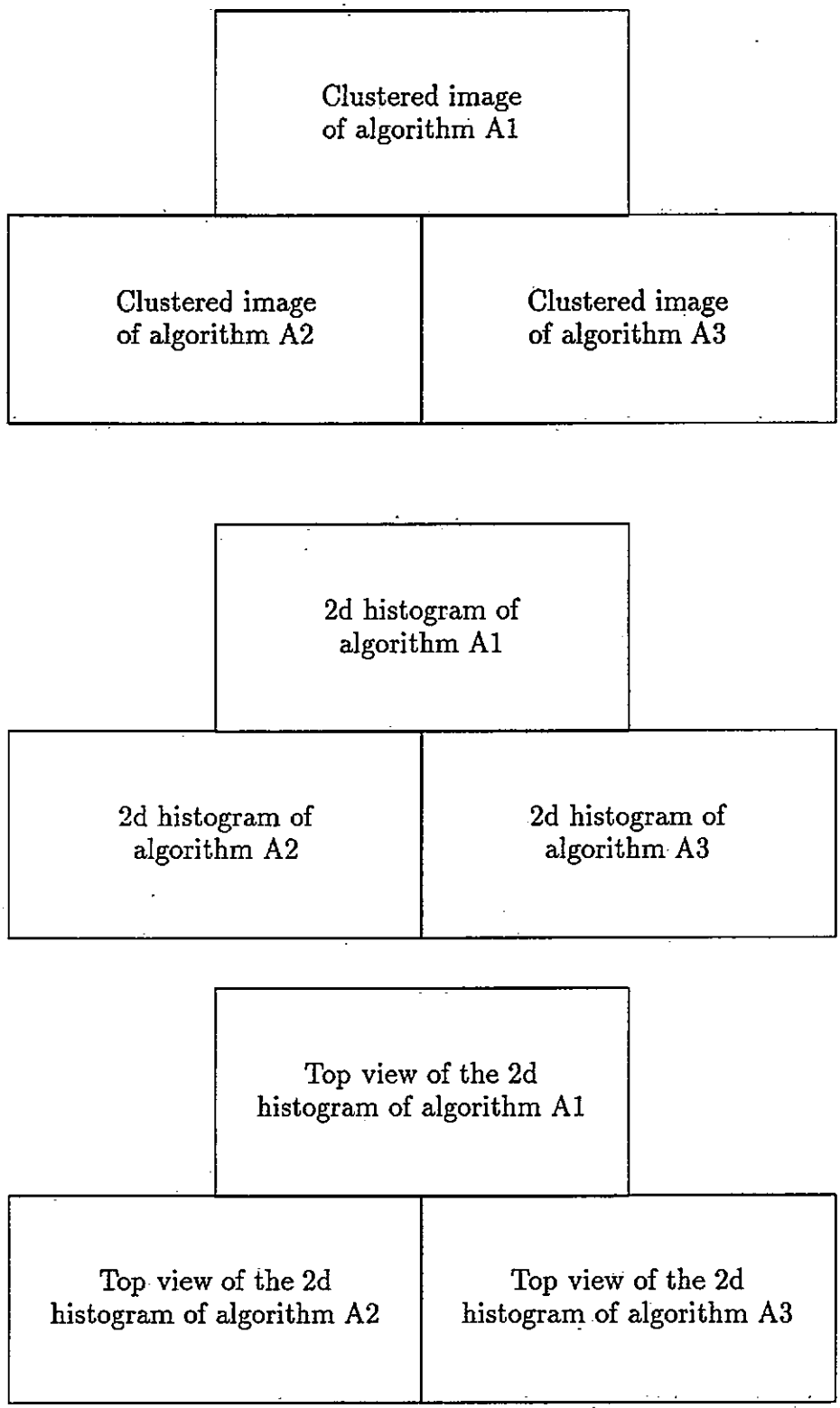


Figure 4.17: Clustering results of 8th March images

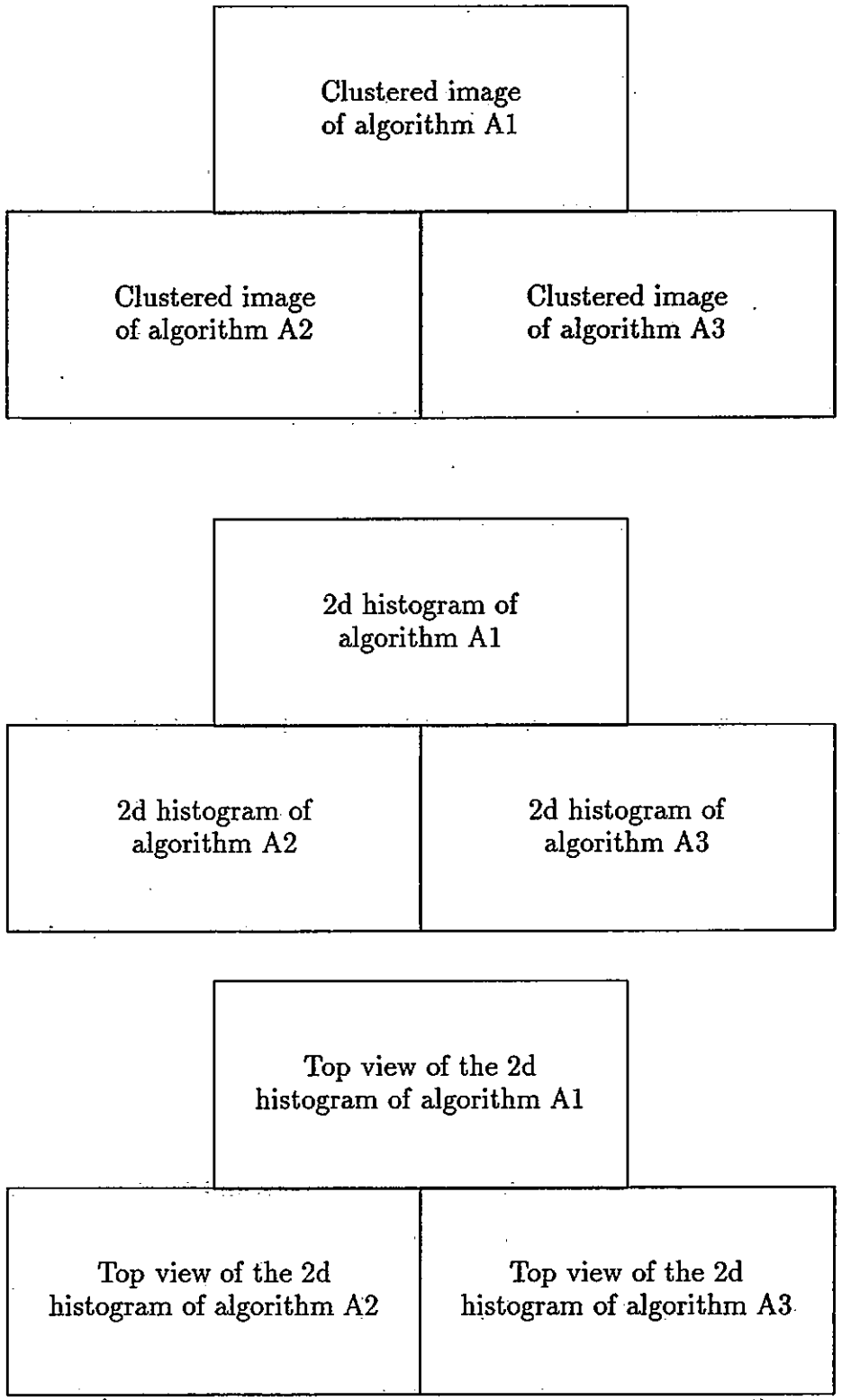


Figure 4.18: Clustering results of 11th March images

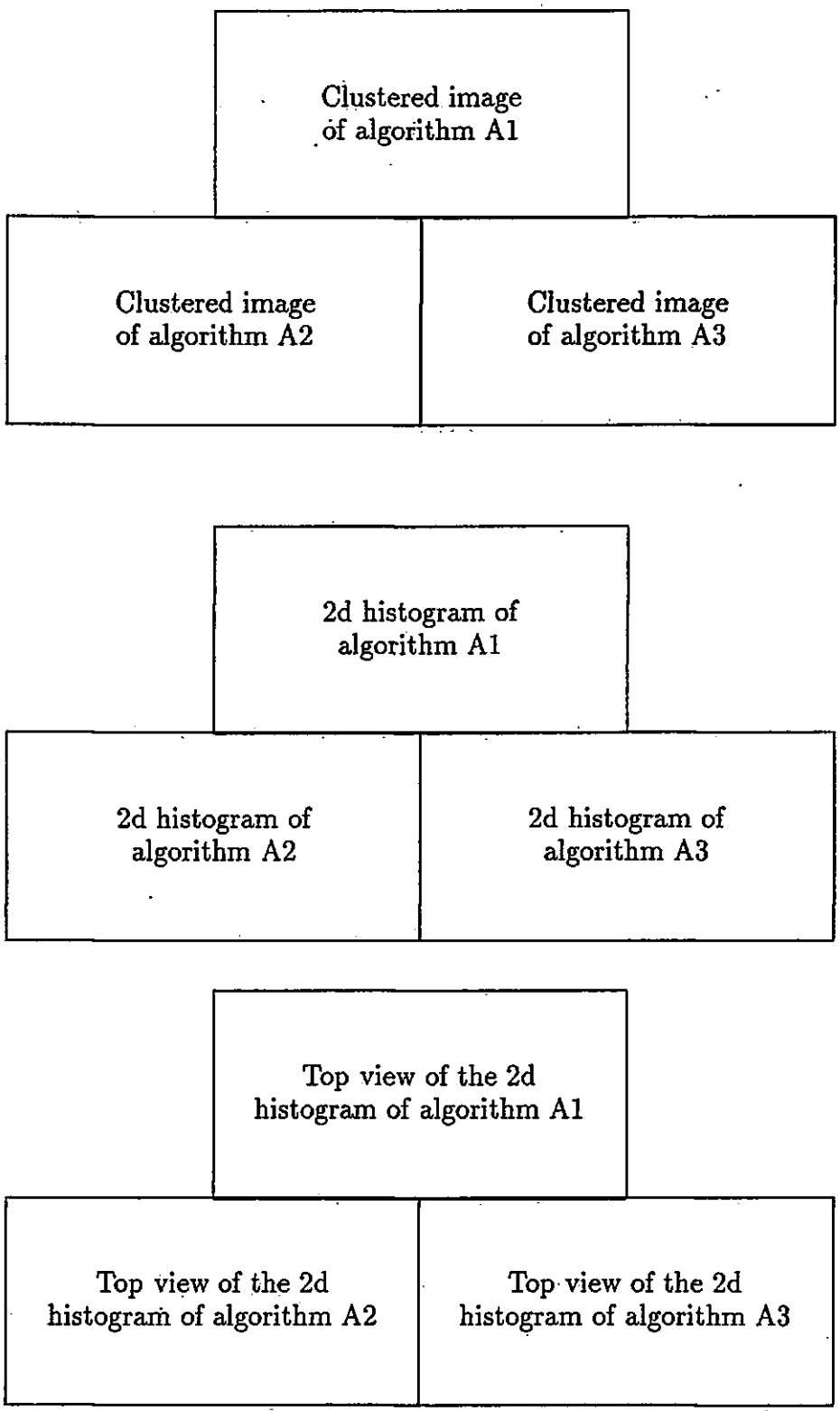


Figure 4.19: Clustering results of 15th March images

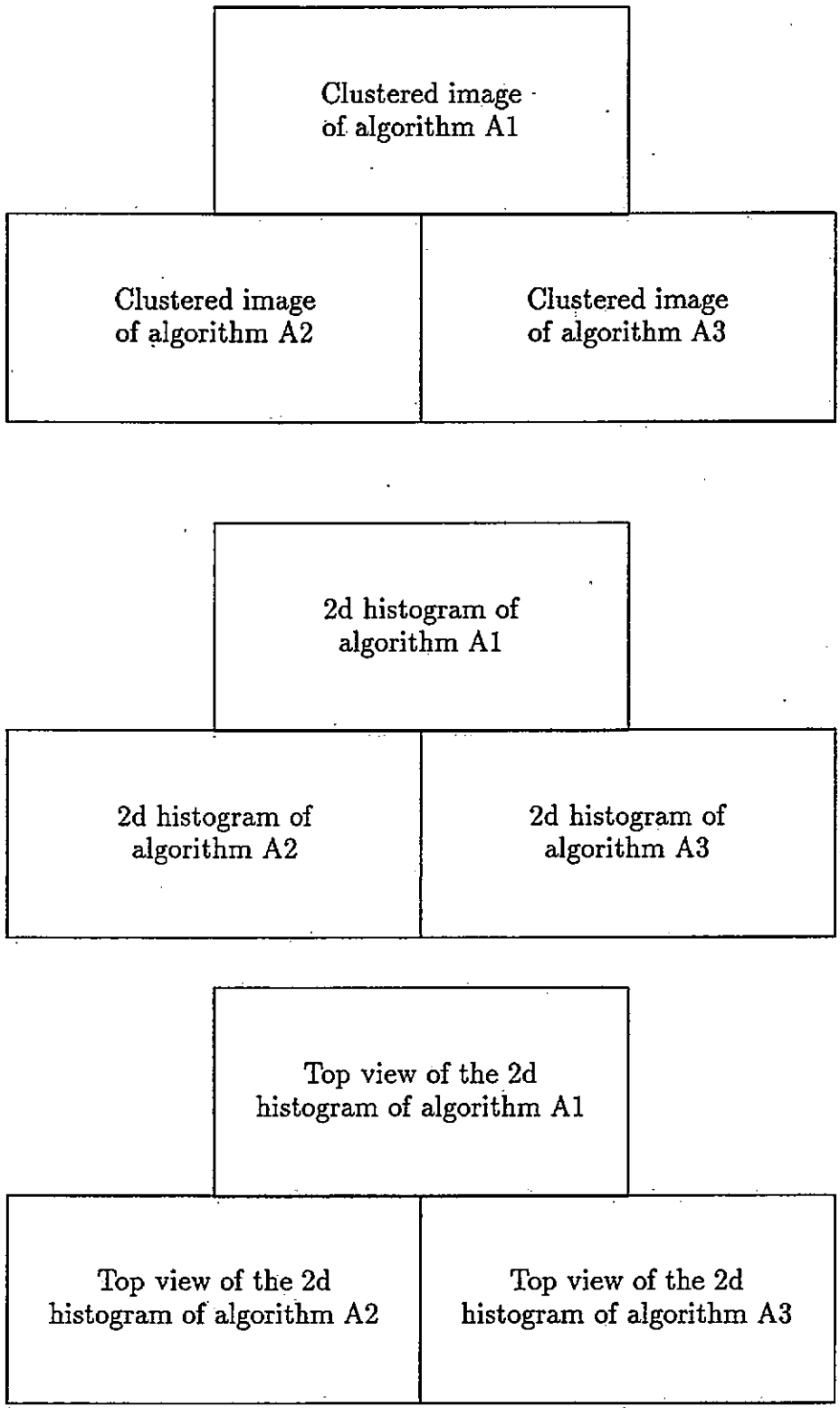


Figure 4.20: Clustering results of 18th March images

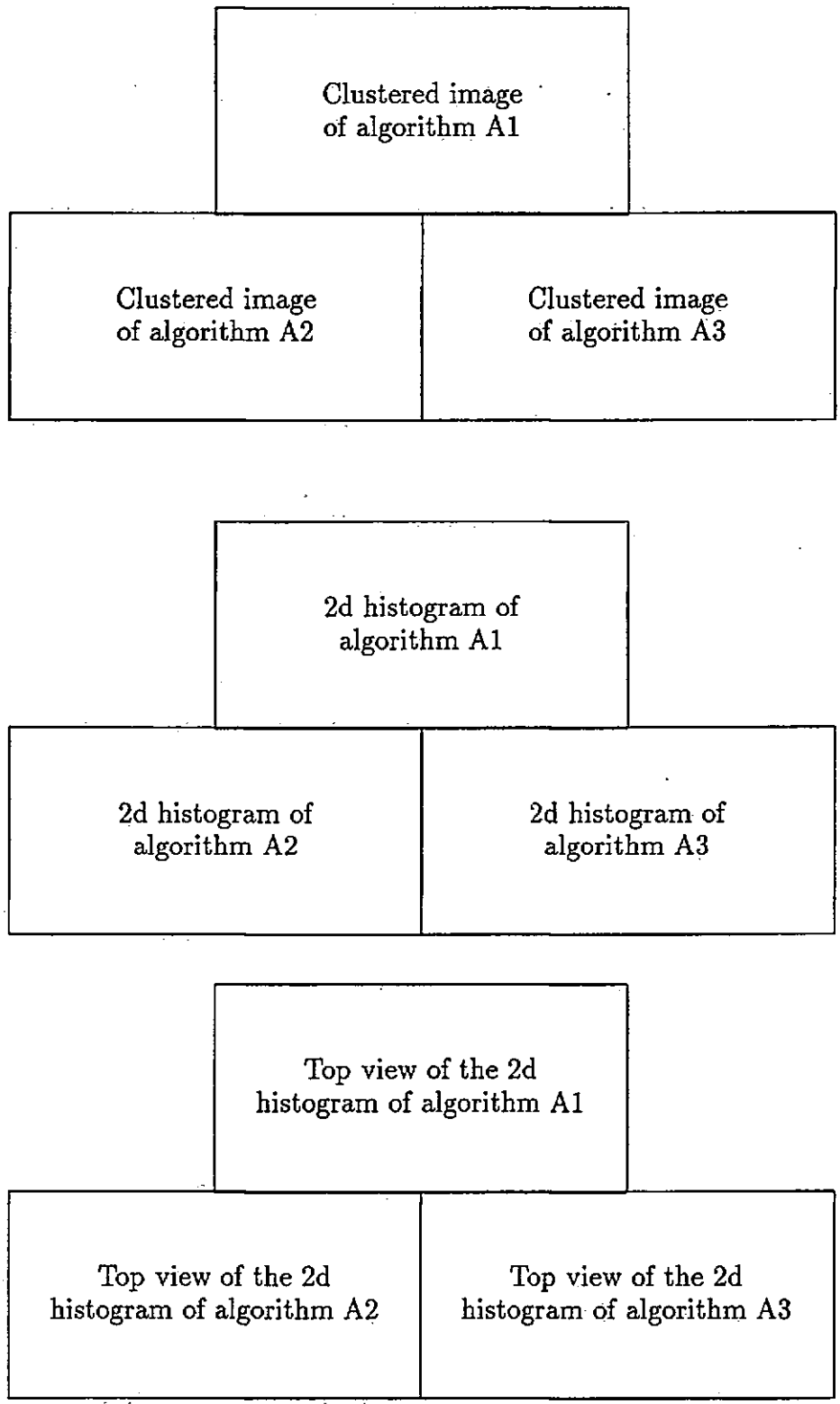


Figure 4.21: Clustering results of 20th March images

cluster	No. of pixels			cluster mean ($\frac{VIS}{IR}$)			variance ($\frac{VIS}{IR}$)		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
				85.7	85.2	77.2	9.2	6.9	4.8
0	6101	6340	1716	207.8	205.3	209.0	4.8	6.4	3.3
				119.1	116.1	81.4	12.7	13.8	12.3
1	13265	18632	11550	206.5	200.7	195.9	9.6	13.2	10.2
				77.7	57.5	120.8	14.2	9.7	13.7
2	9196	5029	19938	187.2	186.2	195.9	7.6	12.2	18.3
				125.2	88.3	71.2	13.9	10.1	18.0
3	6788	7016	21854	171.9	170.5	157.9	11.6	12.0	10.7
				93.6	130.4	29.8	7.5	14.0	6.1
4	4498	3825	10472	160.1	163.6	145.9	6.2	7.6	7.9
				59.1	62.0	—	13.4	13.5	—
5	17663	16289	—	155.3	153.0	—	10.1	8.4	—
				27.3	27.5	—	4.3	4.2	—
6	8025	8405	—	143.3	144.7	—	5.3	7.1	—

Table 4.1: Clustering statistics for 5th March 1991 images.

cluster	No. of pixels			cluster mean ($\frac{VIS}{IR}$)			variance ($\frac{VIS}{IR}$)		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
				96.5	89.9	96.1	16.9	13.6	18.9
0	28002	20581	29253	201.2	204.5	200.4	11.0	10.1	11.6
				143.9	117.5	124.2	10.7	10.2	15.4
1	2401	6053	3289	167.8	194.7	171.0	11.6	6.6	4.5
				100.7	144.1	98.9	13.7	10.5	18.0
2	14963	2403	14919	162.2	168.3	159.4	9.3	11.9	9.6
				56.7	100.2	46.4	12.0	13.8	13.4
3	8703	16528	9585	154.8	164.1	150.8	14.1	10.4	10.7
				24.4	57.7	22.3	5.6	11.9	4.7
4	7549	7868	5335	138.8	154.8	134.6	7.0	13.2	3.4
				46.2	32.6	48.6	5.0	12.1	5.2
5	3918	12103	3141	124.8	134.4	121.2	9.3	10.4	6.7

Table 4.2: Clustering statistics for 8th March 1991 images.

cluster	No. of pixels			cluster mean ($\frac{VIS}{IR}$)			variance ($\frac{VIS}{IR}$)		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
				82.8	102.8	100.6	7.5	16.9	15.5
0	5811	21139	32725	187.8	185.9	175.8	8.9	11.7	17.7
				112.3	60.4	123.4	12.5	7.5	8.6
1	14588	3238	5886	185.7	174.6	153.9	12.6	9.1	6.0
				66.7	116.6	72.3	8.9	9.3	12.2
2	9363	10792	13286	155.1	154.2	151.2	16.7	7.1	12.0
				98.5	93.2	28.6	9.7	6.7	4.5
3	18755	10881	2896	154.4	153.3	145.1	8.6	7.3	5.6
				125.0	65.9	110.3	7.8	12.1	4.1
4	4198	8976	1573	152.2	144.2	143.7	5.3	7.2	3.6
				26.3	27.5	48.3	6.4	7.3	5.7
5	3902	4356	8190	142.7	143.4	128.2	11.7	11.7	14.5
				48.8	49.1	18.1	5.9	5.8	1.3
6	8907	6154	973	128.6	121.2	127.3	13.1	7.6	1.7

Table 4.3: Clustering statistics for 11th March 1991 images.

cluster	No. of pixels			cluster mean ($\frac{VIS}{IR}$)			variance ($\frac{VIS}{IR}$)		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
				91.2	99.2	92.7	13.1	15.6	13.0
0	7096	18771	8803	208.7	196.0	206.8	4.5	12.0	5.8
				101.4	56.7	100.6	14.9	10.5	15.2
1	14217	2567	21922	185.8	180.9	175.2	9.9	8.0	13.4
				125.8	123.0	127.2	8.3	9.0	8.2
2	5814	7253	5004	156.2	157.0	153.6	7.5	7.4	6.0
				64.6	97.1	71.0	13.7	10.9	15.9
3	16010	17232	16265	151.9	155.6	150.2	17.3	11.4	12.9
				102.0	65.6	108.5	9.0	12.2	5.7
4	14434	11038	3993	151.9	144.4	144.8	8.6	11.1	4.3
				23.8	25.3	25.2	3.8	5.2	5.1
5	2820	3318	3288	134.7	136.8	136.0	9.1	10.8	9.9
				47.8	48.1	49.3	4.4	4.8	6.4
6	5145	5357	6245	112.3	112.5	114.2	4.2	4.4	6.1

Table 4.4: Clustering statistics for 15th March 1991 images.

cluster	No. of pixels			cluster mean ($\frac{VIS}{IR}$)			variance ($\frac{VIS}{IR}$)		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
				114.9	104.5	119.2	6.3	13.1	5.8
0	7222	26229	6863	215.8	205.7	214.3	2.9	8.7	5.4
				100.9	57.7	98.4	15.8	12.7	16.3
1	27441	2567	28934	196.6	194.1	196.8	11.5	11.5	12.1
				63.9	115.4	66.0	14.3	9.7	13.1
2	14086	5299	11819	155.2	181.8	153.0	16.7	8.0	14.1
				106.4	75.6	107.1	14.0	13.5	13.4
3	7647	5659	7623	151.4	172.9	152.0	6.2	5.1	7.0
				26.1	98.5	28.3	3.7	17.4	5.4
4	3777	10519	4395	143.6	149.5	145.6	5.9	6.6	7.3
				17.4	24.9	17.8	1.1	6.0	1.6
5	1391	5690	1584	128.3	141.7	128.7	2.6	10.7	2.9
				50.2	55.0	51.0	6.9	9.3	7.0
6	3972	9573	4316	115.5	131.7	116.3	6.0	16.3	6.5

Table 4.5: Clustering statistics for 18th March 1991 images.

cluster	No. of pixels			cluster mean ($\frac{VIS}{IR}$)			variance ($\frac{VIS}{IR}$)		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
				112.0	70.8	111.9	14.4	16.8	11.1
0	12835	2065	8757	195.2	198.4	198.3	8.1	9.0	6.1
				103.3	102.5	104.0	10.7	10.2	15.4
1	12330	10611	14324	176.7	195.3	177.0	7.6	7.2	13.6
				77.8	108.1	74.4	10.1	15.1	10.5
2	10640	17386	11034	167.8	174.8	173.3	16.7	10.4	14.4
				50.9	62.1	113.3	10.4	12.0	11.3
3	9803	12199	10163	150.2	106.5	152.5	15.4	15.2	13.0
				116.9	113.2	39.6	11.1	15.8	11.8
4	6883	7530	10596	147.2	144.3	151.3	9.0	9.4	10.2
				24.2	26.9	20.0	4.7	6.9	2.7
5	6314	8037	2509	140.0	143.6	129.0	9.7	12.0	4.5
				49.7	50.5	51.3	7.1	7.6	8.3
6	6731	7708	8147	105.9	108.2	109.3	7.4	9.3	10.1

Table 4.6: Clustering statistics for 20th March 1991 images.

Since multiple solution exists for iterative clustering, it is very difficult if not at all possible to compare the statistics of the results, therefore they are compared based on the actual classes found in the original images. The exact number of clusters are also difficult to determine, but it is set such that the feature space are well partitioned. Due to the relatively low resolution of METEOSAT at high latitude it is not able to distinguish all cloud types, especially in regions where multi-layers of cloud exist.

The cloud types found in these images are:

1. Land
2. Sea
3. Low cloud
4. Middle cloud (Altostratus, Stratocumulus etc.)
5. Cumulus (subpixel)
6. thick Cirrus
7. thin Cirrus
8. Deep convective cloud

The following paragraphs concentrate on the comparison of results obtained using different initial partition methods, i.e. the Global-Local algorithm (A1) and the dynamic algorithm using random initial partition (A2), since the results of the histogram clustering are less important.

On the 5th March, both algorithm separate the low clouds and land well, the main difference is the thick Cirrus (middle of the window), it has been over estimated by the A2 algorithm. Most of the Stratus with thin Cirrus above has been mixed with the thick Cirrus. The Global-Local algorithm separate the stratus cloud underneath the thick cirrus well, without overestimating the thick cirrus.

The main features of the spiral cloud on 8th March has been identified well by both algorithm. Areas containing cumulonimbus (bottom) and thick Cirrus

(bottom right) are the main difference. The A2 algorithm has split the cluster of thick cirrus into two clusters, in this case it is better to keep the thick cirrus in one cluster.

On the 11th March, low stratus cloud has been well represented by both algorithms. The main difference is on the top left where a group of thick cirrus has been identified by the Global-Local algorithm. The thick cirrus has been assigned to the stratus underneath the thick cirrus by the A2 algorithm.

On the 15th March, the anvils of located at the cold front has been separated well by the Global-Local algorithm but not by the random algorithm. The amount of thick cirrus (top left) has also been overestimated by the A2 algorithm.

On the 18th March, the random algorithm fails to separate the warm frontal cloud, this consists of a mixture of nimbostratus, altostratus and cirrus. The cirrostratus along the warm front has been separated particularly well by the Global-Local algorithm. The altostratus and stratocumulus behind the warm front and the warm frontal cloud has been wrongly assigned to a single cluster by the A2 algorithm.

On the 20th March, the frontal band of thick cirrus has been under estimated by the A2 algorithm as well as an area of cumulonimbus over southern England.

4.5 Discussion

In all cases, clustering using a random initial partition closely resembles the Global-Local clustering algorithm. However, in general the Global-Local clustering approach generates better clusters than using the random initial partition approach, as expected. In all six sets of images the thick cirrus and deep convective clouds in particular are better represented by the Global-Local clustering approach. Another disadvantage of the random initial partition approach is that the number of clusters is very difficult to control. Very often it generates kernels which does not attract enough points and are discarded, which in turn increases the variation of other kernels. Although a kernel will be split if its variation ex-

ceeds the limit, this usually results in unsatisfactory clusters or oscillation between split and merge.

The dynamic algorithms converge as expected and less than 3% of the pixels were transferred at the 11th iteration.

The histogram clustering (A3) is found to be able to provide very good initial partitions for the dynamic clustering algorithm (A1). The split and merge capabilities are also very useful for recovery from bad random initial partition.

In most cases the classes with very small variance and which are close in the feature space (e.g. different type of land or sea) are mixed together. This is not a serious problem for cloud classification, however it can be improved by only estimating kernels which exceed the split or merge parameters, because these clusters usually can be identified very well by the histogram clustering algorithm, and if their kernels are not disturbed they will not affected by split or merge of other clusters. For the Global-Local algorithm, if no split or merge has occurred the final partition is very similar to the initial partition generated by the histogram clustering. This implies that the dynamic clustering algorithm is very stable once it locks onto a cluster and will not move very much in the feature space.

In this chapter the histogram clustering has been shown to provide good initial partitions for classical iterative partitional clustering algorithm. The advantages of the Global-Local clustering algorithm are:

1. High efficiency (it exploits multiple occurrence of a pattern vector).

2. It obtains an initial partition objectively using a very efficient histogram clustering algorithm which requires few control parameters.

3. It uses an optimum Gaussian kernel as a cluster model, which has found to suit cloud imagery well.

Chapter 5

A Spatial-Spectral Clustering Algorithm

The Global-Local clustering scheme presented in Chapter 4 classifies each vector using only its spectral features. This approach is usually termed per-pixel clustering. Per-pixel clustering algorithms are popular in clustering of remotely sensed data, because they require little/no data preprocessing and are very efficient. However, they do not utilize the spatial information in an image. Spatial information is important for several reasons, 1) human interpretation of images relies heavily on the spatial as well as the spectral relationship of objects, 2) texture, spectral and contextual features can be found in most images, 3) the low resolution of most weather image is usually such that a pixel grey level representing a mixed measurements of more than one object.

It is well known that (Kettig and Landgrebe 1976, Kittler and Pairman 1985a) per-pixel classifications of remotely sensed data are "noisy". This result is due to uncertainty of boundary pixels being classified without considering the categories of its neighbourhoods. The "classification noise" is particularly undesirable for subsequent shape analysis of the object shapes.

Spatial information can be characterised by textural features or extraction of homogeneous regions in the image. Texture refers to a description of the spatial variability of tones found within part of a scene. Various measures of texture have

been successfully used to improve classification of remotely sensed data (Haralick and Shanmgau 1973). One disadvantage of textural measures is that there is an effective reduction in spatial resolution of the final classified image because for the measure to be effective the whole image is divided into small regions and these regions are classified using their textural measures. For example Haralick and Shanmgau (1973) use 64 x 64 pixels sub-regions and 44 textural features to classify LANDSAT data, they claimed a 10% improved accuracy over per-pixel classification. Similarly approaches for cloud images was reviewed in Chapter 2.

Whereas textural features contain information about the spatial distribution of tonal variations within a band. Contextual features contain information derived from blocks of pictorial data surrounding the area being analysed. Since the tracking of mesoscale cloud motion require a target window as small as 4 x 4 pixels, the textural approach is not suitable and we shall concentrate on contextual approaches.

In this chapter a new Spatial-Spectral clustering algorithm is presented. This comprises of two stages: firstly the image is segmented into spatially connected homogeneous regions corresponding to objects in the original images using a method called Graph Theoretic Hierarchical Segmentation (GTHS). In the second stage, objects belonging to the same category are grouped using an approach similar to GTHS, but they are grouped based on their spectral similarity only.

5.0.1 Review of Contextual Classifiers

The use of contextual information is based on the assumption that (Haralick and Kelly 1969):

1. Objects which are very close together are probably the same or similar type of object.
2. A sensor which is sensing the same or similar types of object will record the same or similar numerical measurements.

The above model leads immediately to the idea of finding homogeneous subregions and then classifying each region as an object. This idea of image partitioning is called image segmentation. Segmentation is defined as the partition of an image I into m spatial connected subregions $P = \{R_1, \ldots, R_m\}$ such that $\bigcup_{i=1}^{m} R_i = I$. Each region is homogeneous such that the variation within each region is less than a threshold $\sigma^2 \leq \theta_t$.

The simplest method is to divide the image into successively smaller rectangles and produce a partition that tends to minimize a criterion function. Robertson (1973) defined a subregion K as homogeneous if the mean $M(K) = M(J)$ for some subregion K within image J. The whole image I is partitioned recursively until all subregions are homogeneous or equal to the minimum block size allowed. The mean test of partition J into J_1 and J_2 is approximated by a multi-variate T^2 statistical hypothesis test that assumes the grey-levels in J_1 and J_2 are normally distributed, and tests the hypothesis that $M(J_1) = M(J_2)$. The subimages are then classified using either supervised or unsupervised methods. This partitioning of the whole image recursively into smaller regions are referred as disjunctive approach or top-down approach.

Farag (1978) presented a top-down procedure based on an information theoretic approach. The algorithm started with the whole image and split recursively into subregions such that the total information is minimised. If all regions are homogeneous the mutual information conveyed should be minimum.

Kettig and Landgrebe (1976) suggested an contextual algorithm for multispectral image classification. The algorithm is to merge 2 x 2 pixel cells (or larger) until regions meet at their boundaries. The merging of cells are based on a multi-variate composite hypothesis test, i.e. if the test is positive two regions are merged, and cells which do not pass the homogeneity test are assumed to lie on boundaries, these pixels are classified using per-pixel methods. The homogeneous regions are called fields and can be classified using supervised or unsupervised methods. Alternatively we can cluster pixels into regions and grow regions until their variance exceeds a predefined threshold. This approach is referred as

conjunctive or bottom-up approach.

Other approaches try to identify homogeneous objects by scanning the image sequentially in a line by line sequence. Nagy and Tolaba (1972) proposed a spatial clustering method based on the extraction of homogeneous regions by strip formation. A strip is a segment of a scan line. The strips are allowed to grow until the addition point raises the internal scatter of the strip above a designated strip threshold. At that point, the formed strip was assigned to a cluster (or designated to start a new cluster), and the formation of a new strip begins. The assignment of a strip to a cluster was done by comparing the strip to the cluster centres. The search for a cluster was done in a decreasing order of cluster population, to eliminate small groups of abnormal components.

Jayroe (1973) introduced a three stage spatial clustering procedure for multi-images. In the first stage, a boundary map is prepared by the thresholding of gradient images. The two gradient images used are obtained by computing the Euclidean distance between nearest neighbours in the horizontal and vertical directions. In the second step clusters are formed by scanning the boundary map with a fixed size window. When the window hits a region in which there are no boundary cells, that region is assigned to cluster 1. The window is then moved further, and if no boundary cells are encountered, the area within the window is assigned to cluster 1. The scanning continues until all possible cells are assigned to that cluster. Next, the window is moved until it hits a new region with no boundary cells, and the process is repeated. Finally, clusters are merged according to their spectral features.

Haralick and Dinstein (1975) proposed a spatial clustering procedure based on gradient images. The procedure starts with computation of a gradient image using Robert's gradient. The gradient image is then thresholded to generate homogeneous regions. A clearing procedure is applied to the threshold image to eliminate fuzzy boundaries. The image is then scanned line by line to identify connected strips, these strips are then merged into spatially connected regions. Finally these regions are clustered.

Kauth et al. (1977) introduced spatial coordinates of each pixel into the vector description of the pixel and to use this information along with the spectral features in a conventional ISODATA clustering algorithm.

Bryant (1979) proposed a spatial clustering procedure based heavily on heuristics. Spatial information is incorporated by identification of pure pixel (fields) and a pixel's label depends on the values of its 4 nearest neighbours. The fields are merged spatially and 5 test pixels are selected from each field. Cluster centres are then generated using the mean vector from each field, and the test pixels are classified with these mean vectors. Finally the means that do not adequately attract test pixels are eliminated. Finally the field mean vectors and pixels are classified using per-pixel nearest neighbour classifier.

Supervised methods using contextual information have been studied by Kittler and Pairman (1985a), and by Swain et al. (1981). They proposed contextual classifiers based on the maximum likelihood classifier. These algorithms are to minimise a loss function which take into account the dependence of a vector and its neighbourhood. The initial label is obtained by conventional per-pixel classification and the label are iteratively change until all labels are stable.

For most contextual classification methods only marginal improvement in classification accuracy is reported, but the change in object shape is obvious and it can affect subsequent machine analysis of objects.

5.0.2 Summary

All spatial-spectral clustering algorithms start with the identification of homogeneous regions or objects(segmentation) and these objects are grouped. a later stage using a clustering algorithm. Most algorithms require some threshold values for testing of a region homogeneity, and spatial information is not fully utilized, i.e. the region are constructed using strip growing or merging of small blocks. The consequence of not being able to make full use of spatial information creates regions with jagged edges and blockiness (Morris et al. 1988).In this chapter an unsupervised approach is used such that no threshold values are required. The

new Spatial-Spectral clustering algorithm is able to produce clusters with accurate boundaries and clusters are less "noisy" compared with per-pixel clustering approaches. These properties are important to machine analysis of the cluster shape.

5.1 Graph Theoretic Hierarchical Segmentation (GTHS)

The first stage of the Spatial-Spectral clustering concerns the generation of spatially connected homogeneous regions. This is achieved by a Graph Theoretic Hierarchical Segmentation (GTHS) technique which clusters the spatial space using a Graph Theoretic method.

Image segmentation algorithms based on graph theoretic clustering are developed in this section. The GTHS algorithms try to exploit the spatial information which is often ignored in many image partition algorithms. Segmentation is a fundamental problem in image analysis and understanding. Clustering algorithms have been shown to be an effective approach for image segmentation (Fu and Mui 1981).

5.1.1 Definition of Spatial Space and Feature Space

Segmentation can be either a partitioning of feature space or the spatial space. For classification purposes, as shown later, a partition of the spatial space requires further grouping of similar segments in the feature space to produce unlabelled classes.

We consider the feature (measurement) space first. A d-dimensional image is a two dimensional sequence of d-tuple vectors and the elements in each vector correspond to the grey-levels L of the sensed image. The feature space is defined as the Cartesian product of $G = L_1 \times L_2 \ldots \times L_d$. If the size of the image is $N \times M$ pixel, then the image I can be regarded as a sequence $I = \{g_{ij} | i \in Z_x, j \in Z_y\}$ where $Z_x = \{1, 2, \ldots, N\}$, $Z_y = \{1, 2, \ldots, M\}$ and $g_{ij} \in G$, $\forall i, j$. Therefore the

spatial space is the Cartesian product of $S = Z_x \times Z_y$. An example is given in Figure 5.1.

1	2	1	6
1	5	6	6
1	10	10	5
9	6	10	9

(a) 4 x 4 image

A	A	A	B
A	B	B	B
A	C	C	B
C	B	C	C

(b) Best partition of feature space

A	A	A	B
A	B	B	B
A	C	C	B
C	C	C	C

(c) Best partition of spatial space

Figure 5.1: Partitions of a 4 x 4 image.

5.1.2 Spatial Clustering

It was shown in section 3.5.4 that graph theory plays an important role in clustering. One of the advantages of graph theoretic clustering is the ability to describe the clusters in a hierarchical order. Owing to different level of interest a data set can be partitioned into many possible combinations(Fig. 3.7). This is particularly true for image segmentation, where an image is segmented into regions that roughly correspond to objects, surfaces or parts of objects of the scene.

Graph theoretic and hierarchical clustering have been used extensively in the partitioning of feature space. In this Chapter it is shown that clustering of the image space (spatial space) is equivalent to partition of a image into spatially connected segments. The basic concept is to cluster the image pixel with the constraint that they are spatially connected. At this point it should be stressed that graph theoretic and hierarchical clustering have one major difference; it is the generation of clusters. In graph theoretic clustering clusters are formed by removing of inconsistent links, while in hierarchical clustering the dendrogram is cut at some level to generate clusters (Fig. 3.12). However, both methods generate clusters in a hierarchical fashion.

A hierarchical structure of a picture is common in low level image analysis (Tanimoto 1978, Tanimoto and Klinger 1980). A hierarchical structure means that the image can be divided into components, corresponding to scene objects, which can then be divided into subcomponents corresponding to objects. The hierarchical level is particularly useful to represent different resolution levels: a region, which is higher in the hierarchy than its subparts, is also larger than its subparts. Hence, higher level regions could be discovered at a coarser resolution than their subparts. Spatial clustering using graph theoretic methods automatically produce the hierarchical result as described above.

In order to obtain segments with accurate boundaries, the process must be able to take into account of both local and global spatial information. Graph theory has been shown to be an effective approach for detecting gestalt clusters (Zahn 1971). The application of graph theory to image segmentation can be found in

Morris et al. (1986) and Daskalakis et al. (1988a). The methods described in this Chapter are similar to the image graph presented by Morris et al. (1986), although the fundamental concept of our segmentation is based on clustering of the image coordinate space and is applicable to multi-spectral data.

The complexity of graph theoretic clustering of the feature space of an image with n pixels requires $n(n-1)/2$ similarity measures, which is prohibitively large even for a 256 x 256 pixel image. In image segmentation, however, only adjacent pixels need to be considered, thus reducing the number of similarity measures to m x n ($m = 4$ or 8). The neighbourhood of a pixel x is shown in Fig. 5.2

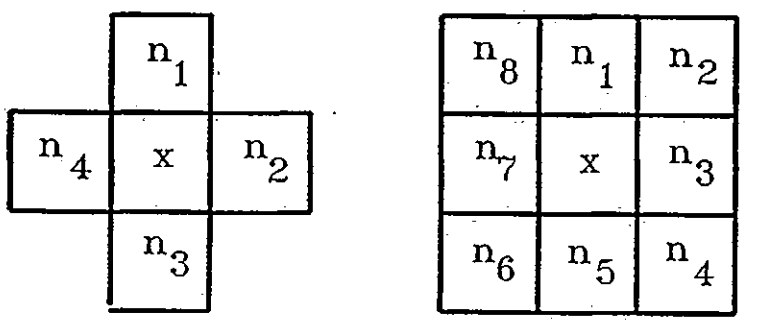


Figure 5.2: The neighbourhood of a pixel x

5.1.3 Basic Graph Theory

In order to present the GTHS segmentation method it is necessary to introduce some graph theory terminology.

1. A graph $G = (V, E)$ consists of a set of nodes or vertices joined by link E. A link can be directed or non-directed: directed link is called an arc, and a non-directed link is called edge. The end node of a directed link is the parent of the start node.

2. A completed graph has every node linked to every other node.

3. A graph is connected if there is a path from any node to any other node.

4. A graph is a directed graph if the links have direction.

5. A graph is non-directed graph if the links have no direction. A weighted graph has a value or weight e_{ij}, associated with the edge linking nodes i and j. The nodes may also be assigned weight v_i, v_j.

6. A path is a sequence of directed links, a chain is a sequence of non-directed links.

7. A cycle in a graph is a path or chain from some node i back to itself.

8. A tree is a connected set of paths or chains such that there are no cycles.

9. A forest is a graph which is not necessarily connected and in which there are no cycles.

10. A directed tree is a directed graph with a specific node called the root, the root has no parent.

11. A spanning tree is a tree that contains every node in G. The weight of a tree is the sum of link weights. The minimal spanning tree (MST) of a weighted graph G is the spanning tree of G which has minimal weight.

5.1.4 Spatial Clustering: A Stepwise Optimal Approach

All agglomerative hierarchical and graph theoretic clustering methods can be regarded as a procedure to minimise a global cost function step by step. In the case of clustering the cost function to be minimized at each step is usually the distance between two groups. Assume the cost of merging two regions R_i and R_j is $C(R_i, R_j)$, and that it is required to minimize the total cost of merging m segments. The criterion function is

$$J(P_m) = \sum_{i,j} C(R_i, R_j) \;\; \forall i, j \in L \tag{5.1}$$

where $L = \{e_{kl} | k, l \in Z_x \text{ x } Z_y\}$ is the set of links connecting the regions.

The identification of the minimum of $J(P_m)$ requires the search of all possible partitions, and a practical solution is found in clustering techniques. Graph theoretic techniques have been used because they can exploit spatial information of an image.

An initial partition of an image with n segments each containing exactly one pixel is $P^0 = \{R_1^0, R_2^0, \ldots, R_n^0\}$. At the kth iteration, the two most similar regions are merged from the P^{k-1} partition to produce a new partition $P^k = \{R_1^k, R_2^k, \ldots, R_{n-k}^k\}$. Therefore the number of regions decreases by one at each iteration, P^k must contains $n - k$ regions. $J(P^k)$ tends to increase after each merging, and can be written as:

$$J(P^k) = J(P^0) + \sum_{\alpha=1}^{k} \left[J(P^\alpha) - J(P^{\alpha-1}) \right] \tag{5.2}$$

It is easy to see that if the increase in $J(P)$ is minimized at each iteration then so is $J(P^k)$. Thus, a suboptimal solution of the global optimization problem is obtained using this stepwise approach. In general the criterion function $J(P)$ increases step by step and the stepwise approach produces a sequence of $J(P^\alpha)$ such that

$$J(P^0) \leq J(P^1) \leq \ldots \leq J(P^\alpha) \leq J(P^k) \tag{5.3}$$

5.1.5 General Form of the Spatial Clustering Algorithm

A graph theoretic segmentation based on the clustering concept presented in previous sections is introduced.

The general procedure for Graph Theoretic Hierarchical Segmentation (GTHS) is as follows:

Step 1. Begin with n segments each consisting of exactly one entity. Set $r = n$.

Step 2. Search for the most similar pair of segments (only spatially close neighbours are compared).

Step 3. Reduce the number of segments r by 1 through linking of segments R_i and R_j. Label the newly formed segment. Save the link connecting R_i and R_j. Update the similarities using the chosen distance measure.

Step 4. Perform Step 2 and 3 $n-1$ times (until all entities are in one segment).

Step 5. All entities are connected by a graph which links are those saved in Step 3.

Step 6. Form segments by removal of the most costly link saved in Step 3.

Step 2 is crucial to the result of the spatial clustering. The similarity measure of neighbouring segments determines how well the global and local spatial relationships are exploited. This is demonstrated using the distance functions in the single linkage and centroid methods.

The GTHS segmentation may consists of one or two stages: The first stage is the construction of a spanning tree, and the process terminates when the required number of segments is obtained. This is called the bottom-up approach. In the second approach the spanning tree is allowed to span the whole image and segments are generated using a second stage. The second stage is the identification of inconsistent edges (e.g. the $m-1$ most weighted edges are removal to obtain m segments, or the spanning tree is partitional such that a criterion function is

optimized). This is called the top-down approach.

The determination of the number of segments present in an image is as difficult as the determination of the number of clusters present in the feature space, and it will be helpful to have some measure of segmentation performance. A segmentation model based on mutual information is presented later. This monitors the segmentation process and is used to suggest suitable number of segments.

5.1.6 The Image Graph

An efficient algorithm is essential for practical use of the spatial clustering method. One way to implement the GTHS segmentation method is to compute all the possible pairwise similarities of a pixel and its neighbours. The neighbourhood of a pixel can be defined by a window of size p x p pixel. For computational reasons and simplicity only 4 or 8 neighbours are usually used, and in this study 8 connectedness are used to demonstrate the ability of spatial clustering (Fig. 5.2).

To implement GTHS segmentation, the image must be mapped onto a graph and the simplest mapping is to consider every pixel as a node in the graph (see Fig. 5.3).

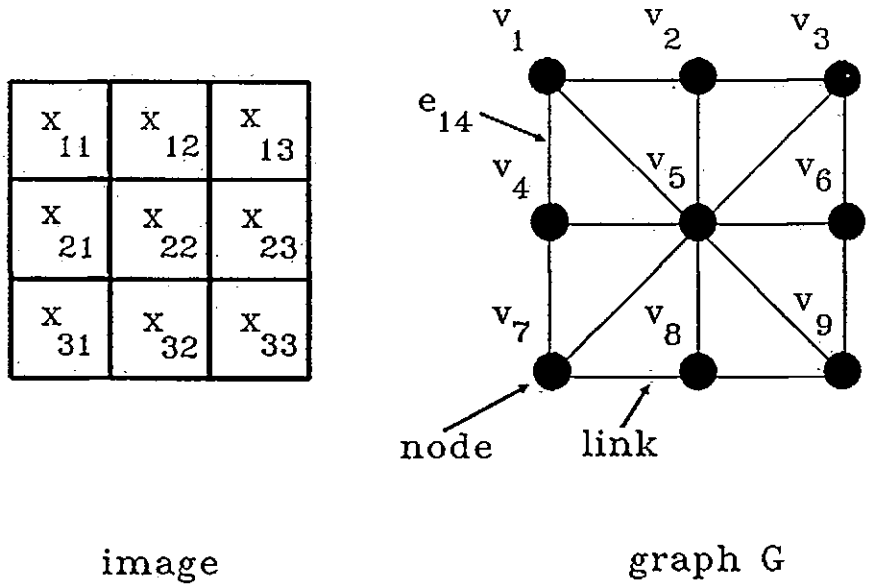


Figure 5.3: Mapping of a 3 x 3 image onto a graph with 8 connectedness.

Every node represents a vector in the image. The vector is simply the grey-level of the pixel. Other features such as texture or grey-level gradient can also be used. Therefore every node carries a weight $v_k = g_{ijk}$, and every link connecting node i and j has weight $e_{ij} = ||v_i - v_j||^2$ which is the squared Euclidean distance between the two vectors mapped onto v_i and v_j. Every node can be connected to more neighbours but considering the complexity only 8 neighbours of a pixel are used. It must be noted that the graph generated by the merging of spatially connected segments is always a spanning tree of the image graph G.

It is noted that the merging stage (Step 1 — Step 5) in the GTHS algorithm is a variation on constructing a Minimal Spanning Tree (MST). There are two well known algorithms to construct a MST (Prim 1957, Kruskal 1956). The importance of choosing a distance function has been stressed previously, and several distance functions are chosen to illustrate the principle of spatial clustering. These distance functions were presented in section 3.5.4. Traditionally a name is associated with each hierarchical clustering method using a particular distance function, the same

name will be used here for convenience, but the algorithms presented here should not be confused with hierarchical clustering (section 3.5.4).

5.1.7 Single Linkage Spatial Clustering

A clustering obtained by single linkage clustering is equivalent to the construction of a minimal spanning tree (MST) on the data set. Efficient algorithms exist to construct the MST and usually either Prim's or Kruskal's MST algorithm is used. It should be noted that Prim's algorithm is more efficient than Kruskal's algorithm, but as will be shown Kruskal's algorithm is more suitable for spatial clustering because it allows both top-down and bottom-up approaches.
Prim's MST algorithm is :

> Begin with an arbitrary node of G and add the edge with smallest weight connected to this node. This link with its two end nodes constitutes fragmented tree T_1. The kth fragmented tree grows by adding the shortest link from T_{k-1} to the nodes of G not in T_{k-1}. This continues until T_{n-1} is the desired MST.

Therefore, in Prim's algorithm the MST grows from a single node by adding the closest node to the current tree at each stage along with the link corresponding to that closest distance. A MST of a 4 x 4 image is shown in Fig. 5.4. Removal of edges e_a and e_b will partition the image into 3 homogeneous regions as required.

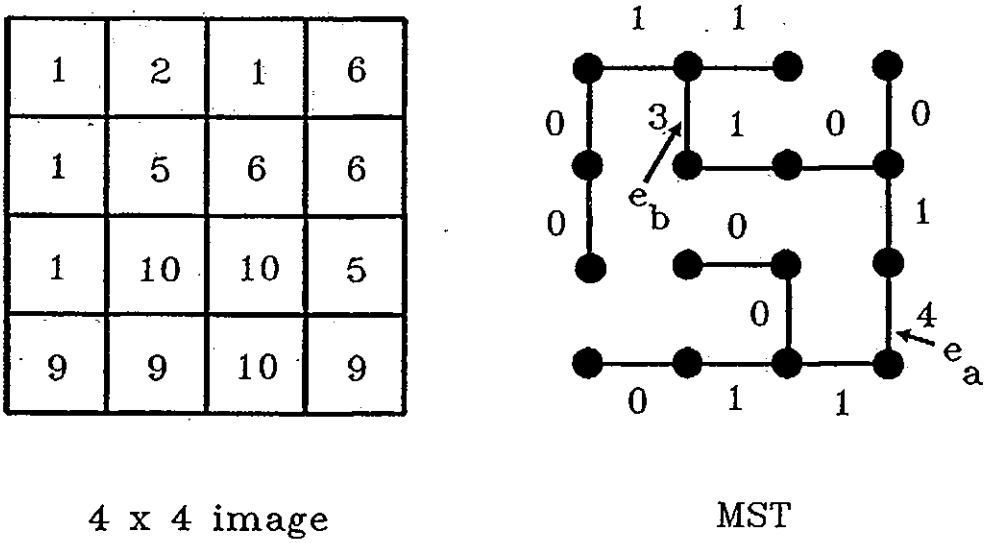


Figure 5.4: MST of a 4 x 4 image (4 connectedness), removal of edge e_a and e_b generate three homogeneous regions.

Since the Prim's MST algorithm only allows the growth of one tree at all times, it is not suitable for the bottom up segmentation approach. This study uses the Kruskal's algorithm which, in contrast to the Prim's algorithm, trees grow simultaneously, starting from the most homogeneous regions.

The segments similarity measure for single linkage spatial clustering is

$$\delta_{min}(T_i, T_j) = \min_{v \in T_i, v' \in T_j} \delta(v, v') \tag{5.4}$$

5.1.8 Complete Linkage Spatial Clustering

The concept of complete linkage is no more complicated than the single linkage except the distance between segments becomes

$$\delta_{max}(T_i, T_j) = \max_{v \in T_i, v' \in T_j} (v, v') \tag{5.5}$$

Figure 5.5 is an example of a spanning tree generated by the complete linkage method (CST).

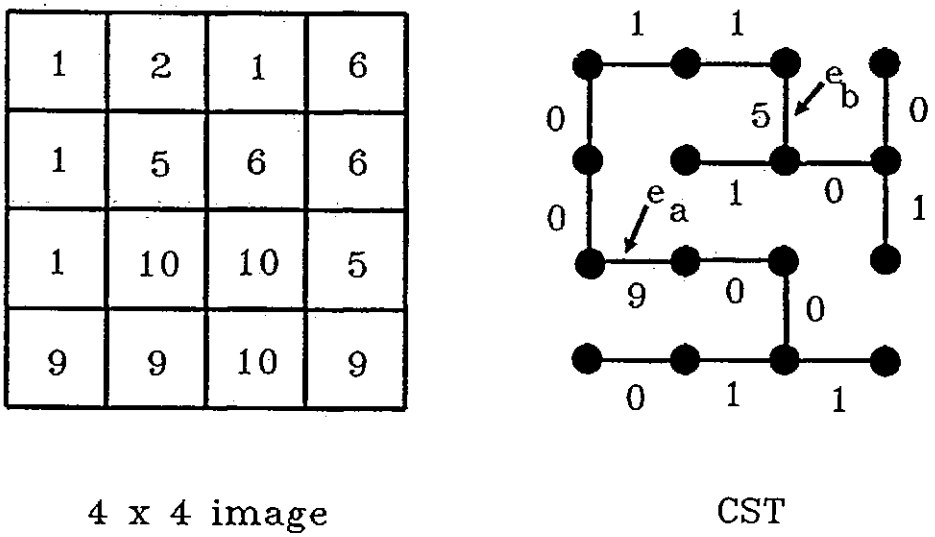


Figure 5.5: CST of a 4 x 4 image (4 connectedness), removal of edge e_a and e_b generate three homogeneous regions.

The algorithm to construct the CST is similar to MST and the weight of the edges between segments is determined using eqn. 5.5.
The algorithm for CST is:

Step 1. Map the image onto a weighted graph G. Set $r = n - 1$.

Step 2. Find the least weighted link.

Step 3. Save the least weighted link.

Step 4. Keep links $e_{k(ij)}$ which satisfy $\delta_{max}(T_k, T_i \cup T_j)$

Step 5. Remove duplicated links.

Step 6. $r = r - 1$ goto Step 2 if $r > 0$.

Step 7. Form a spanning tree with the saved links.

5.1.9 The Centroid Method for Spatial Clustering

The distance measures used in the MST and CST will be shown to belong to local type distances. These distances do not incorporate the spatial information beyond a pixel's nearest neighbours and so, as one could expect, they are sensitive to local variations such as noise. A possible way of incorporating spatial information beyond a pixel's nearest neighbours is to use the average distance (global type) between two spatially connected regions. This section presents methods of constructing a spanning tree based on global distances.

The distance measure used to construct the spanning tree by the centroid method (CEST) is :

$$\delta_{mean}(T_i, T_j) = \delta(m(T_i), m(T_j)) \tag{5.6}$$

where $m(T_i) = \frac{1}{n_i} \sum_{v_j \in T_i} v_j$ is the mean or centroid of region R_i defined by T_i, $n_i = \#\{v_j | v_j \in T_i\}$. It is noted that the Recursive Spanning Tree (RST) which is equivalent to CEST has been used by Morris et.al (1986) and by Lau and Wade (1991) for image segmentation.

The algorithm to generate CEST is as follows:

Step 1. Map the image onto a weighted graph.

Step 2. $r = n - 1$.

Step 3. Save the next least weighted edge, say e_{ij} between nodes i and j (in general node i will represent n_i original node weights and node q will represent n_q original node weights. The weight of n_i, n_j will be the mean of the weights of all the nodes it represents).

Step 4. Merge the two nodes i and j to make a new node k with weight equal to the mean of all the node weights in nodes i and j, i.e.
$v_k = \frac{1}{n_i+n_j} \sum_{v \in T_i, v \in T_j} (v + v')$

Step 5. Find the new edge weights (using eqn. 5.6) for all edges which are now connected to node k (frequently this process leaves redundant edges, which are discarded).

Step 6. $r = r - 1$, goto Step 2 if $r > 0$

Step 7. generate a CEST using the saved links.

The interpretation of the CEST is made easy by a recursive form of generating the MST. At each stage of the iteration the total number of nodes is reduced by one through merger of two closest nodes i and j. The merged node will represents a number of nodes which have been merged together and the whole process of merging is repeated. The node weights of the new node is represented by the mean of the merged node, and the link weights connecting the new nodes are updated. As the nodes are merged, it is possible for one node weight to affect another which is not necessarily its nearest neighbour. Initially only local information is used since each region only consists of one pixel, but as the iteration progresses and the regions grow, more global information is used. An example of CEST is given in Fig. 5.6).

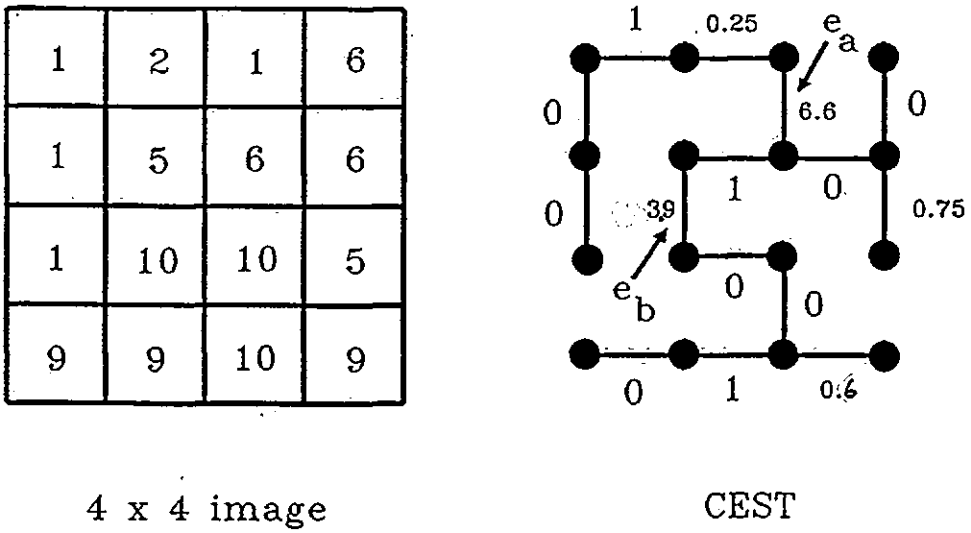


Figure 5.6: CEST of a 4 x 4 image (4 connectedness), removal of edge e_a and e_b generates 3 homogeneous regions.

5.1.10 The Variance Method for Spatial Clustering

So far all spatial clustering methods only involve minimisation of edge weights in each iteration. The stepwise optimal approach can be generalised by maximizing an objective function instead of a distance function and a popular choice for an objective function is to minimize the variance of a segment. This function is justified by the assumption that each region should be homogeneous in some scene. This method was proposed by Ward (1963) and he illustrated his method with an error sum of squares objective function.

The algorithm of the variance method for constructing a spanning tree (VST) is similar to other spatial clustering methods and is as follows:

Step 1. Map the image onto a weighted graph.

Step 2. $r = n - 1$

Step 3. Find the pair of distinct trees T_i and T_j whose merger would increase the criterion function as little as possible.

Step 4. Save the edge e_{ij}

Step 5. $r = r - 1$, if $r > 0$ goto Step 3.

Step 6. Generate spanning tree using the saved edges.

The criterion function over all partition is

$$J(P) = \sum_{\forall T_i} \sum_{v \in T_i} \sum_{j=1}^{d} (v_j - m_j(T_i))^2 \tag{5.7}$$

where $m(T_i)$ is the mean of tree T_i

Suppose segment R_i and R_j are chosen to be merged and the resulting segment is denoted as R_k. Then the increase in $J(P)$ is

$$\begin{aligned} \Delta J(P) &= \sum_{q \in T_i \cup T_j} \sum_{k=1}^{d} (v_{qk} - m_k(T_i \cup T_j))^2 \\ &\quad - \sum_{q \in T_i} \sum_{k=1}^{d} (v_{qk} - m_k(T_i))^2 \\ &\quad - \sum_{q \in T_j} \sum_{k=1}^{d} (v_{qk} - m_k(T_j))^2 \\ &= \frac{n_i n_j}{n_i + n_j} \sum_{k=1}^{d} (m_k(T_i) - m_k(T_j))^2 \end{aligned} \tag{5.8}$$

Note that eqn. 5.8 is the weighted squared distance of the two centroids, while the Centroid method the distance is unweighted. Therefore the variance method also uses global information. Figure 5.7 shows an example of VST.

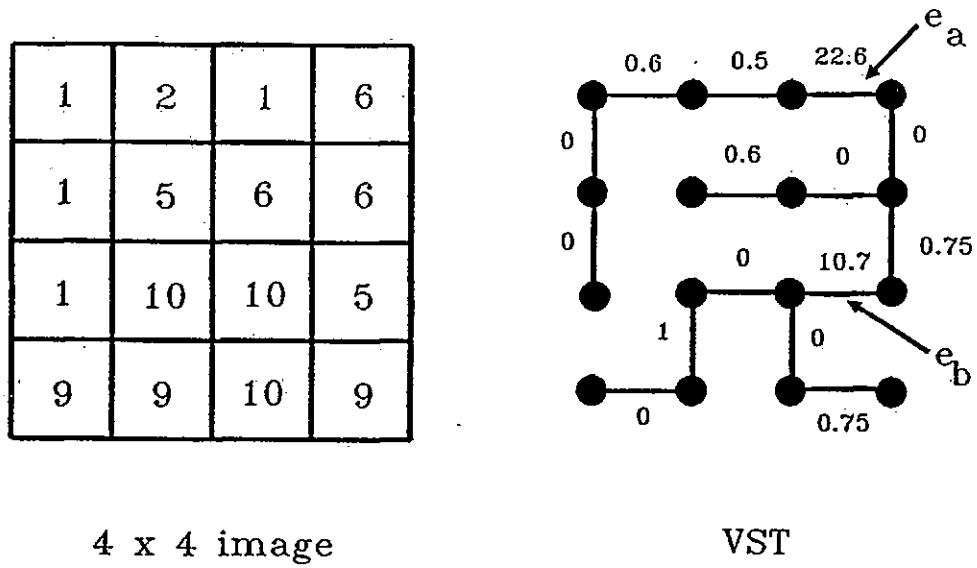


Figure 5.7: VST of a 4 x 4 image (4 connectedness), removal of edge e_a and e_b generates three homogeneous regions.

It is noted that an agglomerative approach similar to the variance method was used by Beaulieu and Goldberg (1989) to segment monochrome image. They have shown experimentally that the change in criterion $\Delta J(P)$ is in general increasing and can be used to guide the selection of the number of segments, but the mutual information approach used in this study produces a better indication.

5.1.11 Summary

The search for the partition P_{min} that minimizes a global criterion J requires a search over the entire space of all possible partitions $\{P\}$. The implementation of a exhaustive scheme is impractical, so a suboptimal solution such as stepwise optimal method has to be used.

In principle any distance used in a hierarchical clustering scheme e.g. average linkage can be used to construct the spanning tree. An important factor that limits the choice is the computational cost. For example, the MST is fast but it

has some undesirable effects, such as chaining. If a globular distance is used the computational cost increases, but with better results. There are many ways to construct a spanning tree, but only MST and CEST will be studied because they are representatives of methods using local and global information. In most cases the spanning tree is not unique because ties are resolved arbitrary.

Fig. 5.4—5.7, show that all spatial clustering methods produce the same partition by removal of the two most weighted edges of these spanning trees (they perform very differently when applied to large images). The most weighted link of the VST is much higher than the MST, CST and CEST, so VST is probably the best among these four trees for segmentation proposes. The properties of the spanning tree also depends heavily on the cost function chosen. There are two major type of cost functions: The first uses local distance such as the nearest distance, so only local information is used. The second type of cost function uses global information by taking account of other nodes in the same region. Global type cost functions are superior to local type for bottom up segmentation, while using the top down approach the performance are about the same as shown later. An efficient algorithm (Daskalaskis et al. 1988a) to implement a spanning tree can be found in Appendix C.

5.2 Bottom-Up Segmentation Approach

Two segmentation approaches can be used to generate segments. The first method is the bottom-up approach. Since the most similar regions are merged in each iteration, the construction of spanning tree can be stopped when the required number of regions is obtained. But the success of this method depends on the algorithm used to construct the spanning tree.

It is found that the Prim's algorithm does not allow segmentation using the bottom-up approach. Assuming a region with only one pixel is trivial, then the Prim's algorithm starts with one node and that node grows by merging one node at a time, so at every iteration there is only one tree in the forest representing a non-

trivial region. However if the Kruskal's algorithm is used (which is computational less efficient than Prim's algorithm) the number of trees in the forest representing non-trivial regions is greater than one after two iterations. This point is obvious from the Kruskal's algorithm:

> Arrange the edges of G in order from smallest to largest weight and then select edges in order making sure to select only edges which do not form a cycle with those already chosen. Stop when $n-1$ edges have been selected. The set of edges is then an MST for G.

If we represent each tree as a region in the image, it is clear that with Kruskal's algorithm regions are allowed to grow as the process continue. As a result GTHS segmentation using Kruskal's algorithm can generate segments without the need to complete the spanning tree.

The growth of a spanning tree generally starts with the most homogeneous regions and these regions act as "seeds". Regions grow around the seeds as the process continues. So spatial clustering belongs to the region growing methods for segmentation (Zucker 1976).

To display a segmented image the regions grey-levels are replaced by the mean of that region. Let the number of nodes in tree T_i be $\#\{v_j|v_j \in T_i\}$, the mean of region R_i is given by

$$m(R_i) = \frac{\sum_j v_j}{\#\{v_j|v_j \in T_i\}} \tag{5.9}$$

Although segmentation can be obtained with less computational cost by terminating the process when the required number of segments is obtained, results may depend very much on the spatial clustering method used. The bottom-up approach is used in this study and it is found that it only performs well with global type distances. A more reliable but expensive way is to complete the spanning tree and partition the spanning tree subject to a criterion function. A minimax method which is a suboptimal method will be presented for the top down approach.

5.3 Top-down Segmentation Approach

A spanning tree connects pixels according to their spatial and spectral relationship. This effectively limit the partition space by eliminating the solutions which clearly do not produce optimal partition. Cutting $m-1$ edges will produce m partitions of the image, generally cutting the most costly edges in a spanning tree produce the most prominent segments, but these segments are not necessary the best partition P_m. This is partly due to how much global information is used by a spatial clustering algorithm. Other methods may need detail inspection of the diameter histogram (Zahn 1971) to locate local minimum which correspond an edge connecting two distinct segments but with weight similar to its neighbouring edges. These methods is rather difficult to program and because of its ad hoc nature the result may be unpredictable.

A more systematic approach is to optimize a criterion function. Again a global optimization method requires search of the entire partition space $\{P\}$ which is computational prohibitive, therefore a suboptimal method has to be used. A spanning tree provides a practical approach for top down segmentation by limiting the search space to only $n-1$ searches. A method is to optimize a criterion function step by step to obtain a local optimum. Let $C(T_i)$ be the cost function of tree T_i, a criterion function

$$J(P) = \sum_{T_i \in P} C(T_i) \tag{5.10}$$

is minimized over all partitions. One possible cost function is

$$C(T_i) = \sum_{v_q \in T_i} \sum_{k=1}^{d} (v_{qk} - m_k(T_i))^2 \tag{5.11}$$

which represent the sum of squares error for segment R_i.

Consider the process using top-down spatial clustering approach, the initial partition consists of one region $P^0 = \{R_1^0\}$. At the kth iteration, the algorithm splits two regions from the P^{k-1} partition to produce a new partition $P^k = \{R_1^k, \ldots, R_{k+1}^k\}$. The criterion function $J(P)$ tends generally to decrease

step by step if we choose the cost function to evaluate the error sum of square or other variance measures of $\{R^k\}$ such that

$$J(P^k) = J(P^0) - \sum_{\alpha=1}^{k} [J(P^{\alpha-1}) - J(P^{\alpha})] \tag{5.12}$$

We want to minimize $J(P^k)$ such that the two most distinct regions are split. It is noted that the minimum value of $J(P)$ is equal to zero, which is the trivial segmentation of each segment contains one pixel. A step wise optimization is therefore to find a region whose split produces the largest decrease of $J(P^k)$. This decrease results from the splitting of a region R_t into R_i and R_j, so we want to minimize

$$\Delta J'(P) = C(T_i) + C(T_j) - C(T_i \cup T_j) \tag{5.13}$$

Since

$$J(P^0) \geq J(P^1) \ldots \geq J(P^{\alpha}) \ldots \geq J(P^n), \;\; \Delta J'(P^n) \leq 0,$$

Therefore we can maximize the negative of $\Delta J'(P)$, which is

$$\Delta J(P) = C(T_i \cup T_j) - C(T_i) - C(T_j) \tag{5.14}$$

In a bottom-up procedure $\Delta J(P)$ is being minimized, which implies that the two regions which increase the total cost by the least amount are being merged. Following the same argument the change in $\Delta J(P)$ of a top down procedure has to be maximized, such that every time two most distinct regions are generated. It must be noted that this stepwise approach only produce a local optimal solution. Nevertheless, as will be seen they generally produce good results.

Suppose we are at the kth partition $P^k = \{R_1^k, \ldots, R_{k+1}^k\}$, the optimization of eqn. 5.14 requiresthe search of $k+1$ trees, and let $\bar{n}$ be the average number of node in each tree, the search requires $O(\bar{n}(k+1))$ operations. One way to speed up the search is to search only the tree T_i with $C(T_i) = \max\{C(T_1), \ldots, C(T_{k+1})\}$, because T_i is potentially the one comprising more than one distinct region and splitting T_i is most likely to produce maximum change in the criterion function.

The algorithm for top-down GTHS segmentation is as follow:

Step 1. $r = m - 1, m$ is the required number of segments.

Step 2. Find the tree T_k which maximize

$$\Delta J(P) = C(T_i \cup T_j) - C(T_i) - C(T_j),\ T_k = T_i \cup T_j, T_k \in G$$

Step 3. Remove the link connecting T_i and T_j.

Step 4. $r = r - 1$, if $r > 0$ goto Step 2 else stop.

The top-down approach is used in this study, and the best result is obtained using top-down approach with a globular distance.

5.3.1 Minimax Segmentation

The minimax segmentation method is a simplified version of the stepwise optimal method just described. The minimax method minimizes the cost function $C(T_i) = \max\{C(T_1), \ldots, C(T_{k+1})\}$ at the kth partition P^k. This criterion based on the assumption that the segment with the largest variance potentially consists of more than one distinct regions. Once the tree $C(T_i)$ with highest cost is identified it is split into two trees $C(T_1)$ and $C(T_2)$ over all possible partition of $C(T_i)$, and selects the partition which minimizes

$$\Delta J'(P) = C(T_1) + C(T_2) \tag{5.15}$$

Since $C(T_i \cup T_j)$ in equation 5.14 is fixed therefore it is equivalent to minimisation of equation 5.15. However, it should be noted that minimax method is suboptimal to the stepwise optimal method because the optimization of $C_{max}(T_i)$ does not necessary produce the largest change in the criterion function. However it is computation efficient. An efficient algorithm (Daskalaskis et al. 1988b) to implement minimax on a spanning tree can be found in Appendix C.

The algorithm for top-down minimax GTHS segmentation is as follows:

Step 1. $r = m - 1, m$ is the required number of segments.

Step 2. Find the tree T_k with $C(T_k) = \max\{C(T_j)|T_j \in G\}$

Step 3. Cut T_k at edge e_{ij} connecting T_i and T_j

$\min[\max[C(T_i), C(T_j)]] \ \forall\, e_{ij} \in T_k,\ T_k = T_i \cup T_j$

Step 4. $r = r - 1$,if $r > 0$ goto Step 2 else stop.

The cost function $C(T)$ used in this study is the intraset distance (see Appendix D), which is essentially the average pairwise distance within a segment, and is

$$C(T_p) = \frac{1}{n_p(n_p - 1)} \sum_{i=1}^{n_p} \sum_{j=1}^{n_p} \sum_{k=1}^{d} (v_{ki} - v_{kj})^2 \quad v_{ki}, v_{kj} \in T_p \tag{5.16}$$

where n_p is the number of nodes in T_p.

It can be shown that $C(T_p) = 2\sum_{k=1}^{d} \sigma_{kp}^2$ where σ_{kp}^2 is the unbiased variance of the kth variance in the p segment.

$$\sigma_{kp}^2 = \frac{1}{n_p - 1} \sum_{i=1}^{n_p} (v_{ik} - m_k(T_p))^2 \tag{5.17}$$

5.4 Monitoring Segmentation

Relatively little work has been done on determining the number of segments in an image. This problem is similar to and as difficult as validating clustering studies. In this section a criterion function based on mutual information is presented. Due to the hierarchical structure of segments, this function only provides guidance for determination of number of segments.

Segmentation can be regarded as the approximation of an image. This is clearly the approach presented in previous sections, where a cost function of each segment is minimized. These cost functions are related to the variance of segments in general, and is most explicit in the centroid and variance methods. However, it is clear that the greater the number of segments, the better is the approximation.

Therefore the goal of segmentation is a compromise of local details and global features required by the application.

Ideally, the segmentation process should automatically terminate once near homogeneous regions have been found and one way of doing this is based upon entropy measure (Daskalakis et.al 1988b, Farag 1978). This is reasonable since the zero-order entropy of a near homogeneous segment, for example, tends to zero. The general information measure of a set of segments $P = \{R_1, \ldots, R_m\}$ can be defined as (Farag 1978)

$$
\begin{aligned}
I(P) &= I(R_1; R_2; \ldots; R_m) \\
&= \sum_{i}^{R_1}\sum_{j}^{R_2}\cdots\sum_{k}^{R_m} p(\mathbf{x}_{1i}, \mathbf{x}_{2j}, \ldots, \mathbf{x}_{mk}) \log \frac{p(\mathbf{x}_{1i}, \mathbf{x}_{2j}, \ldots, \mathbf{x}_{mk})}{p(\mathbf{x}_{1i})p(\mathbf{x}_{2j})\ldots p(\mathbf{x}_{mk})} \\
&= \sum_{i}^{R_1}\sum_{j}^{R_2}\cdots\sum_{k}^{R_m} \\
&\quad p(\mathbf{x}_{1i}, \mathbf{x}_{2j}, \ldots, \mathbf{x}_{mk}) \left[\log \frac{1}{p(\mathbf{x}_{1i})} + \log \frac{1}{p(\mathbf{x}_{2j})} \ldots + \log \frac{1}{p(\mathbf{x}_{mk})} - \log \frac{1}{p(\mathbf{x}_{1i}, \mathbf{x}_{2j}, \ldots, \mathbf{x}_{mk})}\right] \\
&= H(R_1) + H(R_2) \ldots + H(R_m) - H(R_1, R_2, \ldots, R_m) \qquad (5.18)
\end{aligned}
$$

where $\mathbf{x}_{mk}$ is the kth vector in segment m. Eqn. 5.18 can be interpreted as the total information conveyed by the segment P with properties

1. $I(P) \geq 0$,
2. $I(P) = 0$ if and only if the vectors are independent, $p(\mathbf{x}_{1i}, \mathbf{x}_{2j}, \ldots, \mathbf{x}_{mk}) = p(\mathbf{x}_{1i})p(\mathbf{x}_{2j}) \ldots, p(\mathbf{x}_{mk})$

For an ideal segmented image $I(P)=0$ since all distributions of $\mathbf{x}$ are independant, in practice we seek to minimise $I(P)$. It should be noted that $H(R_1, R_2, \ldots, R_m)$ is the joint entropy of the segments and is independent of the partition (constant), since it is a function of the probability distribution of the image itself. Therefore we could terminate segmentation when $H(P) = \sum_i^m H(R_i) < \theta_t$, where θ_t is a threshold and $H(P)$ is the segment entropy. In practice, due to problems of computing the higher order segment entropy, only the zero order approximation would be used.

This is reasonable, since, when the segmentation proceeds, interaction between segments will decrease and most higher order terms can be neglected (Ryan 1968).

Daskalakis et.al (1988b) used a similar criterion to monitor the segmentation. They assume the segment entropy $H(P)$ is composed of two components by

$$H(P) = H_s + \sum_j p(R_j)H(R_j) \tag{5.19}$$

where H_s is the entropy due to the existence of segments, $p(R_j)$ is the probability of occurrence of a particular segments R_j and $H(R_j)$ is the entropy of segment R_j. They assume the existence of segments obeys a Rayleigh pdf and pixels within a segment obey a normal pdf.

In this study, the concept of mutual information is used and the segmentation process is modelled as a noisy communication channel (see Fig. 5.8).

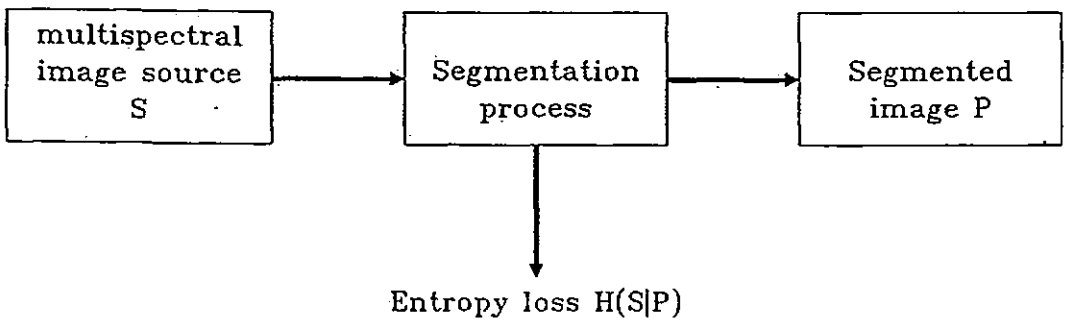


Figure 5.8: Segmentation modelled as an information flow process.

For such a channel, the mutual information common to both ends of the channel is given by the well known expression

$$I(S;P) = H(S) - H(S|P) \tag{5.20}$$

where $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is the input image, $P = \{R_1, R_2, \ldots, R_m\}$ is the segmented image.

$$\text{The source entropy is } H(S) = -\sum_i p(\mathbf{x}_i) \log p(\mathbf{x}_i) \tag{5.21}$$

The information given by a set of segments P is not equal to the sum of the information given by each segments taken separately, therefore they are not statistically independent. It is noted that the mutual information measure is symmetrical in its two arguments and may also be expressed as

$$I(S;P) = H(P) - H(P|S) \tag{5.22}$$

Eqn. 5.22 indicates that the set of segments P should be statistically independent to maximize $H(P)$, and at the same time to be highly statistically dependent on a given image to minimize $H(P|S)$.

Since $H(S)$ is constant for a given image, the goal is to minimize the quantity $H(S|P)$. If $H(S|P) = 0$, there is no ambiguity in the channel output, and the set of segments is the best representation of the input image. For the segmentation problem, $H(S|P)$ denotes an uncertainty in the segmentation or a "segmentation loss". It follows that if $H(S|P) = 0$, we could consider the segmentation process complete in the sense that the source image has been segmented into homogeneous regions.

The conditional entropy $H(S|P)$ can be expressed as

$$H(S|P) = H(S,P) - H(P) \tag{5.23}$$

For simplicity we might assume that the m segments of the processed image are approximately statistically independent and write $H(P)$ as the zero-order entropy

$$H(P) \;=\; -\sum_{j=1}^{m} p(R_j) \log p(R_j)$$

$$= -\sum_j \sum_{\mathbf{x}_i \in R_j} p(\mathbf{x}_i, R_j) \log p(R_j) \tag{5.24}$$

Here $p(\mathbf{x}_i, R_j)$ is the joint probability of pattern $\mathbf{x}_i$ and segment R_j. The zero-order entropy assumption becomes more realistic as segmentation proceeds and individual segments become more homogeneous and statistically independent. The joint entropy in eqn. 5.23 can be written as

$$H(S,P) = -\sum_j \sum_{\mathbf{x}_i \in R_j} p(\mathbf{x}_i, R_j) \log p(\mathbf{x}_i, R_j) \tag{5.25}$$

Substituting eqn. 5.23 gives

$$\begin{aligned} H(S|P) &= -\left[\sum_i \sum_j p(\mathbf{x}_i, R_j) \log p(\mathbf{x}_i, R_j) - \sum_i \sum_j p(\mathbf{x}_i, R_j) \log p(R_j)\right] \\ &= -\sum_j \sum_{\mathbf{x}_i \in R_j} p(\mathbf{x}_i, R_j) \log \left[\frac{p(\mathbf{x}_i, R_j)}{p(R_j)}\right] \end{aligned} \tag{5.26}$$

$$\text{But} \quad p(R_j) = \sum_i p(\mathbf{x}_i, R_j)$$

and

$$\begin{aligned} p(\mathbf{x}_i, R_j) &= p(R_j) p(\mathbf{x}_i, R_j) \\ &= p(\mathbf{x}_i) p(R_j|\mathbf{x}_i) \quad (Bayes' rule) \end{aligned}$$

Therefore

$$H(S|P) = -\sum_i \sum_j p(R_j|\mathbf{x}_i) p(\mathbf{x}_i) \log \left[\frac{p(R_j|\mathbf{x}_i) p(\mathbf{x}_i)}{\sum_{\mathbf{x}_i \in R_j} p(R_j|\mathbf{x}_i)\, p(\mathbf{x}_i)}\right] \tag{5.27}$$

Since $H(S|P) = H(P|S)$

$$H(S|P) = -\sum_i \sum_j p(\mathbf{x}_i, R_j) p(R_j) \log \left[\frac{p(\mathbf{x}_i|R_j) p(R_j)}{\sum_{\mathbf{x}_i \in R_j} p(\mathbf{x}_i|R_j) p(R_j)}\right] \tag{5.28}$$

Note that eqn. 5.28 can be expressed as

$$\begin{aligned} H(S|P) &= -\sum_j p(R_j) \sum_{\mathbf{x}_i \in R_j} p(\mathbf{x}_i|R_j) \log p(\mathbf{x}_i|R_j) \\ &= \sum_j p(R_j) H(R_j) \end{aligned} \tag{5.29}$$

where $p(\mathbf{x}_i|R_j)$ is the probability of pattern $\mathbf{x}_i$ occurring in segment R_j and $H(R_j)$

is entropy of segment R_j. If segment R_j is homogeneous, then $p(\mathbf{x}_i|R_j) = 1$ at some input pattern $\mathbf{x}_i$ and $H(R_j) = 0$.

For any real pictorial data the segments always have some residual variance and so, in general, we look for a significant reduction in $H(S|P)$, rather than $H(S|P) = 0$. Segmentation can then be achieved by terminating segmentation when the rate of change falls below a nominal threshold.

The minimisation of $H(S|P)$ can use either eqn. 5.27 or 5.29. In this study it is assumed that $p(\mathbf{x}_i|R_j)$ are normally distributed (see Appendix E), and the probability of a segment is given by

$$p(R_j) = \frac{\#\{\mathbf{x}_i \in R_j\}}{\#\{\forall \mathbf{x} \in P\}} \tag{5.30}$$

5.5 Spatial-spectral Hierarchical Clustering

The second stage of the Spatial-Spectral algorithm is the clustering of regions generated by GTHS. The spatially connected segments $\{R\}$ are supposed to be homogeneous and representing objects in the image. Some of these segments are similar in the feature space, even though they may be spatially separated, and spectral clustering is required to group these regions into several categories. By doing this we assume each segment represent an object belongs to a given category.

The grouping of segments is similar to the grouping of pixels except the spatial relationship is not considered. The clustering of segments is based on the same principle of stepwise optimization. Given a set of segments $\{R_i\}$ $i = 1, \ldots, m$, these segments are merged based on minimisation of a cost function $C(R_i, R_j)$ when segment R_i and R_j are merged. The objective is to minimize the overall cost of merging:

$$J(\omega) = \sum_{\omega} C(R_i, R_j) \tag{5.31}$$

where ω is all possible partitions.

The cost function $C(R_i, R_j)$ can be one of the distance functions discussed

in section 3.5.4. However, for the purpose of spectral clustering these distance functions are found to be inadequate to discriminate classes, mainly because they do not convey sufficient information about the segments. For example, if only the centroid of two segments are used the variation of the two segments is ignored. Therefore a better similarity measure is required. The overall Spatial-Spectral clustering algorithm can be divided into two stages process: The first stage is either the bottom-up or top-down GTHS, and the second stage is the clustering of segments (Fig. 5.9).

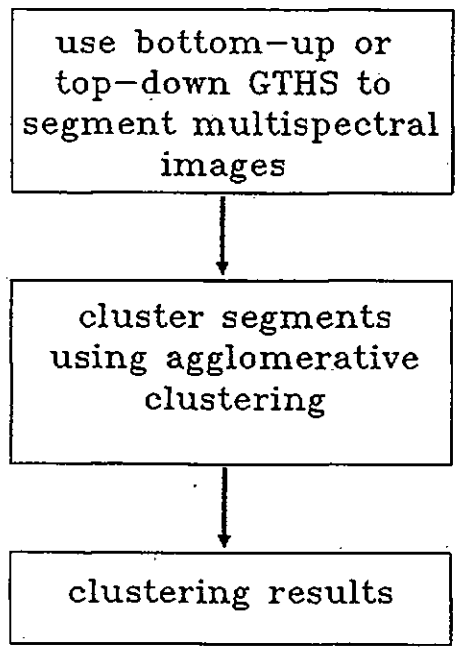


Figure 5.9: Spatial-spectral clustering approach.

A better choice of cost function is the intraset distance which is a measure of variance. The intraset distance is the average within group distance of the merging R_i and R_j. It is noted that this distance has also been used in top-down approach for spatial clustering. The intraset distance is repeated here,

$$\delta_{avg}(R_i, R_j) = \frac{1}{n_p(n_p-1)} \sum_{i=1}^{n_p} \sum_{j=1}^{n_p} \sum_{k=1}^{d} (\mathrm{x}_{ki} - \mathrm{x}_{kj})^2 \tag{5.32}$$

where $n_p = \#\{R_i \cup R_j\}$ and $\mathrm{x}_{ki}, \mathrm{x}_{kj} \in R_i \cup R_j$.

The algorithm of the second stage of the Spatial-Spectral clustering is similar to the first stage and is as follows:

Step 1. Set $r = m - 1$

Step 2. Start with m segments. $P = \{R_1, \ldots, R_m\}$.

Step 3. Find R_i and R_j such that $C(R_i, R_j) = \min_{i \neq j} C(R_i, R_j)$.

Step 4. Merge R_i and R_j, update all pairwise similarities.
Save link connecting R_i and R_j

Step 5. $r = r - 1$

Step 6. If $r = 1$ or $r =$ required number of cluster stop, else goto Step 3.

Step 7. Form clusters using the trees generated.

Again top-down and bottom-up approaches can be implemented using the above algorithm. It has been found that the top-down approach does not offer much advantage over the bottom-up approach.

Intraset distance was used in this study and it does not produce good results because it tends to underestimate the distance between small segments and large segments.

5.5.1 Statistical Hypothesis as a Distance Measure

A better measure is based on statistical hypothesis, which has been used extensively in image segmentation. A statistical test involve testing of a hypothesis and the decision will be simply accept or reject the test. It is noted that for any tests to be developed it is necessary to assume a definite probability distribution for the random variable in the segments, and in this study a multi-variate Gaussian model is assumed. Suppose there are two sets of sample $\mathbf{X}_1, \mathbf{X}_2$, a *null hypothesis* $H_0; \mu_1 = \mu_2$ is to test whether the mean μ_1, μ_2 are the same, It is assumed that the two populations have a common variance covariance matrix. Two types of error can result from this decision. They are:

$$P(\text{Type I error}) \;=\; P(\text{rejecting } H_0 | H_0 \text{ true}) = \alpha$$

$$P(\text{Type II error}) \;=\; P(\text{accepting } H_0 | H_1 \text{ true}) = \beta$$

where H_1 is called the alternative hypothesis and is automatically accepted if the null hypothesis is rejected, so $H_1; \mu_1 \neq \mu_2$.

It is assumed that every points in segments R_i are with the normal distribution, and the null hypothesis is to test whether the two distribution are the same, if the test is rejected it is concluded that the two segments belong to different populations. The statistical decision therefore accepts H_0 if the two distributions have a distance less than a threshold t (confidence level, see Fig. 5.10).

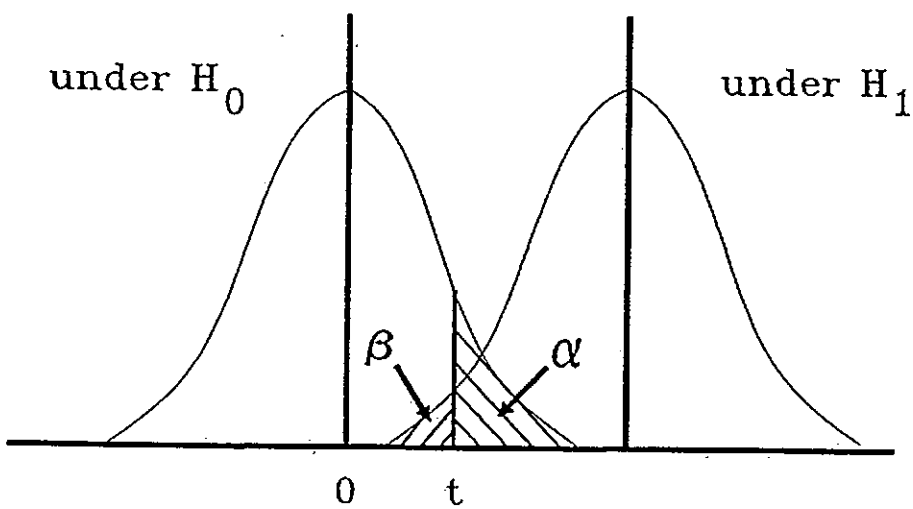


Figure 5.10: Probabilities of error in hypothesis testing.

For simplicity we shall use a single variate example to demonstrate the statistical hypothesis test of two populations. Suppose two segments R_i and R_j with common population variance σ^2 and respective mean m_i, m_j, and all points in each segment has distribution $N(m_i, \sigma^2)$ and $N(m_j, \sigma^2)$, then the distribution of $\mu_i = \frac{1}{n_i}\sum x_{ik}$, $x_{ik} \in R_i$ and $\mu_j = \frac{1}{n_j}\sum x_{jk}$, $x_{jk} \in R_j$ is given by $\mu_i = N(m_i, \frac{\sigma^2}{n_i})$and $\mu_j = N(m_j, \frac{\sigma^2}{n_j})$. Therefore $\mu_i - \mu_j$ is distributed as $N(\mu_i - \mu_j, \sigma^2(\frac{1}{n_i} + \frac{1}{n_j}))$. The null hypothesis for the test will be that there is no difference between the two population means, i.e. $H_0 : \mu_i = \mu_j$ or $H_0 : \mu_i - \mu_j = 0$. So that,

under $H_0, \mu_i - \mu_j$ is $N(0, \sigma^2(\frac{1}{n_i} + \frac{1}{n_j}))$. Therefore the difference of two populations' mean has a normalized distribution (for unknown common population variance, σ^2).

$$T = \frac{\mu_i - \mu_j}{\sigma\sqrt{\frac{1}{n_i} + \frac{1}{n_j}}} = N(0,1) \tag{5.33}$$

Therefore the T distribution can be used as the similarity measure of the two populations. If the common population variance σ^2 is unknown, the unbiased estimate of the common population variance $\hat{\sigma}^2$ is given by

$$\hat{\sigma}^2 = \frac{(n_i - 1)S_i^2 + (n_j - 1)S_j^2}{n_i + n_j - 2} \tag{5.34}$$

where S_j and S_j are the sample variance of R_i and R_j respectively. If $\hat{\sigma}^2$ is used the distribution under H_0 becomes $T(n_i + n_j - 2)$ instead of $N(0,1)$. Eqn. 5.33 can be generalised to multi-variate (Lindeman et al. 1980, pp 183)

$$T^2 = \frac{n_i n_j}{n_i + n_j}(\mu_i - \mu_j)^T \Sigma_{ij}^{-1}(\mu_i - \mu_j) \tag{5.35}$$

where

$$\Sigma_{ij}^{-1} = (n_i + n_j - 2)(A_i + A_j)^{-1} \tag{5.36}$$

$$A_i = \sum_{k=1}^{n_i}(\mathbf{x}_k - \mathbf{m}_i)(\mathbf{x}_k - \mathbf{m}_i)^T \tag{5.37}$$

$$A_j = \sum_{k=1}^{n_j}(\mathbf{x}_k - \mathbf{m}_j)(\mathbf{x}_k - \mathbf{m}_j)^T \tag{5.38}$$

It is noted that eqn 5.35 is the weighted squared Mahalanobis distance of μ_i and μ_j with Σ_{ij} being the pooled sample within-groups variance covariance matrix of population i and j respectively, therefore is a similarity measure of two populations. Hence the statistical distance can be incorporated into the clustering algorithm.

An assumption in the T^2 test is that the two segments under test are having a common covariance matrix. Although this assumption is very restrictive, in general it produces good class discrmination.

In each iteration the two most similar segments R_i and R_j will be merged, therefore type II error β is minimized. This is reasonable because in hierarchical segmentation type II error is considered to be the most serious because merging of two segments which committed a type II error cannot be recovered in subsequent process. Whereas type I error which keep separating two similar segments can be corrected in a following step. Therefore, it is advantageous to keep type II error as small as possible in each iteration, which implies that type I error has to be maximised.

Let D_{ij} be the distance between groups of segments, and d_{ij} denotes the distance between two segments.

The Spatial-Spectral Clustering Algorithm is:

Step 1. Use either top-down or bottom-up GTHS to generate m segments with m suggested by the mutual information models.

Step 2. Store the $m(m-1)/2$ distances (d_{ij}) in a m x m matrix.

Step 3. Find the smallest distance (ties are resolved arbitrary). If this is D_{kq}, merge groups k and q and call the new group r

Step 4. Calculate the distance between the new group r and each of the existing groups. Replace the kth and qth rows and columns of the matrix by the single row/column of new distances, thereby reducing the order of the matrix by one.

Step 5. Goto Step 2 if the number of group is more than one, otherwise, stop if the number of groups equal to the required number of clusters.

Other statistical distance can be substituted for the Mahalanobis distance in eqn. 5.35 such as the Bhattacharyya and Divergence distance, but the statistic of the hypothesis will be unknown because they allows segments to have different covariance matrix. The Bhattacharyya distance of two normally distributed population is (Hand 1981):

$$\delta_B(R_i, R_j) = \frac{1}{8}(\mu_i - \mu_j)^T(\Sigma_i + \Sigma_j)^{-1}(\mu_i - \mu_j) + \frac{1}{2}\log\frac{|\Sigma_i + \Sigma_j|}{|\Sigma_i|^{\frac{1}{2}}|\Sigma_j|^{\frac{1}{2}}} \quad (5.39)$$

The Divergence distance for two normally distributed population is:

$$\delta_d(R_i, R_j) = \frac{1}{2}\mathrm{tr}[(\Sigma_i - \Sigma_j)(\Sigma_j^{-1} - \Sigma_i^{-1})] + \frac{1}{2}\mathrm{tr}[(\Sigma_i^{-1} + \Sigma_j^{-1})(\mu_i - \mu_j)(\mu_i - \mu_j)^T] \quad (5.40)$$

The estimates of covariance matrix Σ and mean μ are given by the maximum likelihood estimator described in section 3.2.

Both Divergence and Bhattacharyya distance allow the two populations to have different covariance matrix, and degenerates to the Mahalanobis distance if two populations have a common covariance matrix.

All three weighted distances have been used in this study and the clustering results are very similar and so only results using eqn. 5.35 will be presented.

The computational cost of hierarchical clustering is high if the number of units to be clustered is large (say > 1000). Note that the clustering stage can be implemented using one of the methods (e.g. stored matrix) described in Anderberg (1973, Ch.6). If using the stored matrix method the complexity is $O(2m^2)$ where m is the number of segments. The contextual clustering methods have been implemented and results compared with those from classic pre-pixel algorithm presented in Chapter 4. It is evident that the contextual clustering produces less noisy results.

5.6 Spatial-Spectral Clustering Results

The Spatial-Spectral clustering algorithms presented in previous sections are demonstrated using three sets of 128 x 128 pixel VIS+IR images (Figure 5.11). These images are subregions of the images taken on 8th, 18th, and 20th March respectively.

5.6.1 Segmentation Results

The different approaches presented in previous sections can be summarised as:

1. Bottom-up approach: Construct a spanning tree of the image graph using Kruskal's algorithm (use either local or global type distance measures), and stop construction of the tree when the required number of segments is obtained.

2. Top-down approach: Construct a spanning tree of the image graph until all pixels are connected (use either local or global type distance measures), cut the tree at the edge which satisfies the minimax or other criteria. To obtain m segments, $m-1$ edges will be cut.

The segmentation results are presented using both top-down and bottom-up approach as well as global and local type distances. Overall we have six permutations:

[S1] Top-down approach using the Minimal Spanning Tree (MST), cut the most weighted $m-1$ edges to obtain m segments.

[S2] Top-down approach using the Minimal Spanning Tree (MST), cut the spanning tree using the minimax criterion.

[S3] Bottom-up approach using the Minimal Spanning Tree (MST).

[S4] Top-down approach using the Centroid Method (CEST), cut the most weighted $m-1$ edges to obtain m segments.

[S5] Top-down approach using the Centroid Method (CEST), cut the spanning tree using the minimax criterion.

[S6] Bottom-up approach using the Centroid Method (CEST).

In Figures 5.12— 5.14 are the segmentation results using the top-down minimax with CEST approach. The six images containing different number of segments (grey-level is the mean of a region), 10 segments, 50 segments, 100 segments,

200 segments, 300 segments, and 600 segments. The results show that the main features can be obtained with 10 segments, however all the details have been lost.

10 segments	50 segments	100 segments
600 segments	300 segments	200 segments

Figure 5.12: Top-Down Minimax CEST segmentation of 8th March images.

10 segments	50 segments	100 segments
600 segments	300 segments	200 segments

Figure 5.13: Top-Down Minimax CEST segmentation of 18th March images.

10 segments	50 segments	100 segments
600 segments	300 segments	200 segments

Figure 5.14: Top-Down Minimax CEST segmentation of 20th March images.

Fig. 5.15— 5.17 shows the corresponding Entropy Loss $H(S|P)$ for the three images. The mutual information model gives the ideal segmentation only when the regions having a uniform gray-level or an impulse like pdf. For real images this is not true and the Entropy Loss reduces to zero when all segments contain just one pixel, i.e. a trivial result. Therefore number of segments is determined by a compromise between the generation of fine detail and basic features. Further more a good segmentation uses the least number of segments to give a maximum amount of information or minimum entropy loss. These Figures also show that for the top-down minimax CEST approach there is the steepest decrease of Entropy Loss from 1 segments to say 100 segments then the Entropy Loss starts to decrease approximately at a constant rate. Referring to the segmentation result this big drop correspond to the generation of the most basic features of the images, when the rate of Entropy Loss becomes constant any further segments generated only refine local details of the images. These effects can be seen in Figures 5.12— 5.14, the 200 segments image is very similar to the 600 segments image. The hierarchical characteristic is also demonstrated in these example. At the highest level the whole image belongs to one segments and at every lower levels a segments is always smaller than or equal to the same region which contains it at a higher level. This also means that the region which remains unchanged at the ith and jth level will have their region boundary unchanged.

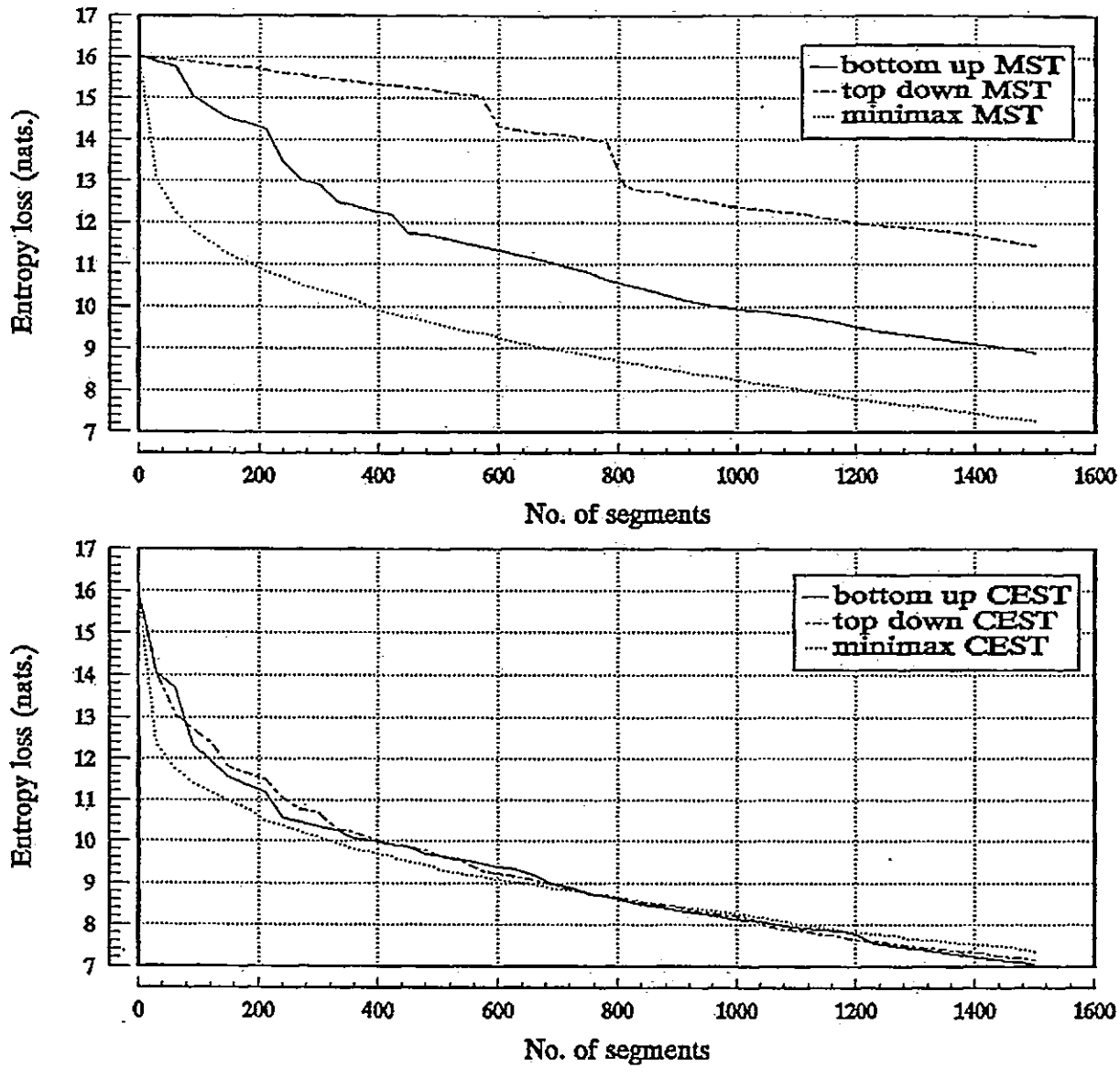


Figure 5.15: Entropy Loss of different segmentation approaches for 8th March images.

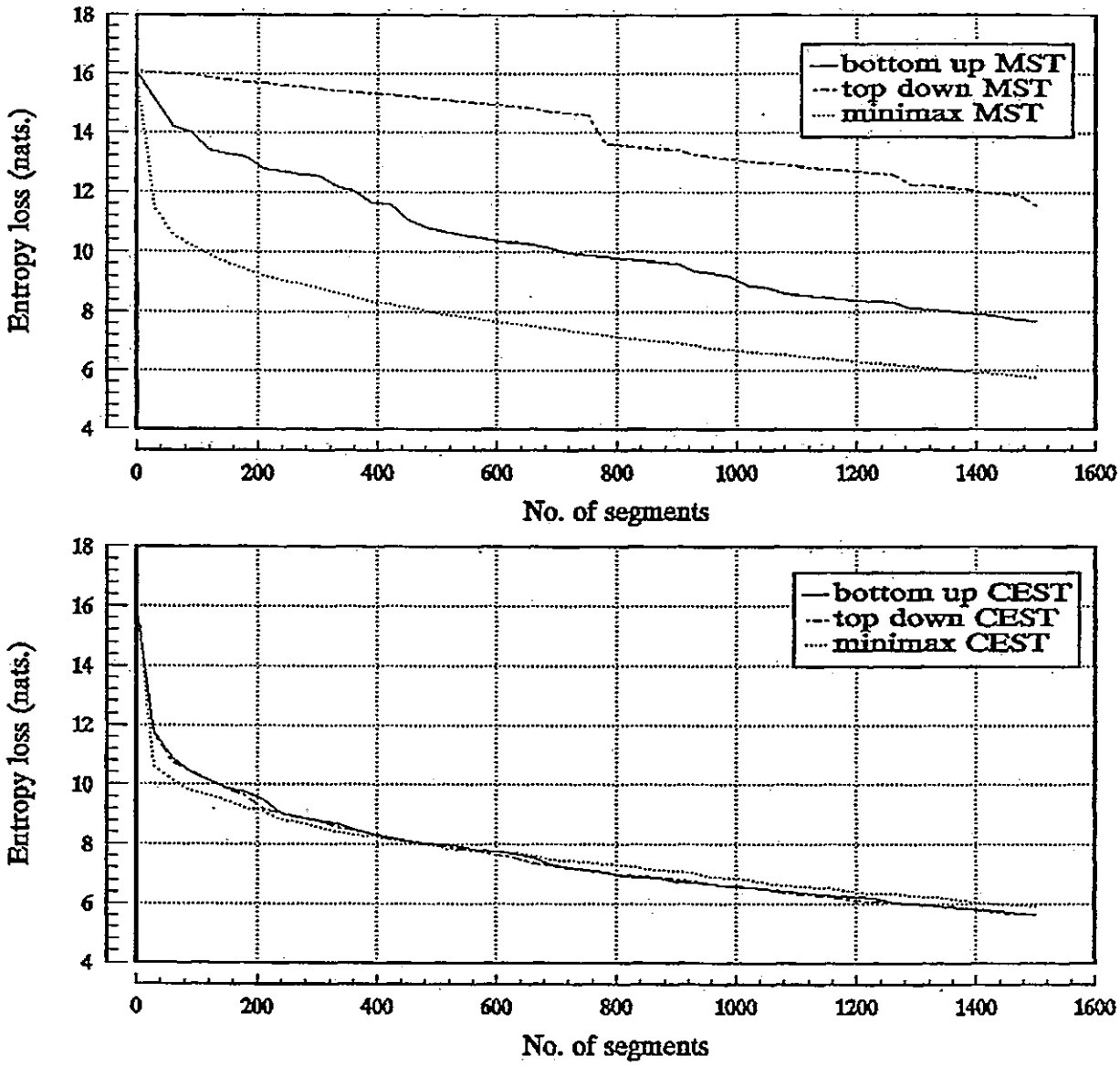


Figure 5.16: Entropy Loss of different segmentation approaches for 18th March images.

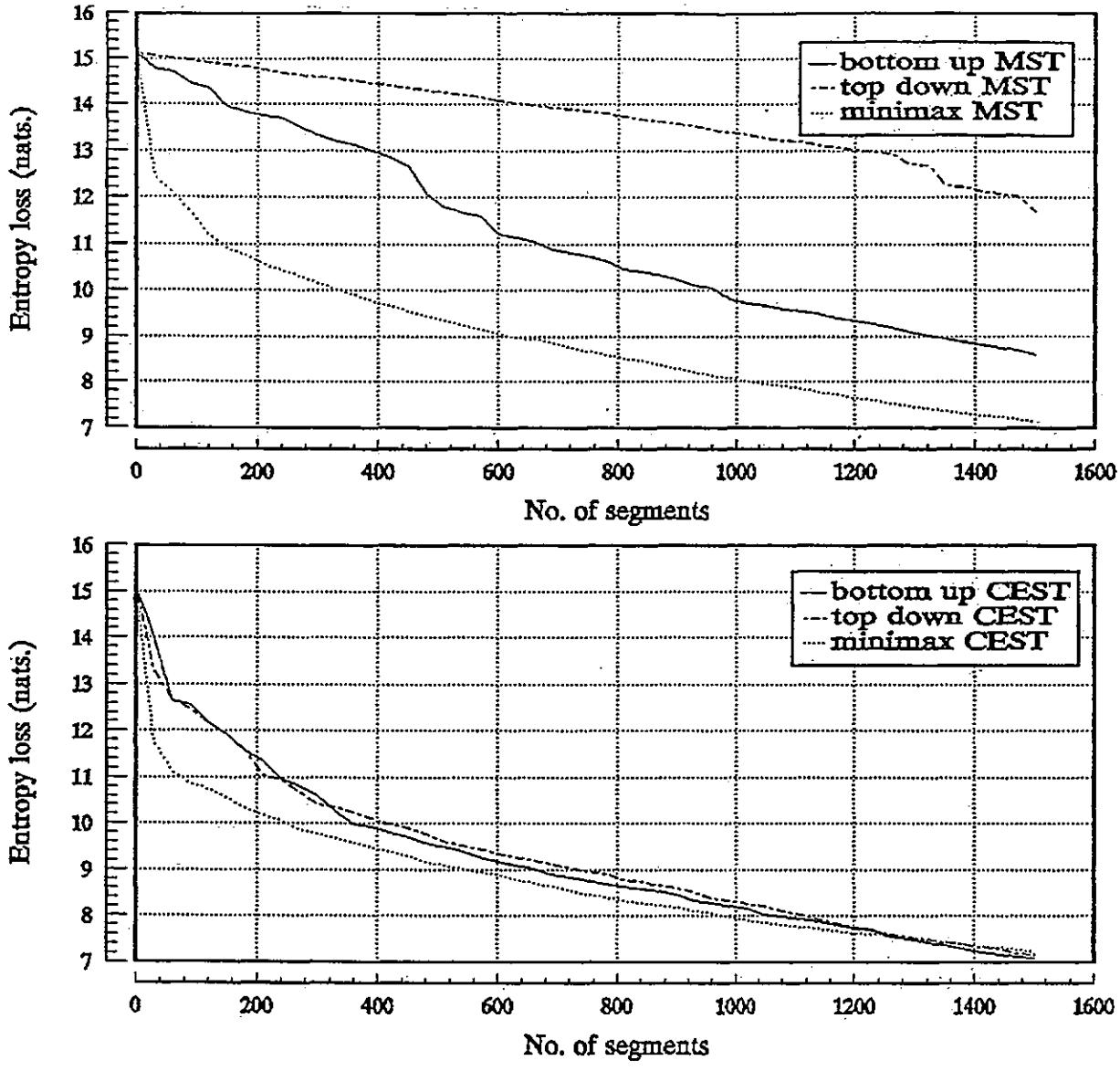


Figure 5.17: Entropy Loss of different segmentation approaches for 20th March images.

The result of the six different segmentation approaches are shown in Figures 5.18— 5.20. These images are all with 300 segments. This number of segments is chosen as the optimal number as suggested by the mutual information model. It is noted that 300 segments is only optimal for top-down approaches and approaches using global distance i.e. CEST. It can be seen that at 300 segments the Entropy Loss is decreasing at a constant rate and this compare well with the segmentation result. The best segmentation, as seen in Figures 5.18— 5.20, is the top-down minimax approach and, this also agrees with the Entropy Loss curves. The results using other CEST approaches follow closely. The worst result is being the top-down MST, at 300 segments only trivial segments are generated. These results demonstrate the effect of using different distance functions and, clearly, global type distances generate better segments than the local type distances, but at higher computation cost.

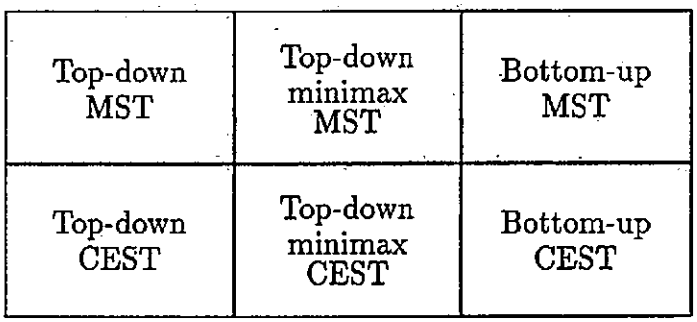


Figure 5.18: Comparison of different segmentation approaches on 8th March images (the number of segments = 300 in each case).

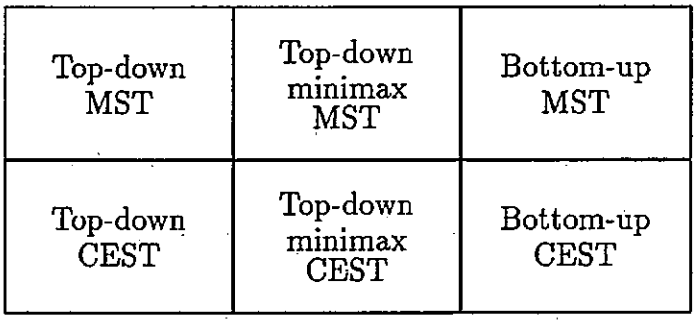


Figure 5.19: Comparison of different segmentation approaches on 18th March images (the number of segments = 300 in each case).

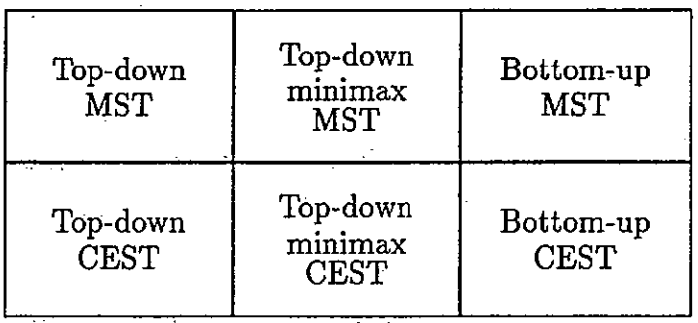


Figure 5.20: Comparison of different segmentation approaches on 20th March images (the number of segments = 300 in each case).

The mutual information model has found to be a useful tool for quantitative monitoring of segmentation. It provides a useful indication for segmentation comparison and suggest a suitable number of segments. This has been demonstrated in Figures 5.18— 5.20 where top-down and bottom-up approaches using MST are with a much higher Entropy Loss than other approaches. The effectiveness of the mutual information model is further illustrated in Figures 5.21— 5.23. In these figures images are having the same Entropy Loss (i.e. different number of segments), it can be seen that all images are very similar (except for top-down MST approach). Hence the mutual information model is rather accurate in describing the information content of the segments.

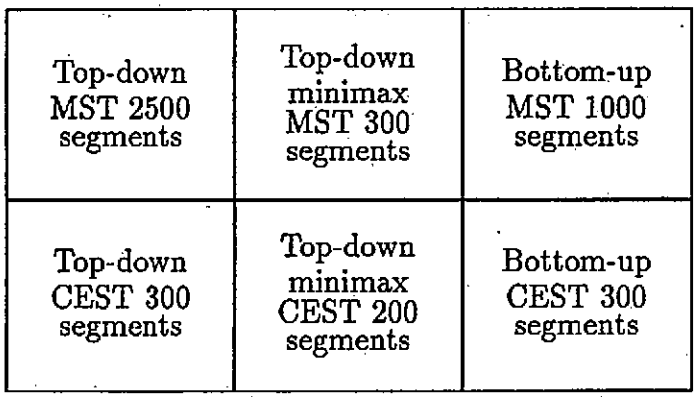


Figure 5.21: Different segmentation approaches with same Entropy Loss, 8th March images.

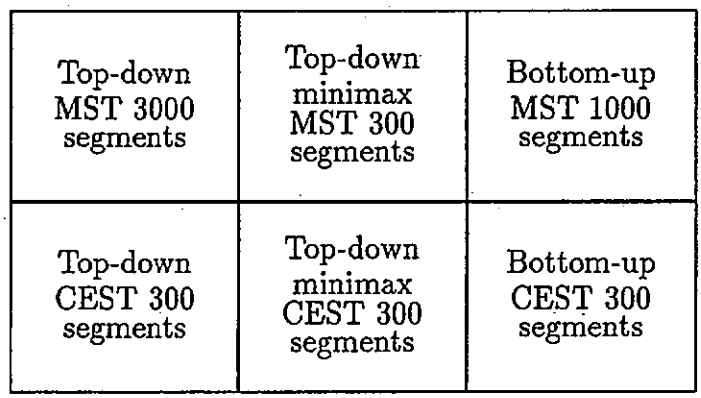


Figure 5.22: Different segmentation approaches with same Entropy Loss, 18th March images.

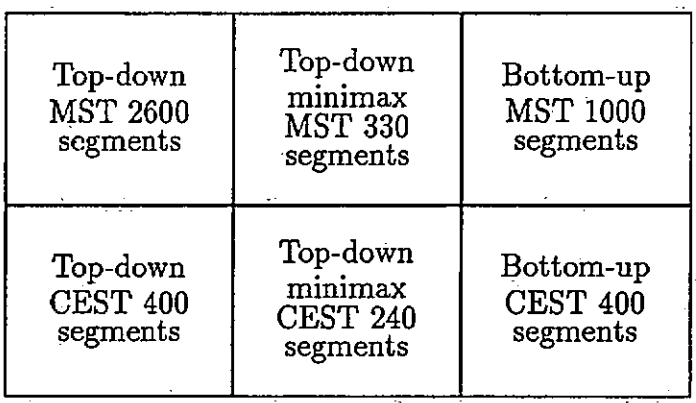


Figure 5.23: Different segmentation approaches with same Entropy Loss, 20th March images.

5.6.2 Properties of MST Segmentation

Basically, the problem of using MST for segmentation suffers the same problem found in hierarchical clustering. The most obvious effect is the "chaining" effect. The cause of this is due to the use of localised distance measure $\delta_{min}(T_i, T_j)$ (eqn. 5.30). Due to the chaining effect two different region can be connected by a series of links with small weight. Therefore the identification of inconsistent edges becomes very difficult if not impossible. Because only localised distance is used, noise in the image, which may be isolated pixels, are identified as separated regions. This means that the MST is sensitive to noise.

5.6.3 Properties of CEST segmentation

The construction of CEST uses global information and therefore it is not sensitive to image noise, and better use of spatial information also generates segments which represent the most important features first.

5.6.4 General Properties of GTHS

1. Spatial information about neighbouring pixels is used, unlike most statistical segmentation e.g. per-pixel clustering. Spatially connected pixels are grouped together if they form a homogeneous segment.

2. Region boundaries are defined very accurately. Jagged edges are not produced as they are, in rectangular segments (Robertson 1973) or strip forming (Nagy and Tolaba 1972). Results in this chapter have been scaled by two so the boundaries may look rather rough.

3. The spanning tree contains all the information needed for splitting the image into any regions in a hierarchical way.

4. Splitting or merging of regions, does not alter the boundaries of other regions.

5.6.5 Clustering Results

This section presents the Spatial-Spectral clustering results. In the last section it was found that top-down minimax CEST approach gives the best segmentation, therefore this approach is chosen to demonstrate the Spatial-Spectral clustering.

Figures 5.24— 5.26 are the results of the Spatial-Spectral clustering using different numbers of segments. These results (all with 5 clusters) show that the choice of number of segments can significantly affect the clustering result. The number of segments are (a) 10 segments, (b) 50 segments, (c) 100 segments, (d) 200 segments, (e) 300 segments, and (f) 600 segments respectively. Inspection of the histogram shows that the images on 20th March (Fig. 5.26) is most difficult to cluster because of the fuzziness of pdf boundaries.

10 segments	50 segments	100 segments
600 segments	300 segments	200 segments

Figure 5.24: Spatial-Spectral clustering of 8th March images with different number of segments (the number of clusters = 5 in each case).

10 segments	50 segments	100 segments
600 segments	300 segments	200 segments

Figure 5.25: Spatial-Spectral clustering of 18th March images with different number of segments (the number of clusters = 5 in each case).

10 segments	50 segments	100 segments
600 segments	300 segments	200 segments

Figure 5.26: Spatial-Spectral clustering of 20th March images with different number of segments (the number of clusters = 5 in each case).

For the 8th March images the results are very similar with 200, 300 and 600 segments. However, with 600 segments small fragments begin to appear, this implies that the cluster becomes more "noisy". The effect is undesirable because the objective of Spatial-Spectral clustering is to reduce boundaries noise.

The results for 18th March are also very similar with 100, 200, 300 segments. The result with 600 segments is rather different and the Cirrus at the edge has been seriously underestimated, and in turns overestimated the subpixel cumulus underneath it.

On 20th March the results are different for all number of segments! The result using 200 segments has assigned a large area of low cloud to sea. The result with 300 segments has estimated the low cloud well and an area of Altostratus has also been identified correctly. With 600 segments the subpixel cumulus at the top has been separated from the sea and start to look "noisy" and the Cirrus in the middle has been underestimated. Overall the best clustering is obtained using 300 segments which is suggested by the Entropy Loss curve. However, the subpixel cumulus at the top has been assigned to cloud free sea, if the objective for classification subpixel cumulus, it will be seriously underestimated.

These results suggest that the optimal number of segments should be chosen as soon as the Entropy Loss starts to decrease at a constant rate.

Figures 5.27— 5.29 shows the comparison of the Global-Local clustering algorithm and the Spatial-Spectral clustering algorithm. These results in general are very similar, but the Spatial-Spectral clustering produces clusters with much cleaner boundaries, as expected. Due to the lack of ground truth observation, it is not possible to quantitatively check the accuracy of the results. Nevertheless these results agree well with visual inspection of the original image, and the objective of using spatial information has been achieved in that the main objects in the image are better defined while none of the boundaries have been substantially shifted.

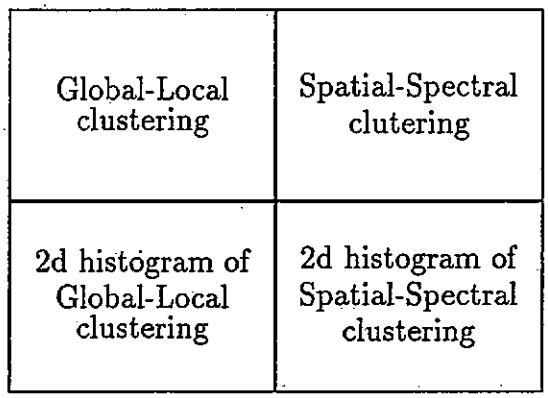


Figure 5.27: Comparison of Global-Local clustering algorithm and Spatial-Spectral clustering algorithm on 8th March images.

Global-Local clustering	Spatial-Spectral clutering
2d histogram of Global-Local clustering	2d histogram of Spatial-Spectral clustering

Figure 5.28: Comparison of Global-Local clustering algorithm and Spatial-Spectral clustering algorithm on 18th March images.

Global-Local clustering	Spatial-Spectral clutering
2d histogram of Global-Local clustering	2d histogram of Spatial-Spectral clustering

Figure 5.29: Comparison of Global-Local clustering algorithm and Spatial-Spectral clustering algorithm on 20th March images.

The difference between the two clustering algorithms is largely due to the decision boundaries in the feature space. It is found that the Global-Local clustering algorithm tends to produce very tight clusters, therefore vectors at the 'base' of a cluster with small variance are assigned to clusters with larger variance. On the other hand, boundaries of a small variance cluster are extended further away from the 'base' by the Spatial-Spectral clustering algorithm. These valleys between each pdf correspond to pixels which are a mixture of more than one class.

Since clustering usually is not an end in itself, clusters with clean boundaries are important to the success of the subsequent machine processing, i.e. cloud wind vector estimation.

5.7 Summary

A Graph Theoretic Hierarchical Segmentation approach is introduced. It generates segments by clustering of the spatial space, and several examples of distance measure have been proposed to construct the image spanning tree. Two segmentation approaches (bottom-up and top-down) based on stepwise optimization are proposed. Distance functions that use global information are found to produce better results that distances using local information. It is also noted that global information can be better utilized by the minimax approach.

A mutual information model has been developed and was shown to be valuable in monitoring segmentation and determination of the optimum number of segments.

A Spatial-Spectral clustering algorithm based on GTHS and stepwise optimization is also introduced. A statistical hypothesis has been used as similarity measure of segments. The Spatial-Spectral clustering algorithm is capable of generating clusters with clean boundaries. The algorithm is unsupervised and required only two clustering parameters, 1) the number of segments, and 2) the number of clusters. It is noted that these algorithms are computation intensive, although their efficiency can be improved by better algorithms and data structures.

Chapter 6

Computation of Cloud Motion Wind (CMW) Vectors

The use of geostationary satellites as a source of wind observations was suggested by Widger and Tourat (1957) before the launch of TIROS I. Today, the wide coverage area of satellites permits wind estimation in remote areas such as polar and ocean areas, and this information is essential in understanding the global climate.

This Chapter starts by introducing the essential elements of cloud motion determination, i.e.: 1) cloud tracer selection, 2) cloud target tracking, 3) cloud height assignment, and 4) cloud motion vector editing. This is followed by detailed discussion of image tracking methods, image registration, image rectification and calculation of wind speed and direction. A new automated cloud motion scheme based on these elements is then presented, and this is used to test the performance of clustering applied to cloud wind determination. Results indicate that clustering is an effective approach for tracking cloud motion.

The Global-Local clustering scheme presented in Chapter 4 is used in the cloud wind tracking scheme because it is very efficient. The concept of using clustering is to partition the image into a number of regions with each region representing a distinct class in the feature space. Some of these classes represent cloud types at different height level, and each class is then tracked individually.

6.1 Elements of Cloud Motion Wind Determination

This section introduces the concepts behind cloud motion wind determination, and implementation details are discussed in next section. Determination of cloud motion winds require both meteorological and image analysis understanding.

The basic procedure to compute wind data is (Hubert 1976),

1. select suitable cloud targets,
2. track the selected targets,
3. assign heights to the resulting vectors,
4. edit the set of vectors.

In general, the first and last steps are related. For example, cloud motion may be measured with little discrimination between suitable and unsuitable tracers. A careful editing procedure is then needed to delete erroneous wind vectors. On the other hand, careful selection of targets (only those advected by the wind) means that little editing of wind vectors is necessary. Usually, automatically derived wind vectors require more careful editing than manually derived wind vectors.

6.1.1 Wind Tracer Selection

The main objective of this study is to investigate the problems of cloud motion tracking in multi-layer areas. Multi-layer cloud areas can be found in most weather images and the primary meteorological interest is to derive low and high level winds from sequence of images. Reliable cloud motion wind can only be obtained by tracking of cloud targets which are advected by air motion in the atmosphere. For automatic cloud motion tracking, identification of the follow cloud types is important (Parikh 1976):

1. Low level cloud (stratocumulus, stratus, cumulus).

2. Mixed cloud (semi-transparent cirrus on top of lower level cloud).

3. Cumulonimbus (not suitable for tracking).

4. High level cloud (cirrus, cirrocumulus, cirrostratus).

The above cloud types are classified by their cloud base height. The division of clouds into low, middle, high, and clouds of vertical development (cumulonimbus), is the one which is most relevant to the problem of wind velocity estimation. The identification of different level of clouds are generally by their spectral and/or textural features in the visible and infrared bands.

The determination of height level of semi-transparent cirrus is a widely recognised difficult problem. Firstly, semi-transparent cirrus cloud is very difficult to identify using computer. Secondly, cirrus cloud temperature can only be estimated with corrected infrared radiance, since the background radiance interferes with the cloud radiance. As a result thin cirrus with middle clouds beneath will appear much hotter than the same cirrus by itself (see Fig. 6.1). The emissivity problem for cirrus clouds is discussed in further detail in Bowen and Saunders (1984). For low level clouds, such as clusters of small cumulus which can not be resolved by the satellite resolution (subpixel cumulus), the grey level in the infrared image will appear darker than the grey level corresponding to height of cloud top because radiation from the warm surface is combined with cloud top radiation in the sensor's field of view.

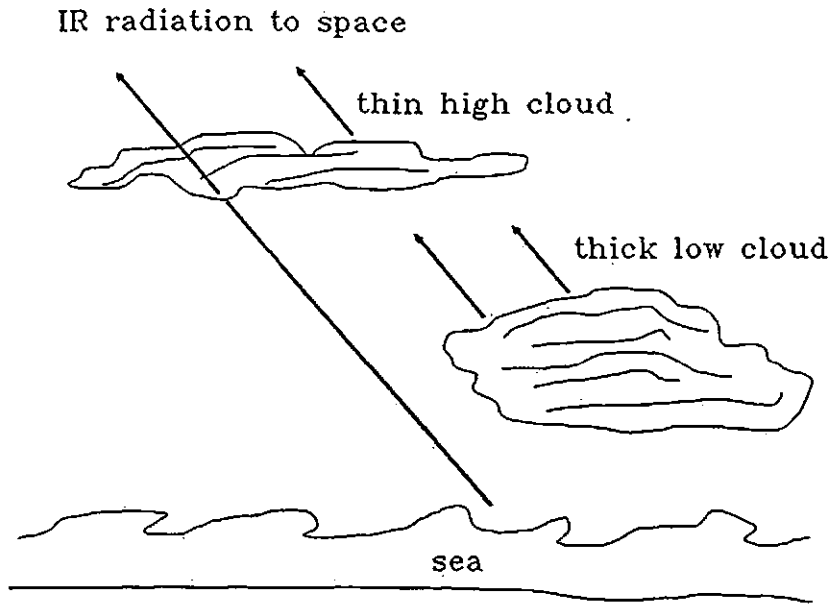


Figure 6.1: The semi-transparent problem: thin cloud such as cirrus often appears to be of warmer because background radiation is confused with the actual radiation.

Availability of infrared images helps to remove some uncertainties which occur when only visible images are available. For example many middle clouds are as bright (in the visible) as lower stratocumulus, but because the latter are almost always warmer they appear much darker in infrared images. Another example is the identification of thin cirrus, they are very poorly seen in the visible image but are prominent in the infrared.

The selection of cloud targets can be assisted by analysis of the synoptic situation. Hubert and Whitney (1971) provided some guidelines for selecting and classifying cloud targets. The synoptic situation is first determined. Grouping of clouds is then deduced by judging how the cloud behaviour fits the appropriate synoptic model, and the target is rejected if its behaviour is not reconciled with the synoptic situation. Once the synoptic situation is determined, observations of cloud characteristics and cloud motion can aid segregating cloud layers and in specifying cloud types. Selection of passive tracers is made concurrently with the

classification of clouds. Some guidelines are:

1. Follow the same point on cloud clusters and patches rather than lines, bands, or areas of equal brightness.
2. Use only those clouds moving at speeds and in a manner that is consistent with the synoptic situation. Beware of motions which appear to move through a pattern of cloud, alternately suppressing and enhancing brightness. This type of motion often conflicts with motion of the individual cloud elements in the same layer and is probably due to gravity waves. Upward motion in crests of such waves enhances cloudiness, and downward motion in troughs suppresses cloudiness. These motions are frequently seen in inversion-dominated low clouds and at various upper levels near cloud fronts. As expected from theory, the orientation of waves and their direction of motion bear no fixed relation to the ambient wind.
3. Use clouds which show the least change during the time-lapse sequence.
4. Take care in tracking clouds that appear to penetrate vertical shear layers. In these cases, try to track the upshear edge rather than the centre of mass. For example, in areas of active convection the cloud area grows rapidly because of anvil growth. The origin of the anvil (the brightest area at rear of the growth area) moves with the middle- and low-level wind. The leading edge of the anvil, while advancing with the high-level wind, may be moving more slowly than the wind because of evaporation. Thus the leading edge of growing cirrus plumes should be avoided.

The manual tracer selection is based on the synoptic situation—location of the fronts and cyclones, location of the major troughs and ridges—and the atmosphere

processes. The wind tracer should located in area which air motion appears to be consistent with the situations revealed by animated sequence of images.

However, even if the direction of motion of the cloud target is consistent with the synoptic situation, speed may be affected by non-advective mechanisms acting in the same direction as expected wind flow. Analysis of cloud type and patterns of cloud motions will often reveal non-advective mechanisms such as vertical updrafts in cumulonimbus clouds and gravity waves (Parikh 1976). Therefore accurate estimation of a wind vector (including height) from a cloud target whose motion is representative of the ambient wind flow can only be achieved by choosing those targets whose size, shape, and brightness are persistent and temperature can be accurately estimated.

The lifetimes of high and low level cloud are very different. In the upper tropopause cloud usually has a short lifetimes (less than 30 minutes), while low level cloud can have longer lifetimes. METEOSAT transmits one set of images every 30 minutes, and so this time resolution only allows cloud tracer with a lifetime longer than 30 minutes to be tracked. Although individual cloud elements can grow and disappear in a short time, mesoscale cloud patterns usually exist more than 30 minutes, especially for low level cloud due to the persistent mesoscale circulation systems (Hubert 1979).

Due to evaporation and condensation, cloud is not a passive tracer of wind, and so it is important to select a target which is believed to be good tracer. Numerous studies have been done on the selection of a wind tracer. Generally, cumuli cloud is a good tracer for low level wind, and cirriform cloud a good tracer for high level wind (Hubert and Whitney, 1971, Hasler et al., 1979).

The low level cloud targets are predominantly convective clouds over ocean, and cloud patterns composed of these cloud types are well suited for lower level cloud tracking (850mb). In the upper level, most potential cloud targets are layer cloud, and these patches of layer cloud often change slowly and may exist for hours. Apparently they persist because large scale vertical motion inhibits their evaporation (Hubert and Whitney 1971).

Targets which are developing or dissipating, as well as cloud types representing lee waves, vertical development, banner clouds, cumulonimbus tops and the edges of frontal cloud should not be selected.

The selection of a cloud targets is therefore a highly complicated decision process, and this is usually done by a trained meteorologist. A compromise must be made in order to extract cloud wind fields using computer programmes. For example, cloud motion can be tracked with little discrimination between suitable and unsuitable tracers. A careful editing procedure is then used to delete erroneous wind vectors. This strategy is usually adopted in automated cloud motion systems, whilst manual methods of cloud tracking usually employ careful selection of cloud targets. The former strategy is adopted in this work.

6.1.2 Tracking the Selected Targets

After the selection of cloud target for wind determination is completed, its displacement between successive images is measured. Cloud displacements are measured using image matching techniques. The matching can be carried out using the infrared or visible image although usually an infrared image is used to track the cloud motion because of its direct relation to cloud top temperature.

Cloud tracking can be done either manually or automatically. Manual methods usually use a movie loop technique which displays a time sequence of images and then a target is selected manually. The tracking is either done by marking the initial and last positions of the target or matching the target using image matching algorithms on a computer interactively.

Cloud tracking are usually computed using image matching techniques. A reference template (subimage) is being searched for in a larger template taken at a time before or after the reference. The reference template usually contains cloud types existing at different height levels, and air motion at different level may differ radically, and the different movement of clouds can confuse the matching process (Arking et.al 1978). So accurate cloud motion can only be obtained by tracking clouds belonging to the same height level.

Cloud motion tracking was first done manually by viewing a sequence of images project on to a screen. Izawa and Fujita (1969) used visible channel images of ESSA and ATS for cloud tracking using manual movie loop techniques. The images were converted to a Mercator map projection and displayed in a movie loop and cloud targets were tracked manually by measuring the displacement on the screen. They found that velocities of high and low clouds correspond approximately to winds at about the 200mb and 800mb levels respectively.

Manual tracking of cloud motion using a movie loop requires a trained meteorologist, and is time consuming and subjective. Leese et al. (1970, 1971) computed wind vector using cross correlation on raw and binary images. The binary image was obtained by an empirical threshold to separate the cloud and background. They found that it is important to ensure no mixed layers of clouds are present when using cross correlation, otherwise the correlation surface would have several peaks in addition to the true one. However cross correlation can produce a better speed resolution than manual methods while there is only minor difference in direction. In the wind systems described by Wilson (1984) a tracking technique called Sequential Similarity Detection Algorithm (SSDA) was used (this is computationally faster than cross correlation).

Typical automated wind systems use sequence of three images for cloud tracking. The middle image in the time sequence is divided into small segments. Statistical analysis is then performed on each segment for presence of cloud targets, if targets are found, they are tracked by image matching techniques in the previous and next image. Since the target checking is rather crude compared with human decision, the tracking results need editing before being used for other purposes.

In this thesis experimental results show that SSDA is in fact a more reliable tracking method than cross correlation. The desire to compute more cloud wind vectors requires automatic algorithms to replace the human operator. However, cross correlation can be confused by multi-layers of cloud and produce spurious winds, and it was thought that cloud could be tracked better if different layers could be separated prior to cross correlation (Hubert and Whitney 1971, Parikh

1976).

The author used clustering to classify the middle one of a sequence of VIS and IR three multi-spectral images. Ideally these clusters are homogeneous segments and each segment represents one class of object. Clouds at different levels have different spectral features (Chapter 1), so a cluster should be a cloud class at a particular level and confusion due to different direction of movement can be alleviated as this work has proved. Comparison of the author's and current approach is outlined in Fig. 6.2.

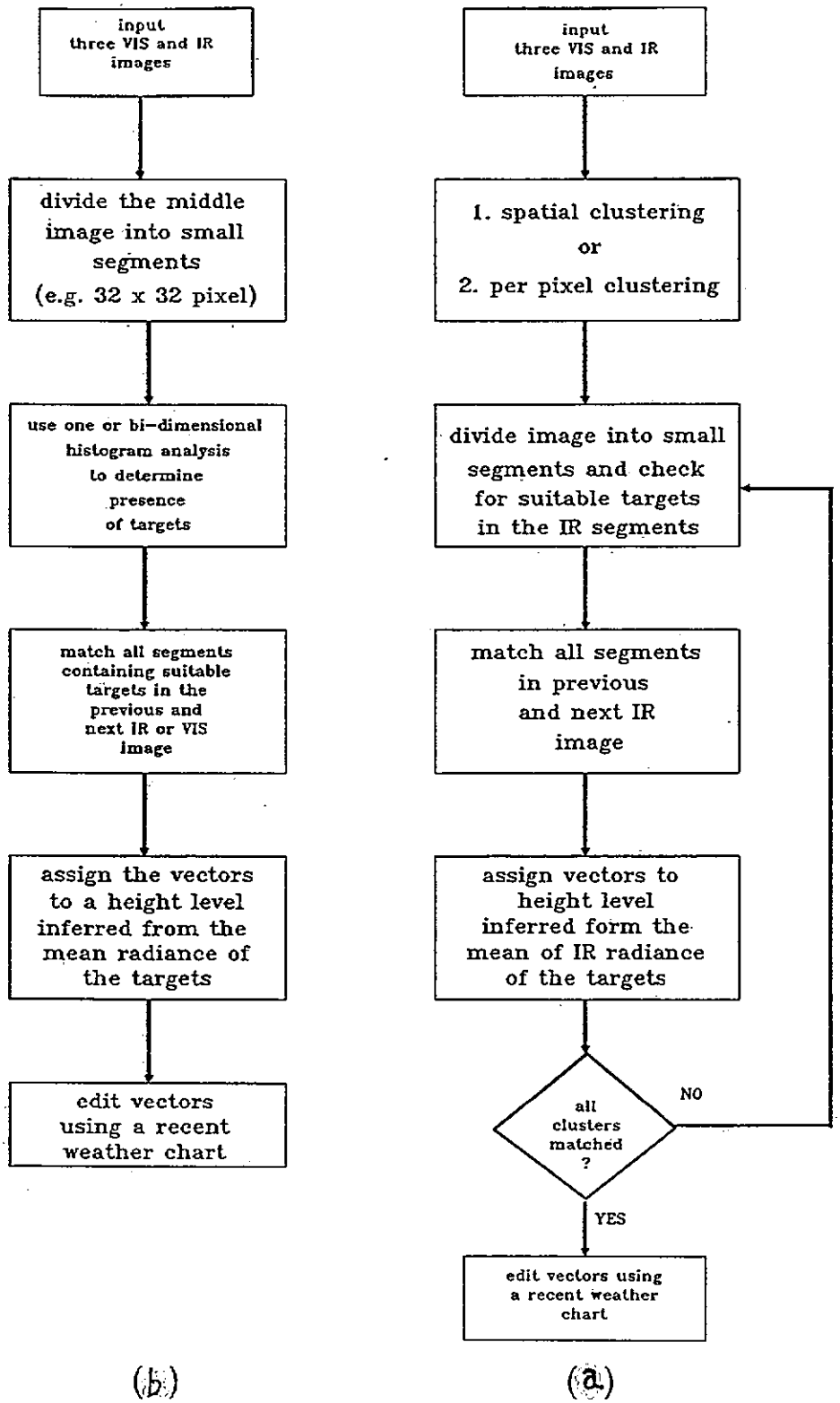


Figure 6.2: Comparison of a) the author's and b) current approach for cloud motion tracking.

In fact the tracking of cloud motion has stimulated the study of motion fields from a time-varying image sequence. Motion fields also find important application in 2 or 3 dimensional computer vision. Studies by Abidi and Gonzalez (1987) applied optic flow for tracking cloud motion in a tornado. Optic flow techniques concern the determination of the "motion" of the individual pixel locations by using intensity data in a sequence of images. The resultant optic flow field is the field of 2-D pixel "velocity" vectors. They used a differential motion measurement approach that was capable of determining the global motion and a more accurate technique called correspondence-base technique which is based on cross correlation. They find that the classical correspondence-base technique was superior in detecting rotational motion, whereas the differential technique fails to detect the detail of any local motion, although it is faster than cross correlation. Optical flow is a major research area on its own and will not be pursued here.

6.1.3 Height Assignment

Cloud height can be inferred from the infrared band image. METEOSAT has an on board calibration routine which is carried out regularly to compensate for changes in the respond characteristics of the detectors of the satellite radiometer. However, the radiation reaching the sensor does not correspond to the temperature of the object being sensed. Therefore accurate cloud top temperature requires a correction for atmospheric absorption and re-emission (the absorption is mainly caused by the water-vapour exist in the atmosphere). The actual temperature of a surface observed by the satellite can be retrieved by computing the atmospheric absorption. These computations require information on the vertical temperature and humidity structure of the atmosphere. The determination of the atmospheric correction at ESOC (Schmetz 1986) is done by radiative transfer calculations for 110 atmospheric profile models. These atmospheric correction models are stored in look-up tables, and the actual atmospheric profiles are assigned to one of the models.

The most significant error in cloud wind is the height assignment of the vec-

tor. Cloud height is measured either in pressure (mb) or distance (m). Satellite observation is directly above the cloud, and so it only provides a brightness temperature of the cloud top. However, for many convective cloud, the speed derived from cloud motion corresponds strongly to the speed near the cloud base. For stratus, on the other hand, the speed may be appropriately assigned to the cloud top. Low-level clouds over oceans are assumed to be cumulus or stratocumulus. The low cloud motion vectors obtained over oceans are assigned to the 900mb pressure level (Hasler et al. 1979), which is statistically near low cloud base. There is no reliable way of measuring cloud base height from geosynchronous satellites, but cloud base can be estimated from aircraft reports, or soundings. In frontal region cumulus cloud winds may be assigned to the middle of the cloud layer. High-level Cirrus cloud winds should be assigned to the mid-cloud level or top level. Satellite wind is a good estimator of level wind at most equatorial through mid latitude ocean area.

Even if the height of the cloud top is known, this may not be the correct level to which the vector should be assigned (Schmetz and Holmlund 1990). Cloud top temperatures frequently do not provide adequate vector height information, because cloud has thickness and different cloud types represents wind best at cloud base, middle or top.

Cloud height is inferred from the infrared radiance but semi-transparent cloud seen in the infrared image has emissivity much lower than unity. The radiation measured is a combination of radiation from cloud-top and from an underlying surface or cloud layer (Fig. 6.1). Therefore, the brightness temperature must be corrected before it is used to infer cloud-top height, otherwise the height will be considerably lower than the true cloud-top height.

The infrared radiance observed by the satellite β_T from a cirrus cloud is approximately given by (Shenk and Curren 1973)

$$\beta_T = \varepsilon\beta_h + (1-\varepsilon)\beta_b \tag{6.1}$$

where

ε = emissivity which is estimated on the basis of empirical rules

$1-\varepsilon$ = high cloud transmittance,

β_h = Plank radiance form high cloud,

β_b = radiance of underlying surface, observed by satellite in neighbourhood of the high cloud, and

β_T = radiance form cloud observed by satellite.

Solve for β_h using eqn. 6.1

$$\beta_h = \beta_b - \frac{\beta_b - \beta_T}{\varepsilon} \tag{6.2}$$

The surface radiance β_b may be inferred from the cloud free pixels in the neighbourhood of the high cloud, and β_T is estimated form be observed infrared grey level, and a value of ε is chosen subjectively according to the general appearance of the cloud. Hubert (1979) shows that an error of 0.05 in $\varepsilon = 0.7$ would produce errors of -50mb or +40mb, respectively. If ε were only 0.25, however, an error in emissivity of 0.05 would produce excessive height errors. Therefore this simple method is only reliable for cirrus with emissivity $\varepsilon > 0.5$. Noted that the emissivity of cirrus clouds can be as low as 0.1.

Szejwach (1982) used the IR and WV channel (METEOSAT) to estimate cirrus cloud temperature, they showed that by substituting the IR and WV radiance of β_h, β_b and ε into eqn. 6.1 a linear relationship between $\beta_{T(IR)}$ and $\beta_{T(WV)}$ is obtained

$$\beta_{T(IR)} = a\beta_{T(WV)} + b \tag{6.3}$$

Eqn. 6.3 is independent of the emissivity $\varepsilon_{(IR)}$ and $\varepsilon_{(WV)}$. Hence, a set of n measurements obtained in both channels over several areas of different optical

thickness from the same cirrus clouds can be used to solve for the cloud top temperature graphically.

The ESOC uses similar methods as Szejwach (1982) to estimate high level cloud emissivity. A set of functions relating $\beta_{T(IR)}$ to $\beta_{T(WV)}$ for different zenith angle ranges was computed form a collection of representative atmospheric soundings. The mean function for each zenith-angle is not constant and depends on the calibration factor for the two channels. In addition the function has to be computed for a complete range of cloud-top heights and model atmospheres. Detail of the correction method can be found in Bowen and Saunders (1984).

Wind vectors can be assigned to cloud top or cloud base once the cloud temperature has been computed. Another method for wind vector height assignment is "Level of Best Fit" (LBF) (Hubert and Whitney 1971). Wind vectors are assigned to level with smallest difference compared with the wind profile of a nearby rawinsonde. The LBF method sometimes gives unrealistic result (Lee 1979), but compare with other methods which requires temperature correction it is simpler to use. One problem in using the LBF is the recurrence of similar winds at different levels. This means cloud motion may resemble an analysis wind at more than one level, and so allow the assignment of a level at a much different height from the actual cloud. Because of it simplicity the "Level of Best Fit" approach is used in this study.

If no correction is required the cloud top temperature can be estimated directly form the infrared or water vapour grey level assuming the cloud target has a emissivity of one.

$$\beta_{T(IR)} = \beta_{T(WV)} = (C - C_o) \text{ x } CAL \tag{6.4}$$

where: $\beta_{T(IR)}$ is the $11\mu m$ IR radiance $(Wm^{-2}sr^{-1}ct^{-1})$

$\beta_{T(WV)}$ is the $6\mu m$ WV radiance $(Wm^{-2}sr^{-1}ct^{-1})$

C is the pixel count

C_o is the space count ($C_o = 5$ for images in this study)

CAL is the MIEC (METEOSAT Information Extraction Centre) calibration coefficient ($CAL \approx 0.077$ for images in this study).

The temperature is then obtained using a radiance to temperature look up table.

6.1.4 Editing Wind Vectors

The last step in wind extraction from geostationary satellite images is quality control or editing. This is applied to both manual and automatic methods. Usually editing for manually derived vectors is based on the subjective view of the meteorologist, whilst more objective methods are employed for automated scheme.

The automatically derived vectors are usually selected by applying some internal test or on comparisons made with the set of derived wind vectors using other methods. Typically this is done by examine the quality of correlation function, such as number of peaks (vector ambiguity); value of the peak coefficient (only suitable for normalised cross-correlation, low values indicates non-persistent cloud pattern); a lack of sharpness of the peak.

Another strategy to improve confidence level is to use an image triple to derive two vectors. If they are not symmetric to a certain degree, this indicates a possibly unsuitable target, and vectors should be rejected. A vector can be compared with near by vectors if the vector does not agree with the other it is rejected.

Using a hierarchical search technique: a large target window generates a first guess to guide the search and then the window size is reduced (Hubert, 1979).

The vectors may also be compared with a set of vectors from a prior wind fields. The prior wind field usually obtained from the most recent analyses of the appropriate standard levels or with satellite winds derived in the previous operation (Bristor 1975).

The final cloud wind vectors will be inspected by an editor who will have a number of facilities to survey the synoptic situation of the wind field. The editor can use some image analysis tool to enhance and superimposes wind field onto the image, also wind field can be superimposed to certain atmospheric layers over conventional analysis (e.g. 850mb, 500mb and 200mb) which give indication of location of cyclones and frontal zones. With this facility the removal of inconsistent erroneous vectors is very easy.

Severe selection criterion can indeed eliminate many erroneous vectors but also eliminate many erroneous vectors but also eliminate valid vectors near disturbance. Thus a balance must be drawn between accepting or rejecting a vector, usually vectors in new or rapidly changing circulation system are most important, and tracking in these areas also prove to be most troublesome (Hubert 1979).

In general, manual tracking is still superior to automated tracking since in complex situations a trained meteorologist can integrate his knowledge and is able to select and track cloud tracers which are either too difficult to pick up or obscured by surrounding clouds (Hubert 1979).

6.2 Details of Automated Cloud Wind Determination

Further details of cloud motion wind determination are discussed in this section, such as implementation of different image matching techniques, image rectification, image registration, wind vector selection and computation of wind speed and direction.

6.2.1 Image Registration

METEOSAT images have a spatial resolution of 5km at subsatellite point, for cloud tracking using a sequence of images 30 minutes apart each one misaligned pixel correspond to a speed error of $2.8 ms^{-1}$, this error increases with the latitude due to larger zenith angle. Smith and Phillips (1972) point out the importance of

picture alignment for accurate cloud motion measurement. Hall et al. (1972) described the use of landmarks or ground control points for picture registration and rectification.

Two images of the same region are said to be registered when equivalent geographic points of the scenes in the two image coincide. For accurate wind vectors calculation, images must be registered prior to image matching. Image registration is necessary because geostationary satellite is not perfectly still relative to the earth when the earth is being scanned. The fluctuation of the satellite generates geometrical distortion in the image and this must be compensated by the data processing station. Currently METEOSAT images are registered and rectified by ESOC in real time before dissemination. The rms error of misalignment is less than 0.4 pixel (Bos. et al. 1990), which correspond to an error of $\pm 0.5 ms^{-1}$ and is acceptable for cloud motion tracking.

Registration accuracy is especially critical for cases of low wind speeds when the image is separated by 30 minutes or one hour. In order to make useful measurements of motion, the displacement must be greater than all errors of its measurements (Hubert 1979). This registration requirement is particularly important in areas near the horizon where resolution is degraded and errors due to distortion and registration are more serious. The sources of error in cloud tracking depends on the combination of image resolution and errors of registration and measurement. On the other, hand manual tracking of cloud can utilize long sequences and this makes the registration accuracy less critical.

6.2.2 Image Matching Methods

Image matching can be roughly classified into two categories, they are correlation and feature matching (Aggarwal et al. 1981). Feature matching algorithms do not utilize intensity of the image but attempt to work with algorithms to locate boundaries or edges between regions. Edge or boundary information is extended to determine the position at which boundaries or edges intercept. The position of this vertex point and the direction and number of line segments emanating

from the vertex point form the basis of map comparison with the metric being some form of a mean square distance measure between locations of vertices in the reference and sensed image. Typical example are given by Goshtashy et al. (1986), Stockman et al. (1982).

The objective of this study is the extraction of mesoscale wind vectors, and distinct features are difficult to be extracted from cloud images only correlation type techniques will be considered. It is noted that the clustering approach in this study extracts the features of each cloud class, and so tracking using a clustered image is actually a combination of feature and correlation matching. One advantage of these techniques is they are non-sensitive to local noise, because they compute the average error between two patterns. Correlation techniques are basically a measurement of similarity between two image patterns. This is usually done in a pixel by pixel comparison of two images of the same object field obtained from different sensors, or of two images of an object field taken from the same sensor at different times.

In general, the image pattern in the sensed image to be matched can have transitional shifts, scale difference, and rotation shifts, as well as geometrical and intensity distortions. Because of computational efficiency, whenever possible, only transitional shifts is assumed. In this study scale differences, rotation shifts and intensity distortions are assumed negligible due to the short time difference between image, and so only transitional shift is measured. Arking et al. (1978) use cross-correlation to measure cloud displacement in a sequence of images and found that good results can only be obtained when the objects being tracked do not change their shape, size and orientation to more than a limited degree. The cross-correlation is less effective when a mixture of motions exists, unless one of the motions is strongly dominant.

This conclusion again supports the methodology in this study where cloud mixtures are separated using clustering before they are tracked.

Arking et al. also studied the use of Fourier phase difference method but they find cross correlation performs better for cloud images.

Image Matching Using Mean Absolute Difference

One of the simplest matching techniques is called 'Sequential Similarity Detection Algorithm' (SSDA) (Baruea and Silverman, 1972). Essentially, an array of data (the target window) is selected from an image and correlated element by element with selected pixels (the search window) of a second image. This techniques does not require normalization as in cross-correlation (to be discussed later). It is defined as the mean absolute difference (eq. 6.5) of the target and search window at every lag position (see Fig. 6.3)

$$S(u,v) = \frac{1}{JK} \sum_{j} \sum_{k} |g_t(j,k) - g_s(j-u, k-v)| \tag{6.5}$$

The best match is determined by the lag position with minimum error. The computation method can be further reduced by rejecting match position by accumulating the sum of difference which exceed a threshold. Generally the sum of difference increase rapidly on mismatch position and only slowly on possible match position so computation saving can be realized by only examine possible position in the highest precision. This technique was applied to cloud motion tracking by Wilson (1984).

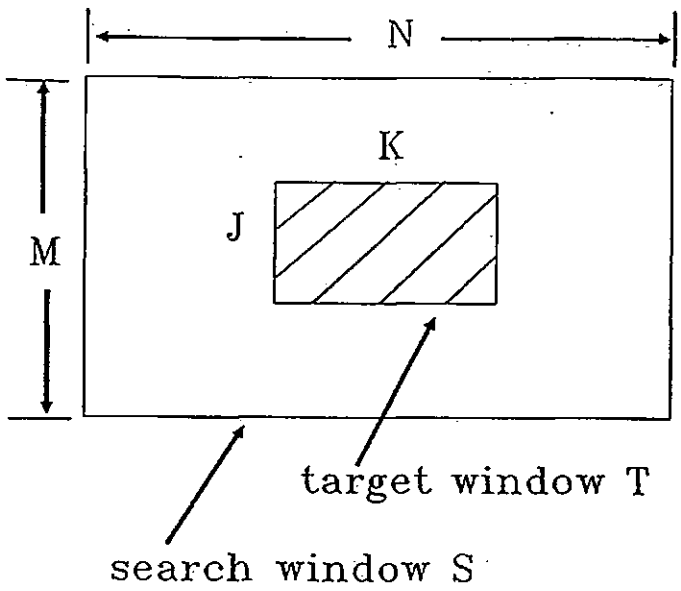


Figure 6.3: Definition of target and search window.

Image Matching Using Cross Correlation

Cloud tracking using cross-correlation was suggested by Leese et al. (1971). The displacement is determined by the lag position which produces the maximum correlation. The normalised cross-correlation function is defined as

$$R(u,v) = \frac{\sum_j \sum_k g_t(j,k) g_s(j-u,k-v)}{\sqrt{\sum_j \sum_k g_t(j,k) \sum_j \sum_k g_s(j-u,k-v)}} \tag{6.6}$$

where g_t is the target window and g_s is the search window and $R(u,v) \leq 1$. The lag position is given by $N-K+1, M-J+1$, in the horizontal and vertical directions, respectively. Computation of the numerator in eqn. 6.6 can be reduced by using the Fourier transform,

$$R(u,v) = \frac{\mathcal{F}^{-1}[G_t^*(x,y) G_s(x,y)]}{\sqrt{\sum_j \sum_k g_t(j,k) \sum_j \sum_k g_s(j-u,k-v)}} \tag{6.7}$$

where $G_t^*(x,y)$ is the complex conjugate of $G_t(x,y)$ while $G_s(x,y)$ is the Fourier transform of $g_s(j,k), G_t^* G_s$ is called the cross spectral density of g_t and g_s. How-

ever, for a small search window size (typically $\leq$ 64 x 64 pixel) the computational saving is little. There are two basic problems with this simple correlation measure. First, the correlation function may be rather broad (no distinct maximum), making detection of the peak difficult. This will happen if the search and target images contain large uniform regions with very few details. Secondly, systematic errors such as differences in scale size, geometric distortion, rotation, and intensity distortion between two images will make matching difficult and effectively suppresses the true peak.

Improved Cross Correlation

The statistical form of cross-correlation produces a more distinct peak than eqn. 6.6 because the image is subtracted by its mean and effectively increases the dynamic range of pixel's grey level.

The statistical cross-correlation function is defined as

$$R(u,v) = \frac{\mathrm{cov}(u,v)}{\sigma_t \sigma_s(u,v)} \tag{6.8}$$

where σ_t is the standard deviation of the target window, σ_s is the standard deviation of the search window at lag position u, v and $\mathrm{cov}(u,v)$ is the covariance between the target window and the search window at lag position u, v. Specifically,

$$\bar{G} = \frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K} g(j,k) \tag{6.9}$$

$$\sigma_t = \left\{\frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K}(g_t(j,k) - \bar{G}_t)^2\right\}^{\frac{1}{2}} \tag{6.10}$$

$$\mathrm{cov}(u,v) = \frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K}[g_t(j,k) - \bar{G}_t][g_s(j-u,k-v) - \bar{G}_s] \tag{6.11}$$

where $\bar{G}$ is the mean grey level of a window.

Another technique to improve cross-correlation is to replace the target and search image with their gradient (edge enhanced) images. The gradient operation

is equivalent to preprocess both image via a two dimensional convolution (Svedlow et al. 1978, Pratt 1973, Arcese et al. 1970).

Speed up the Matching Process

Beside computation of cross-correlation via the Fast Fourier transform, the search of correlation peak can be speeded up by hill-climbing techniques (Jain and Jain 1981), coarse-fine search (Rosenfeld and Vanderbrug 1977), hierarchical search (Wong and Hall 1978) and Sequential Similarity Detection Algorithm (Barnea and Sliverman 1972). These techniques can be roughly divided into sequential and hierarchical search methods.

The brute force methods for computation of cross correlation require the examination of possible lag position in the highest resolution and is a very expensive process.

The sequential techniques accumulate the sum of error between the target image and each position on the search image. Since mismatches of error usually grow faster than matches, the poorly matched area can be detected quickly and rejected at an early stage of the operation. In this way the total cost of searching a match is reduced considerably.

In the hierarchical approaches, resolution of images and template are reduced by averaging or pyramids (reduced sample rate). The matching is performed first over the reduced resolution images, and if a promising area is detected for a given threshold, the matching is performed at higher resolution within this region.

Most of these methods require a threshold to reject a lag position as mismatch. A threshold is usually depend on the reference image. For applications required to perform a large number of matches the determination of a threshold is time consuming and may generate excessive false alarms if the threshold is not set correctly.

A fast matching technique which does not require threshold for cloud motion tracking is desirable. One of this methods searches the correction peak using the steepest descent method. A 2-dimensional search method following this approach

was proposed by Jain and Jain (1981) for image displacement measures. The search is accomplished by successively reducing the area of search. Each step consists of searching five lag positions (Figure 6.4).

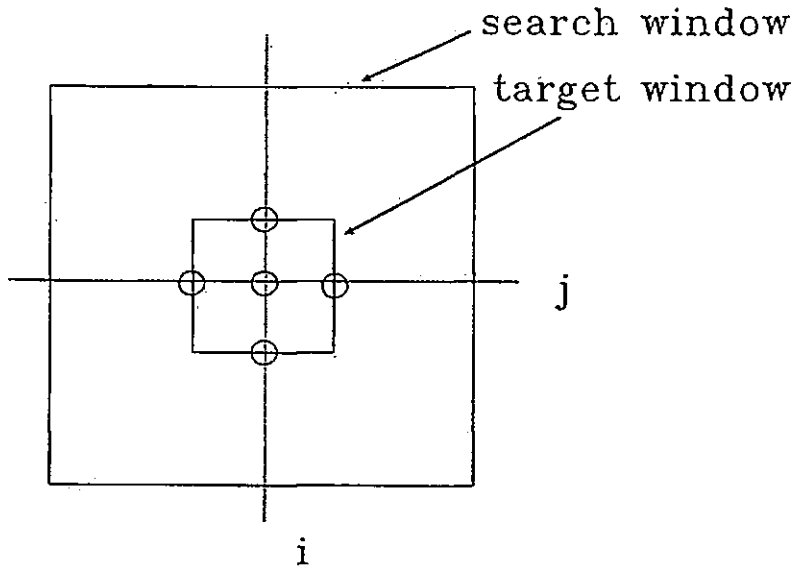


Figure 6.4: Illustration of the five locations which will be computed at the beginning of the hill climbing algorithm. The area contained by the five locations contract after each step until the area reduces to a 3 x 3 pixel size. In the final step all the nine locations are searched and the location corresponding to the maximum or minimum is the match position.

The 2d-logarithm search algorithm is as follow:

For any integer $m > 0$, we define,

$A(m) = \{(i,j) | -m \leq i,j \leq m\}$

$\{(0,0),(m,0),(0,m),(-m,0),(0,-m)\}$

Step 1. $n' = \text{int}(\log_2 p)$ where $p = \max\{M-K+1, N-J+1\}$

$n = \max\{2, 2^{n'-1}\}$

$q = l = 0$ (start from the centre of search image)

Step 2. Set $B'(n) = B(n)$

Step 3. find $(i,j) \in B'(n)$ such that cross correlation $R(i+q, j+l)$ is maximum. If $i = 0$ and $j = 0$, goto Step 5;

Otherwise goto Step 4.

Step 4. $q = q + i, l = l + j$;

$B'(n) = B'(n) - (-i,-j)$ (shift target window to new starting point)

goto Step 3.

Step 5. $n = n/2$. If $n = 1$, goto Step 6,

otherwise, goto Step 2.

Step 6. Find $(i,j) \in A(1)$ such that $R(i+q, j+l)$ is maximum

$q = q + i, l = l + j$

(q,l) is the match position.

It is noted that successful match depends on the smoothness of the correlation surface $R(u,v)$, and if many peaks exist around the true peak then very often a wrong match is obtained. This is because all hill climbing algorithms only guarantee convergence to a local optimum. However, the 2d-log search is very efficient for example with 121 lag positions only 13 to 21 locations need to be searched to obtain the match position. The 2d-log search is applicable to other matching methods which optimize some function, such as the mean absolute error method used in this study.

6.2.3 Strategies for Cloud Motion Vector Selection

The size of the target window and the spacing between them determines the resolution of the computed wind field. Since a image matching function provides no information which feature in the search window produced the best match. Thus the extracted wind is assigned to the location in the centre of the area. The use of a smaller window would increase the resolution, but would degrade the computational stability because of less distinct feature within the target window. Therefore, there exists an optimal window size for image from a particular satellite. It is found in this study that the METEOSAT data has an optimal window size of 24 x 24 pixel, whilst Lunnon and Lowe (1990) with the Meteorological Office found an optimal wind size of 16 x16 pixel using METEOSAT images.

The maximum wind speed that can be computed is limited by the number of lag positions in any direction. The ESOC use a 96 x 96 pixel search window, and a 32 x 32 pixel target window therefore the maximum shift in either horizontal or vertical direction is 32 pixels and this corresponding to a distance of 160km at ssp. If the two images are taken 30 minutes apart, the maximum speed is therefore 173 knots. The correlation function usually contains more than one maximum and the maximum may not represent the true displacement.

In other cases, all peak values may be very close. This may due to target window containing a large uniform overcast with no prominent features so that many lag positions produce high correlation. Figures 6.5— 6.6 are examples of matching surfaces generated using cross-correlation and absolute mean error methods.

Due to the uncertainty normally encountered in interpreting correlation result, the matching is guided by weather forecast. A first guess is obtained from the numerical model closest to the time the images were taken, then the matching is only done in a reduced search area suggested by the first guess (Bristor 1975, see Fig. 6.7).

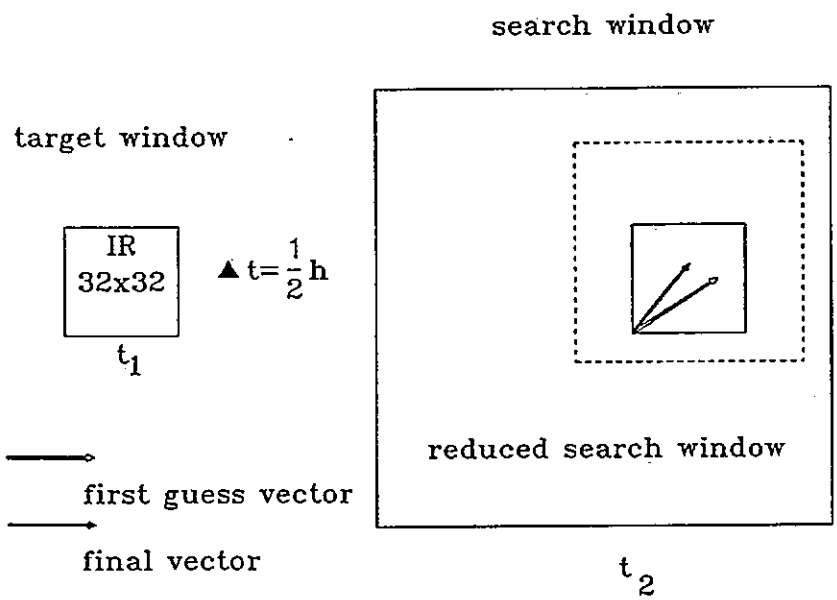


Figure 6.7: Image matching strategy adopted by NESS.

ESOC uses three infrared images to compute wind vectors (see Fig. 6.8) and this strategy is used in this study. The centre image is the reference image, and matching is done on the two adjacent images. A steepest ascent search strategy similar to 2d-log search is used to speed up the computation. Only targets which can be tracked in both adjacent image will be used to compute wind vectors. This implies that cloud targets have to have a lifetimes of at least one hour and is more likely a suitable tracer. Cloud targets which produce vectors which are not symmetrical within certain thresholds are rejected.

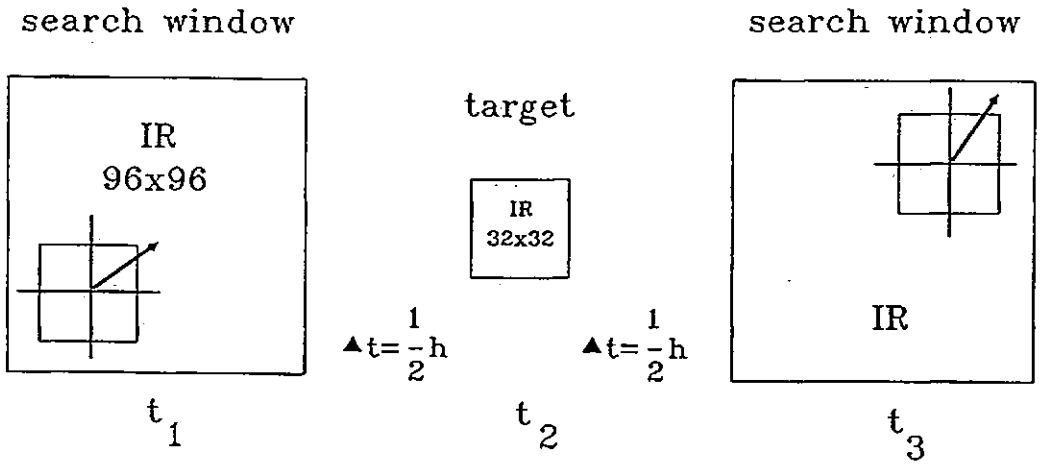


Figure 6.8: The image matching strategy adopted by ESOC.

The Japanese system uses a similar strategy to ESOC, but they use a coarse-fine search to reduce computation. Visible as well as infrared images are used for tracking (Hubert 1979).

6.2.4 Spatial Resolution

METEOSAT image usually has a resolution of 5km at the subsatellite point. Due to curvature of earth, the resolution gradually decrease with increase in latitude. For this reason the routine METEOSAT wind product is only derived within 55^0 latitude and longitude from subsatellite point at ESOC. To compute cloud motion at higher latitudes it is necessary to compute the actual displacement, which is usually done by geometric correction of the sensed image.

The instantaneous field of view (IFOV) (Kashef et al.1982) is the measure of the ground cell size from which energy is reflected and emitted before passing through the radiometers optical system. The IFOV of each detector (VIS, IR, WV) can be calculated with the aid of Figure 6.9.

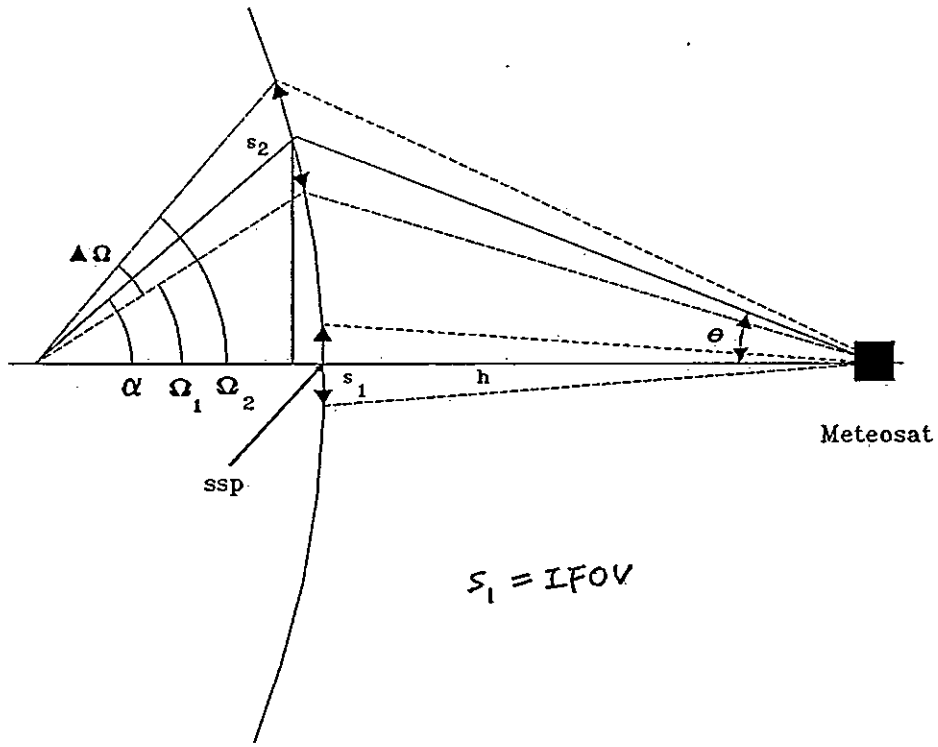


Figure 6.9: The variation of spatial resolution with latitude.

The spatial resolution is given by $R\Delta\Omega$ where R is the earth radius and $\Delta\Omega$ is the angle subtended by the ground cell in radian.

$$\Omega_1 = \sin^{-1}\left[\left(\frac{R+h}{R}\right)\sin\left(\theta - \frac{\text{IFOV}}{2}\right)\right] - \left(\theta - \frac{\text{IFOV}}{2}\right) \quad (6.12)$$

$$\Omega_2 = \sin^{-1}\left[\left(\frac{R+h}{R}\right)\sin\left(\theta + \frac{\text{IFOV}}{2}\right)\right] - \left(\theta + \frac{\text{IFOV}}{2}\right) \quad (6.13)$$

$$\begin{aligned}\Delta\Omega &= \Omega_2 - \Omega_1 \\ &= \sin^{-1}\left[\left(1+\frac{h}{R}\right)\sin\left(\theta + \frac{\text{IFOV}}{2}\right)\right] \\ &\quad - \sin^{-1}\left[\left(1+\frac{h}{R}\right)\sin\left(\theta - \frac{\text{IFOV}}{2}\right)\right] - \text{IFOV}\end{aligned} \quad (6.14)$$

$$R = 6378 \text{ km}$$

$$h = 35900 \text{ km}$$

$$\text{IFOV} = 7.1819 \times 10^{-3} \frac{rad}{\Lambda} \text{(IR,WV, and half resolution VIS)}$$

The vertical spatial resolution of METEOSAT at 60^0 latitude is therefore nearly double that at subsatellite point and at southern England it is approximately 10 x 5km in the north-south and east-west direction respectively. Therefore image matching results have to be corrected to the true distance.

6.2.5 Image Rectification

Once the displacement is measured using some image matching techniques. The velocity is easily computed by converting the displacement to distance and divided by the time difference between two images.

It is noted that all METEOSAT images have geometric distortion, this distortion increases as the IFOV move away form the subsatellite point. In order to compute the true displacement the images must be rectified before image matching. This approach is computational expensive, in this study the image matching is done on raw image and the resulting initial and final position is mapped to the actual longitude and latitude on the earth surface. In this way only two coordinates need to be computed.

Rectification is the process by which the geometry of an image area is made planimetric. In the case of METEOSAT the image is a two dimensional projection of the three dimensional earth surface, the problem of finding the mapping between a pixel's coordinates, and the earth location is often referred as image to map rectification. This is not to be confused with image-image registration where the two images may have similar distortion and one of them is treated as reference.

Generally only two types of distortion can be removed by rectification, they are radiometric distortion and geometric distortion (Kashef and Sawchuk 1982). Radiometric distortion is caused by atmospheric and sensor induced filtering, sensor imperfections, scanner non-uniform responses, detector gain variations and sensor detection gain errors. Geometric distortion is caused by:

1. Sensor related: Variations in the motion of the sensor over successive passes introduce distortions. For example, irregular angular velocity of the spin scan operation.
2. Alignment Variations: The variation in alignment of the sensor with respect to the spacecraft coordinates axis.
3. Attitude Variations: Variations in the spacecraft attitude (yaw, roll, and pitch) with respect to a previous pass will cause registration error.
4. Ephemeral Variations: The results form variations in the location of the platform with respect to the ground with successive passes over a given region.

There are mainly two methods to rectify images with geometric distortion. The first uses direct modelling, the time-dependent deviations of the satellite from nominal position, attitude and speed are described by an image geometry model. The geometry model provides a deformation vector field which relates the ideal reference image to the actual image. This method is used by ESOC to rectify METEOSAT image operationally (Bos et al. 1990). The second is an empirical method which requires no explicit knowledge of the distortion effects, but uses ground control points (GCPs) whose position in both image and map are utilised to derive the mapping or transformation equations.

GCPs are distinct features in the image for which the latitude and longitude values are known. As the model method requires many parameters, the empirical method is used. The first step of rectification is to establish a distortion model. Let us define the coordinates system of image and map as in Figure 6.10

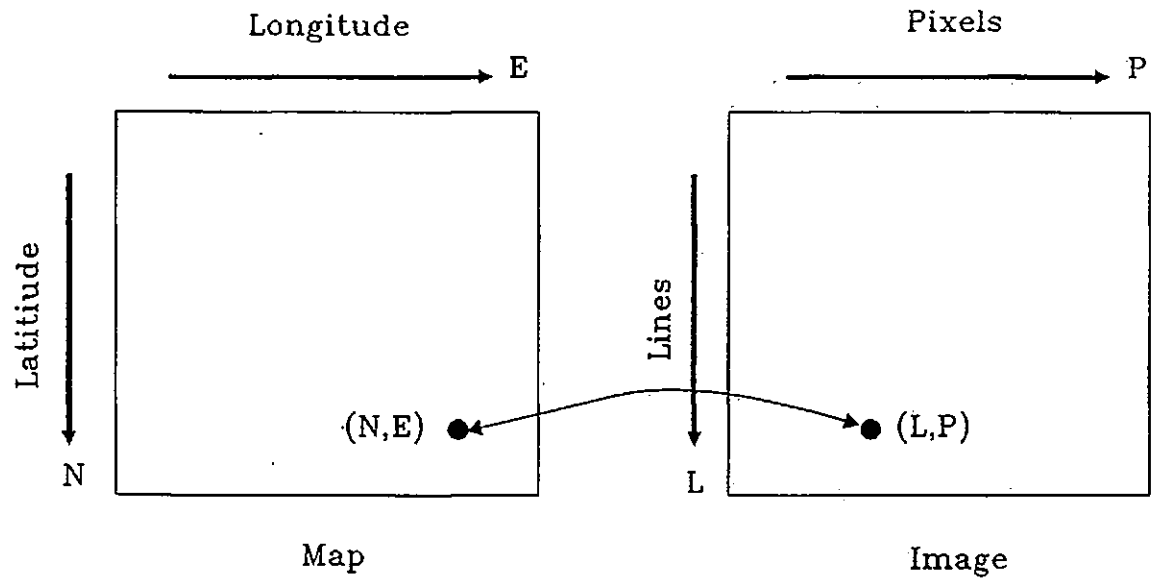


Figure 6.10: Coordinate system for image rectification.

Suppose the distortion relationship between image and map is given by distortion functions

$$E = \alpha(P, L) \tag{6.15}$$

$$N = \beta(P, L) \tag{6.16}$$

Let the map be a function $f(E, N)$, it can be written in term of E, N:

$$f(E, N) = f(\alpha(P, L), \beta(P, L)) = g(P, L) \tag{6.17}$$

The distortions given by eqn. 6.15 and 6.16 may be non-linear functions of (P, L) but must be a one to one mapping of points form one coordinate system to another. Since we are trying to obtain an estimate of the map from the image, the system may be inverted, the inverted distortion functions are:

$$P = \phi(E, N) \tag{6.18}$$

$$L = \psi(E, N) \tag{6.19}$$

found and used to generate the rectified image $\hat{f}(E,N)$ using intensity interpolation.

To solve the distortion model of eqn. 6.15 and 6.16, pairs of ground control points in both the map and image must be found. These points are usually small islands or distinctive features on the coastline, such as headlands.

The empirical method is to approximate the distortion function by polynomials of degree n having the form

$$E = \alpha(P,L) = \sum_{i=0}^{m}\sum_{j=0}^{m-i} a_{ij}P^iL^j \tag{6.21}$$

$$N = \beta(P,L) = \sum_{i=0}^{m}\sum_{j=0}^{m-i} b_{ij}P^iL^j \tag{6.22}$$

The value of degree m usually depends on the accuracy required, if the distortion area is large and severe, then m equal to 2 or 3 should be used.

Third order polynomials are used in this study because of the large area to be converted. The area coverage is (moving form the top right hand corner in a clockwise direction) $61^0N5.5^0E, 40^0N3.4^0E, 40^0N10.7^0W, 61^0N17.1^0W$. Using third order polynomials for $\hat{E}$ and $\hat{N}$ we have

$$\begin{aligned}\hat{E} &= a_0 + a_1P + a_2L + a_3P^2 + a_4PL + a_5L^2 \\ &\quad + a_6P^3 + a_7P^2L + a_8PL^2 + a_9L^3 \end{aligned} \tag{6.23}$$

$$\begin{aligned}\hat{N} &= b_0 + b_1P + b_2L + b_3P^2 + b_4PL + b_5L^2 \\ &\quad + b_6P^3 + b_7P^2L + b_8PL^2 + b_9L^3 \end{aligned} \tag{6.24}$$

where P and L are the row(pixel) and column(line) of a pixel coordinate respectively and $\hat{E}$ and $\hat{N}$ are estimates of the Northing (latitude) and Easting (longitude) respectively for the corresponding pixel. Eqn. 6.23 and 6.24 are of the same form and may be generalised:

$$\hat{\gamma} = c_0 + c_1P + c_2L + c_3P^2 + c_4PL + c_5L^2 + c_6P^3 + c_7P^2L + c_8PL^2 + c_9L^3 \tag{6.25}$$

The solving of $\alpha(P, L)$ is the same as for $\beta(P, L)$ but each has it own set of coefficients. $a_0, \ldots, a_9$ and $b_0, \ldots, b_9$ respectively. To solve eqn. 6.25 only ten equations are required which implies ten GCPs is enough and the system of equations is said to be exactly determined. However, the solution of these exact systems is often ill-conditioned (numerically unstable) and it is usually solved by using more than ten GCPs. One standard method for solving this over-determined system (no exact solution) of equations is least-squares analysis.

First the non-linear equation 6.25 has to be transformed into a linear equation by change of variables. Let

$$\begin{aligned} x_0 &= 1 \\ x_1 &= P \\ x_2 &= L \\ x_3 &= P^2 \\ x_4 &= PL \\ x_5 &= L^2 \\ x_6 &= P^3 \\ x_7 &= P^2L \\ x_8 &= PL^2 \\ x_9 &= L^3 \end{aligned} \tag{6.26}$$

Substitute $x_0, \ldots, x_9$ into eqn. 6.25 gives

$$\begin{aligned} \hat{\gamma} &= c_0x_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \\ &\quad + c_5x_5 + c_6x_6 + c_7x_7 + c_8x_8 + c_9x_9 \end{aligned} \tag{6.27}$$

Least square method is to fit a model to minimize the sum of square error between the model and the estimate (between E and $\hat{E}$, and N and $\hat{N}$ respectively). Two sets of equations must be determined: one for calculating $\hat{E}$ and the other for calculating $\hat{N}$ form P and L.

The objective function of the least square method is

$$e = \sum_{i=1}^{n} (\gamma_i - \hat{\gamma}_i)^2 \tag{6.28}$$

when n is the number of GCPs. Eqn. 6.28 may be expanded by substituting for $\hat{\gamma}_i$ from eqn. 6.27 and by introducing $x_{0i}, \ldots, x_{9i}$ for the ith GCP:

$$e = \sum_{i=1}^{n} (\gamma_i - c_0 x_{0i} - c_1 x_{1i} \ldots - c_9 x_{9i}) \tag{6.29}$$

The solution of this criterion can be found by minimizing the sum of square error e (see Appendix F).

6.2.6 Calculation of Distance

The distortion models gives the estimates of longitude and latitude of a pair of pixel coordinates in the image. If the starting point is (P_0, L_0) and the ending point is (P_1, L_1) the corresponding longitude and latitude are (E_0, N_0) and (E_1, N_1) respectively. The geometry of Earth's surface is calculated with oblique spherical trigonometry (Ayres 1954), which approximates the Earth's surface by a sphere. The distance of cloud displacement can be measured assuming that the cloud's path is along a great circle. The course (direction) is measured clockwise from North. The situation is represented by the spherical triangle in Figure 6.12.

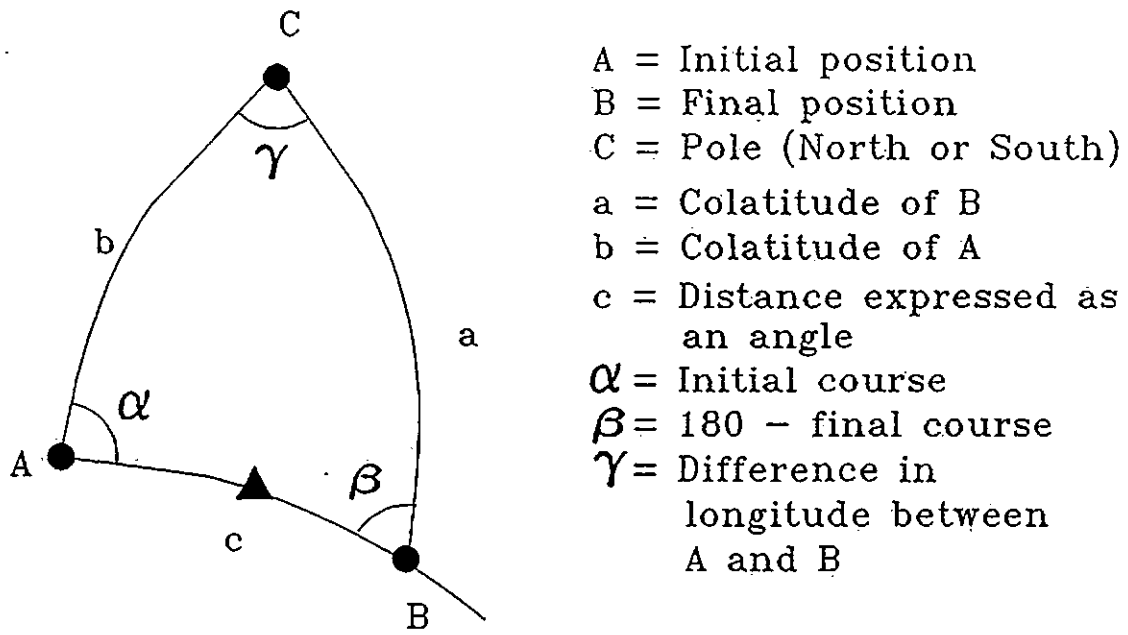


Figure 6.12: Spherical triangle for calculating the distance between A and B.

Note that a, b and c are arclengths expressed as angles subtended at the Earth's centre. The latitude and longitude of A and B are used to find a, b and γ as follows;

$a = 90^0 -$ (latitude of B)

$b = 90^0 -$ (latitude of A)

$\gamma =$ |(longitude of A)-(longitude of B)|

The great circle distance C is given by cosine law

$$\cos C = \cos a \cos b + \sin a \sin b \cos \gamma \tag{6.30}$$

The wind direction will be defined as the initial course α, since the initial course is very close to the final course due to short distance travelled, using sine law

$$\sin \alpha = \frac{\sin a \sin \gamma}{\sin C} \tag{6.31}$$

If the final position is East of the initial one, the initial course is α. Otherwise the initial course is $360^0 - \alpha$. The length of arc C is converted to minute, since

one minute of arc ($1/60^0$) at the Earth's surface is a distance of one nautical mile, the arc length C is $60C$ nautical miles. One nautical mile is equivalent to 1.15078 statute mile or 1.852km.

The distance d is then used to compute the average speed of the cloud which is then the estimate of wind velocity around the tracer. The average wind speed is

$$v = \frac{\text{distance travelled by the cloud tracer}}{\text{time between images used to track the cloud tracer}} \tag{6.32}$$

For METEOSAT

$$v = \frac{d}{0.5} \text{ knots/hr.} \tag{6.33}$$

N.B. 1 knot = 0.5148 m/s

6.3 An Automated Cloud Motion Determination Scheme

An automated scheme was developed to compare the "separate then track" approach and the traditional "track without discrimination" approach. Figure 6.13 shows the automatic scheme used in this study.

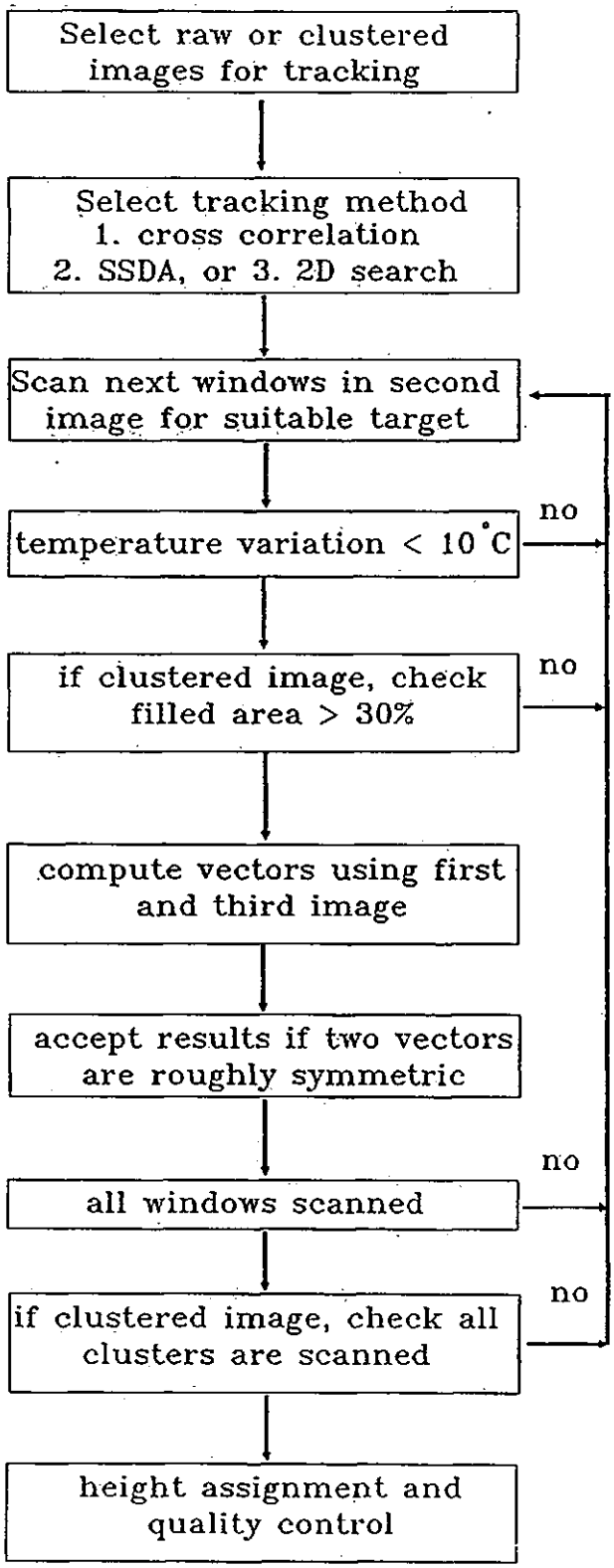


Figure 6.13: The automatic cloud motion wind scheme used in this study.

The Global-Local clustering algorithm presented in Chapter 4 is used to separated cloud classes, then tracking is done using one cloud class at a time. Tracking is also done using raw image such that a target window may contain clouds from more than one height levels. Results show that the "separate then track" approach improves tracking ability significantly. The improvement is particularly obvious in frontal areas where cloud mixtures are moving in different direction and these are most valuable to modelling of weather systems.

The scheme is developed based on the elements discussed in previous sections. Some simplification has to be made since the main objective of the scheme is to test the performance of the "separate then track" approach and it is not intended for operational use. The height assignment is the most difficult part to be carried out among all others, since the correct cloud top height requires accurate cloud top temperature (radiance) estimation. This process is very complicated and requires a database of 110 atmospheric models (Bowen and Saunders 1984). We have taken a simplified approach namely the "Level of Best Fit" (Hubert and Whitney 1971) which assigns vectors to height level which produce the minimum velocity difference. The LBF method has been compared with cloud top and cloud base assignment method by Lee (1979), the LBF method has found to give unrealistic results in some cases, nevertheless LBF is a useful for statistical analysis (Hubert 1979).

Different combination of target size and image resolution will give slightly different cloud patterns, therefore using different satellite data to track the same target should produce difference error. Another objective of the experiment is to investigate an optimum target size for METEOSAT images for cloud tracking. Three tracking methods namely; 1) normalised cross-correlation, 2) absolute mean difference (SSDA) and 3) 2d-log search with absolute mean difference. Target window sizes are 4 x 4, 8 x 8, 16 x 16, 24 x 24, and 32 x 32.

The scheme uses either raw or clustered images, and any one of the three target tracking techniques so giving six experimental approaches.

The basic approach is to use sequences of three (visible and infrared) images

spanning a total of one hour. The second infrared image is divided into non-overlapped areas which define possible target windows, and these are then checked for suitable cloud tracers. If the window has an infrared variance < 100 (i.e. black body scene temperature variations are usually $< 10^0 C$), then it is tracked in the first and third image of the sequence.

When using the clustered approach, only the middle pair of images are clustered, and only pixels $g_t(j,k) \in \omega_i$ will be kept in the target window when using cluster ω_i for tracking. This results in a target window which is not necessarily filled with pixels, and so a further check must be made. At present, a window less than 30% filled is rejected for a target (the threshold is not critical but if it is too small spurious winds may be generated). Alternatively, tracking can be done using only the cluster which has the most number of pixel in a window. Clearly, target windows containing more than one cluster may have more than one wind vector, and results show that only a small number of target window generate more than one vectors (see Table 6.1— 6.12).

Since no prior information of wind direction is given, targets windows are located in the centre of search windows, and 15 to 25 lag positions are allowed in the horizontal and vertical direction respectively (dependents on the maximum wind speed found in the reference wind field from the Meteorological Office). This provides for a maximum wind speed of at least 75.5 knots (15 lag position) in the southern most of the clustering window. Since the spatial resolution decreases with increase of latitude, the maximum speed is larger than 75.5 knots in area of higher latitude. Two vectors are obtained by tracking cloud tracers in the first and third images in the sequence. Vectors corresponding to minimum (SSDA) or maximum (cross correlation) falling on the borders of the search area are rejected. Vectors are then checked for symmetry; if the speed difference is $> 50\%$ of the smaller vector, or the direction differs by $> 30^0$, no wind vector is generated for that window.

6.4 Cloud Motion Wind Results

The six sets of images described in chapter 4 are used to test the cloud motion scheme. Each of these images include either a warm or cold front and is suitable for cloud motion tracking.

Cloud motion tracking using raw and clustered infrared images has been tried. All tracking was done on images without geometric rectification, and the displacements were then corrected using a least square error rectification model (section 6.2.5). The outputs of the cloud tracking scheme were compared with the Meteorological Office numerical weather model results, and only wind vectors which are close to the predicted results were selected. The criteria to select a 'valid' vector are: if it is within a speed deviation of less than 50% and a direction deviation of less than 30^0. The wind vectors are assigned to a height level using the "Level of Best Fit" method , i.e. to the level with minimum speed deviation. The middle image is divided into an array of target windows and so higher wind density can be obtained by overlapping the target windows.

Tables 6.1— 6.12 gives the detailed cloud motion wind results for the six sequences (L=low level (850mb), M=middle level (500mb), H=high level (250mb)). Figures 6.14— 6.19 give the numerical prediction results (provided by Met. Office) interpolated onto a 16 x 16 target size grid, showing the wind field at 850mb (red), 500mb (green), and 250mb (yellow) levels. These reference wind fields indicate that wind at different levels can have very different speed and direction.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	size	method
18	38	148	12	11	41	-0.63	10.07	4.80	4.61	-4.28	-3.28	2.86	13.27	10.19	4x4	x
36	56	151	22	42	85	0.97	4.18	4.80	2.36	1.36	-2.27	4.15	7.72	9.89	4x4	ssda
73	93	75	35	65	36	1.26	0.63	3.84	7.51	-2.50	-10.18	4.28	6.03	8.21	4x4	2ds
27	24	55	24	21	41	1.24	5.92	3.32	3.00	2.66	-1.95	2.83	10.63	7.39	8x8	x
30	41	71	27	38	57	0.43	2.63	4.00	4.39	-1.38	-5.33	2.84	7.44	8.68	8x8	ssda
29	32	39	20	29	28	0.43	1.37	3.84	6.51	-3.09	-7.17	4.13	6.61	7.96	8x8	2ds
9	29	26	8	27	26	1.14	3.15	3.41	-0.14	-4.41	-6.23	2.65	7.16	6.92	16x16	x
13	31	23	12	29	21	-0.39	2.18	1.92	-4.83	-5.28	-5.87	2.32	4.47	7.73	16x16	ssda
12	23	13	8	22	12	-0.51	1.62	0.38	-4.62	-3.02	-7.86	2.79	4.86	8.03	16x16	2ds
10	7	10	9	6	10	-0.79	4.61	3.22	1.02	-0.71	-2.92	3.63	6.95	6.13	24x24	x
7	8	9	7	8	9	-0.34	0.44	-0.02	6.69	2.50	-3.71	3.92	4.52	4.56	24x24	ssda
7	8	6	7	8	5	-0.34	0.14	-3.79	6.69	3.90	-0.68	3.92	4.78	5.70	24x24	2ds
6	3	3	5	3	3	0.55	3.20	-3.65	-2.54	4.28	0.82	7.65	3.82	5.32	32x32	x
5	2	4	5	2	4	0.55	5.57	-2.36	-2.54	3.33	-3.10	7.65	5.58	7.29	32x32	ssda
5	2	2	4	2	2	-0.21	5.57	-6.46	0.38	3.33	-7.36	6.57	5.58	10.04	32x32	2ds

Table 6.1: Wind vectors results by tracking raw images on 5th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Multi-vec.	Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	window	size	method
22	51	189	7	14	35	0.01	8.94	6.29	11.16	1.01	-0.38	2.21	12.07	11.37	0	4x4	x
44	78	165	25	44	82	0.35	5.16	5.57	8.35	2.61	-0.55	2.45	9.02	9.67	1	4x4	ssda
94	113	102	40	70	46	1.63	1.03	3.07	6.30	-4.51	-9.91	4.35	6.03	7.92	3	4x4	2ds
22	29	72	18	19	41	1.18	4.31	3.69	5.01	-0.01	-4.70	3.29	7.61	8.10	1	8x8	x
29	46	79	23	33	56	0.26	4.53	4.13	3.74	-0.99	-5.04	2.98	8.08	8.53	7	8x8	ssda
36	41	46	21	33	26	0.30	0.76	4.67	7.89	-3.39	-5.69	3.06	5.07	8.09	0	8x8	2ds
11	27	37	10	21	32	2.26	2.92	1.86	4.58	-1.48	-4.17	3.77	7.27	5.58	6	16x16	x
16	28	33	13	23	27	-1.10	2.57	2.91	-2.99	-1.05	-6.02	5.32	4.67	6.76	7	16x16	ssda
18	20	24	11	19	16	2.95	0.64	2.40	4.94	-0.54	-9.62	3.67	3.57	5.95	4	16x16	2ds
10	15	18	6	11	11	-1.13	2.57	2.35	5.59	0.73	-4.56	3.26	5.33	6.49	5	24x24	x
9	12	18	5	10	13	0.63	0.11	3.73	7.21	-0.08	-10.40	1.65	4.69	6.16	7	24x24	ssda
9	8	16	5	7	10	1.39	0.86	2.60	12.65	0.77	-9.78	1.81	4.84	6.93	2	24x24	2ds
8	7	13	6	6	10	1.80	4.38	-1.21	2.82	1.40	-4.98	3.39	6.90	7.16	5	32x32	x
8	7	14	7	6	9	0.50	4.42	-0.87	6.23	-0.89	-6.29	3.45	6.66	3.85	7	32x32	ssda
6	7	10	5	6	9	-1.81	4.42	-0.77	9.74	-0.89	-6.90	6.50	6.66	3.73	3	32x32	2ds

Table 6.2: Wind vectors results by tracking clustered images on 5th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	size	method
55	56	103	23	18	31	4.01	3.86	8.14	5.26	0.50	-1.24	6.98	6.31	12.86	4x4	x
51	87	101	26	39	59	2.45	3.21	10.20	-1.31	1.50	0.60	4.30	6.55	15.27	4x4	ssda
91	99	103	35	48	59	1.15	0.22	2.75	-0.59	0.33	0.89	4.43	4.67	6.97	4x4	2ds
34	31	41	21	22	32	1.44	3.10	4.18	-0.16	2.33	1.85	3.17	5.95	10.34	8x8	x
38	45	48	26	29	33	1.42	3.25	3.95	6.34	1.37	0.38	4.32	5.83	7.93	8x8	ssda
32	40	43	21	26	25	1.69	1.34	1.37	-1.56	1.69	0.64	3.35	5.78	5.35	8x8	2ds
9	19	15	8	12	13	-0.06	3.09	8.00	4.28	5.21	5.65	2.30	5.63	15.68	16x16	x
12	21	9	11	11	8	0.04	3.22	1.16	6.53	3.41	5.99	2.38	6.22	4.50	16x16	ssda
11	24	8	10	14	6	0.26	2.36	-0.68	1.46	3.99	9.07	2.24	5.65	2.00	16x16	2ds
6	4	3	4	2	3	-1.98	-3.83	-4.95	8.70	-6.85	-1.43	2.16	3.84	6.51	24x24	x
6	5	2	5	2	2	-1.93	-3.83	1.76	5.15	-6.85	-8.42	3.25	3.84	1.81	24x24	ssda
5	6	3	4	2	3	-2.45	-0.97	-1.75	-1.74	-4.46	-4.30	3.78	2.74	3.55	24x24	2ds
1	4	0	1	2	0	3.91	1.36	—	27.88	-2.52	—	3.91	2.75	—	32x32	x
3	3	0	3	1	0	5.01	1.58	—	12.60	10.61	—	6.36	1.58	—	32x32	ssda
3	3	0	3	1	0	5.01	1.58	—	12.60	10.61	—	6.36	1.58	—	32x32	2ds

Table 6.3: Wind vectors results by tracking raw images on 8th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Multi-vec.	Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	window	size	method
45	63	133	10	16	38	4.97	1.85	9.23	6.22	-1.57	1.44	7.74	6.89	14.00	1	4x4	x
66	88	123	25	38	50	2.42	2.64	9.45	2.72	3.69	-0.53	4.38	6.80	13.66	4	4x4	ssda
104	125	116	39	55	60	1.86	-0.58	3.59	-0.85	0.91	3.14	4.36	4.94	7.88	3	4x4	2ds
35	42	51	17	30	34	1.54	3.36	8.46	0.09	-2.81	-0.96	3.48	5.78	13.48	1	8x8	x
35	50	70	22	30	41	2.08	2.15	5.56	2.88	1.52	-0.77	3.71	6.40	9.93	6	8x8	ssda
38	44	54	26	22	37	0.48	2.14	-0.63	0.65	1.26	0.08	3.18	4.94	6.90	6	8x8	2ds
14	33	22	9	21	17	-0.15	1.45	6.16	1.90	2.30	1.01	3.93	4.83	14.23	3	16x16	x
16	32	18	11	20	14	0.23	2.38	0.98	6.27	4.07	-2.63	2.24	5.26	4.73	1	16x16	ssda
19	37	9	12	26	5	0.52	1.85	0.58	-3.15	1.96	-1.02	2.51	4.61	2.96	3	16x16	2ds
7	15	19	5	8	14	0.67	0.91	4.07	8.97	-0.89	-0.65	3.66	4.52	7.80	4	24x24	x
16	13	15	11	8	10	0.43	1.97	2.74	2.83	-0.80	4.00	4.01	4.89	5.27	4	24x24	ssda
11	13	8	8	6	6	-0.11	1.12	0.95	2.71	-9.90	-3.24	2.28	2.20	6.79	2	24x24	2ds
2	14	9	2	8	7	2.90	0.46	3.99	10.90	-1.51	5.94	3.07	3.38	10.43	1	32x32	x
5	13	8	4	5	7	1.38	1.06	-0.85	8.42	5.91	11.48	1.82	1.97	7.19	3	32x32	ssda
8	13	5	5	6	4	2.95	-1.47	-3.38	14.23	5.22	17.09	5.29	4.97	8.19	3	32x32	2ds

Table 6.4: Wind vectors results by tracking clustered images on 8th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	size	method
23	38	178	13	23	65	0.14	2.92	3.33	9.46	6.33	8.29	2.02	4.77	7.42	4x4	x
34	72	209	22	45	113	-2.70	1.81	3.74	6.24	4.34	4.88	5.46	4.49	7.08	4x4	ssda
87	78	123	40	47	59	-0.57	0.95	2.99	3.94	-0.27	5.44	4.00	3.29	5.70	4x4	2ds
24	29	89	15	24	64	0.24	3.22	3.27	8.72	5.76	8.72	3.52	6.32	6.61	8x8	x
31	29	100	25	20	74	-0.34	1.17	2.41	8.73	1.84	5.05	3.41	4.09	6.43	8x8	ssda
30	32	54	18	23	38	0.25	0.09	2.27	9.09	-0.54	3.84	2.91	2.85	6.84	8x8	2ds
15	12	19	10	10	16	-0.00	1.22	2.50	7.31	4.79	8.73	2.93	3.67	5.48	16x16	x
13	12	15	9	10	12	-0.08	2.20	1.68	-0.53	5.17	7.36	2.76	4.28	4.83	16x16	ssda
16	9	14	11	8	11	-0.95	1.38	1.40	2.78	3.00	7.85	2.60	4.57	4.21	16x16	2ds
3	7	7	2	6	6	-2.80	-1.25	1.23	11.06	2.32	7.88	3.85	4.45	5.68	24x24	x
7	7	8	5	6	7	-0.91	-1.10	-0.97	7.42	-2.11	4.84	2.75	2.73	3.39	24x24	ssda
6	6	9	4	5	8	-1.56	-1.96	-1.39	3.05	-3.83	2.36	3.08	2.62	3.52	24x24	2ds
3	1	3	3	0	3	-2.05	—	1.01	14.87	—	2.15	3.92	—	2.43	32x32	x
1	1	2	1	0	2	-4.95	—	0.91	12.30	—	1.81	4.95	—	3.14	32x32	ssda
1	1	3	1	0	3	-4.95	—	0.14	12.30	—	0.03	4.95	—	1.60	32x32	2ds

Table 6.5: Wind vectors results by tracking raw images on 11th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Multi-vec.	Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	window	size	method
15	46	235	7	17	56	-0.24	3.66	3.42	17.95	3.80	5.00	1.17	5.06	7.81	0	4x4	x
31	59	239	17	35	114	-2.61	1.37	3.34	4.44	6.42	3.42	5.62	3.88	7.37	3	4x4	ssda
93	97	144	36	50	54	-0.98	0.78	2.03	5.83	1.30	4.13	4.41	3.43	5.26	2	4x4	2ds
20	25	102	14	21	68	-0.25	1.96	3.68	10.43	6.09	9.01	4.11	4.85	7.20	2	8x8	x
21	41	110	14	25	78	-0.14	0.10	2.28	7.91	2.84	6.71	4.03	3.53	5.64	6	8x8	ssda
35	42	52	26	28	31	-0.69	0.17	1.64	6.97	1.31	4.09	3.19	3.35	6.10	3	8x8	2ds
14	19	49	10	12	36	-0.68	1.05	2.40	9.75	7.71	6.98	2.23	3.67	6.21	7	16x16	x
21	28	34	12	21	25	-1.29	1.96	2.62	11.93	2.28	5.85	2.93	5.56	6.27	9	16x16	ssda
20	22	24	11	14	19	-0.12	2.53	1.23	4.69	2.40	7.54	2.21	6.27	3.92	5	16x16	2ds
9	19	18	8	13	15	-1.44	2.21	3.35	19.73	7.44	6.04	2.86	3.49	5.83	4	24x24	x
8	21	17	8	13	15	-1.40	0.57	1.59	13.82	3.39	5.31	3.36	2.91	5.11	3	24x24	ssda
7	17	14	7	12	12	-1.91	-0.11	1.80	13.68	-0.96	3.86	3.60	2.62	5.36	3	24x24	2ds
4	10	13	3	7	12	-5.38	1.01	2.02	16.83	11.22	3.83	5.40	4.16	5.27	3	32x32	x
6	8	13	5	6	11	-1.37	0.74	1.07	14.95	3.22	3.61	5.04	3.68	4.55	3	32x32	ssda
9	8	14	7	7	12	-2.31	0.40	0.04	13.58	0.60	2.05	5.41	3.02	3.12	4	32x32	2ds

Table 6.6: Wind vectors results by tracking clustered images on 11th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	size	method
37	38	164	18	28	56	0.24	1.16	3.85	5.62	-4.24	4.17	3.81	4.29	6.58	4x4	x
79	74	165	50	54	85	-0.05	0.14	4.15	4.22	-3.84	1.64	4.01	3.92	7.73	4x4	ssda
102	96	155	38	58	69	0.19	0.37	1.90	1.44	-1.70	2.56	4.61	3.89	6.05	4x4	2ds
57	53	60	40	49	49	0.19	0.02	1.74	1.67	-6.60	4.71	3.17	3.55	4.99	8x8	x
71	70	64	57	62	47	0.95	-0.09	2.83	-0.03	-3.08	2.09	3.65	3.59	5.66	8x8	ssda
54	56	41	37	50	25	0.40	-0.51	1.57	2.17	-4.18	5.87	4.39	3.45	6.48	8x8	2ds
23	27	17	22	25	12	0.65	-0.97	0.50	0.63	-1.28	1.66	3.49	4.06	8.46	16x16	x
29	22	18	26	20	11	1.90	-0.47	1.87	1.31	-1.73	3.87	3.80	4.18	8.77	16x16	ssda
23	25	14	20	22	8	1.51	-0.43	0.87	3.15	-1.07	7.08	3.51	3.25	6.90	16x16	2ds
11	11	5	9	10	4	0.83	0.18	-0.13	2.01	4.43	1.45	4.06	3.93	3.40	24x24	x
14	12	5	12	11	4	0.71	0.30	1.89	1.62	1.10	2.71	3.28	3.08	5.03	24x24	ssda
14	12	3	11	11	2	0.83	0.30	4.71	2.75	0.34	12.64	3.42	3.00	7.17	24x24	2ds
8	4	2	8	4	2	1.41	-1.38	-4.22	-4.78	6.09	-5.39	3.83	3.30	4.54	32x32	x
11	1	1	10	1	1	0.98	-0.23	-8.47	-1.77	6.81	-4.33	4.06	0.23	8.47	32x32	ssda
10	1	0	9	1	0	0.21	-0.23	—	-0.37	6.81	—	3.37	0.23	—	32x32	2ds

Table 6.7: Wind vectors results by tracking raw images on 15th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Multi-vec.	Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	window	size	method
24	40	203	10	21	48	1.12	0.53	4.73	-1.13	-8.97	2.93	4.35	3.87	8.09	0	4x4	x
85	77	201	54	55	87	1.21	0.54	4.90	4.58	-3.66	0.70	4.10	4.48	7.85	0	4x4	ssda
126	114	165	52	61	65	0.37	0.42	2.81	3.06	-3.26	0.22	4.55	3.61	6.36	6	4x4	2ds
39	50	73	26	45	50	-0.25	-0.46	2.27	2.50	-5.86	4.29	3.42	4.21	5.85	6	8x8	x
60	82	95	46	67	59	0.80	0.43	3.43	1.23	-4.89	1.03	3.66	4.04	6.53	13	8x8	ssda
55	68	60	37	53	33	0.27	-0.01	2.06	6.38	-3.94	2.80	4.75	3.82	7.97	6	8x8	2ds
36	29	34	28	25	28	0.56	0.39	0.16	-0.46	-1.14	2.43	3.52	3.51	5.47	6	16x16	x
42	36	35	29	29	24	2.12	-0.45	0.78	-0.67	-5.23	7.17	3.24	3.28	5.65	18	16x16	ssda
35	29	27	24	23	17	2.20	0.59	0.23	1.12	-1.55	4.65	3.50	2.97	6.61	10	16x16	2ds
21	20	11	13	16	10	-0.09	0.46	1.78	8.22	-3.90	3.94	2.45	3.89	5.41	5	24x24	x
27	25	12	19	21	10	1.78	1.19	2.81	4.18	-3.31	-1.19	4.06	2.89	7.76	8	24x24	ssda
26	22	4	21	18	3	1.39	0.91	-0.27	3.79	-4.04	4.92	4.88	2.52	3.35	6	24x24	2ds
16	14	5	10	12	4	1.49	-0.56	2.41	-3.17	-4.61	0.19	3.56	2.32	3.02	7	32x32	x
16	11	9	10	10	7	1.84	-1.15	1.29	-2.26	-5.19	-2.76	4.97	2.11	9.56	7	32x32	ssda
17	10	5	10	10	3	0.85	-1.15	2.03	1.44	-5.19	1.44	4.08	2.11	2.76	6	32x32	2ds

Table 6.8: Wind vectors results by tracking clustered images on 15th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	size	method
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	4x4	x
22	27	145	6	11	59	-1.06	14.15	-10.14	2.22	11.88	-5.97	3.11	15.84	15.40	4x4	ssda
38	61	99	6	17	37	5.94	7.58	-9.04	17.85	12.28	5.90	9.45	10.79	16.24	4x4	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	8x8	x
25	24	63	8	18	52	1.89	15.71	-13.74	2.48	12.16	-5.96	6.61	17.01	16.24	8x8	ssda
17	10	19	3	5	10	11.79	7.76	-9.63	-2.79	7.21	0.83	13.27	11.74	12.05	8x8	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	16x16	x
9	13	19	2	8	19	-3.74	14.76	-17.75	1.80	12.28	-7.10	3.74	16.53	18.80	16x16	ssda
9	8	12	2	5	12	-4.92	2.59	-15.75	-6.73	4.84	-6.67	5.13	9.00	16.76	16x16	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	24x24	x
4	7	4	2	4	4	-0.46	14.34	-20.58	10.88	2.56	-6.15	1.47	15.30	21.19	24x24	ssda
3	3	2	2	2	2	-0.46	3.26	-17.73	10.88	-8.60	-6.88	1.47	5.14	18.06	24x24	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	32x32	x
0	2	1	0	2	1	—	13.36	-16.05	—	14.29	3.26	—	15.43	16.05	32x32	ssda
0	2	1	0	2	1	—	18.05	-16.05	—	22.84	3.26	—	18.30	16.05	32x32	2ds

Table 6.9: Wind vectors results by tracking raw images on 18th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Multi-vec.	Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	window	size	method
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	0	4x4	x
11	30	175	3	11	62	2.89	12.71	-8.24	-7.83	6.95	-3.87	3.30	15.66	15.93	0	4x4	ssda
35	63	109	7	16	39	4.76	8.58	-10.40	14.31	9.76	5.48	8.71	11.93	15.53	1	4x4	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	0	8x8	x
17	30	56	0	22	37	—	12.83	-13.46	—	11.14	-4.29	—	15.42	15.56	0	8x8	ssda
14	18	25	3	10	10	5.52	4.72	-10.06	11.71	8.61	-0.30	7.09	8.80	11.89	1	8x8	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	0	16x16	x
12	16	27	3	10	26	5.15	10.02	-16.53	5.33	10.48	-3.78	13.58	13.64	17.88	1	16x16	ssda
8	11	14	1	7	12	-3.48	11.01	-12.49	8.33	4.18	-6.58	3.48	14.77	14.52	1	16x16	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	0	24x24	x
1	7	5	0	5	4	—	12.59	-22.14	—	4.54	-6.93	—	15.23	23.18	0	24x24	ssda
2	5	3	0	3	2	—	14.07	-19.23	—	-3.10	-6.71	—	14.15	19.32	0	24x24	2ds
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	0	32x32	x
1	1	1	0	1	1	—	5.65	-14.76	—	7.11	-12.15	—	5.65	14.76	0	32x32	ssda
1	1	0	0	1	0	—	10.97	—	—	29.12	—	—	10.97	—	0	32x32	2ds

Table 6.10: Wind vectors results by tracking clustered images on 18th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	size	method
5	25	146	2	20	42	11.81	5.47	-0.74	7.49	0.24	-0.66	11.97	12.68	9.88	4x4	x
15	57	127	5	51	76	-3.69	5.82	2.80	7.28	1.27	-1.42	9.29	11.23	16.29	4x4	ssda
51	52	30	13	20	7	6.12	-2.95	-1.25	3.69	7.10	-0.17	7.67	9.26	5.20	4x4	2ds
3	22	43	2	22	33	4.81	9.34	-3.33	-3.90	3.73	-2.41	5.24	13.50	13.04	8x8	x
7	28	50	5	24	47	5.80	8.19	-0.71	7.41	4.30	-2.72	6.96	11.65	16.25	8x8	ssda
13	14	13	3	10	7	6.77	-0.28	5.64	-0.35	4.91	-4.44	10.60	7.88	13.25	8x8	2ds
0	14	8	0	14	8	—	10.68	-4.74	—	2.87	-1.33	—	13.15	12.43	16x16	x
2	15	7	1	15	7	8.79	6.56	-2.00	-19.04	2.36	-2.30	8.79	13.28	11.52	16x16	ssda
3	7	0	1	7	0	8.79	0.31	—	-19.04	0.36	—	8.79	13.66	—	16x16	2ds
0	7	0	0	6	0	—	8.50	—	—	1.56	—	—	11.70	—	24x24	x
0	6	0	0	5	0	—	9.74	—	—	-1.56	—	—	10.89	—	24x24	ssda
0	1	0	0	1	0	—	2.70	—	—	-1.84	—	—	2.70	—	24x24	2ds
0	2	0	0	2	0	—	6.00	—	—	10.63	—	—	6.56	—	32x32	x
0	2	0	0	2	0	—	4.77	—	—	10.20	—	—	6.15	—	32x32	ssda
0	1	0	0	1	0	—	0.89	—	—	11.09	—	—	0.89	—	32x32	2ds

Table 6.11: Wind vectors results by tracking raw images on 20th March.

Computed vectors			'Valid'vectors			Mean speed dev.			Mean dir. dev.			RMS speed dev.			Multi-vec.	Target	Tracking
L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	window	size	method
2	26	233	1	19	36	1.79	4.68	-1.79	5.30	1.59	-1.80	1.79	13.52	8.76	0	4x4	x
11	63	138	3	48	66	-2.78	4.65	5.02	19.30	0.19	-3.17	9.53	11.82	16.90	2	4x4	ssda
46	74	54	12	31	4	3.80	-3.26	0.27	11.39	9.80	0.50	7.22	8.98	4.74	2	4x4	2ds
5	18	67	4	17	39	-0.31	11.54	-0.66	-13.48	2.53	-3.13	14.16	13.06	13.17	1	8x8	x
10	32	55	5	28	40	4.35	8.75	0.99	10.65	1.94	-2.44	7.49	13.48	14.35	1	8x8	ssda
15	25	19	4	12	7	-3.43	-1.76	4.41	14.60	2.59	-8.13	10.13	10.43	14.71	0	8x8	2ds
4	14	26	3	13	22	-5.59	10.89	-2.03	5.51	1.57	-1.06	9.24	13.43	11.44	1	16x16	x
6	14	22	3	14	17	8.36	7.97	-4.14	-11.36	3.15	-2.18	10.31	10.50	11.21	2	16x16	ssda
5	5	4	2	5	3	-5.13	5.00	-6.70	-2.24	-2.18	-1.68	18.97	5.85	7.20	0	16x16	2ds
1	10	13	1	10	11	3.47	7.76	-5.39	-10.25	1.65	-1.52	3.47	11.73	13.31	2	24x24	x
2	12	9	2	10	9	4.29	7.23	1.81	-12.91	4.41	-3.59	7.26	10.27	6.62	1	24x24	ssda
4	4	3	2	3	3	8.80	1.80	-1.34	8.37	7.08	-2.65	8.91	6.96	9.73	0	24x24	2ds
0	2	1	0	2	1	—	13.16	1.56	—	11.59	4.76	—	13.56	1.56	0	32x32	x
0	1	1	0	1	1	—	8.65	-0.86	—	9.30	-1.50	—	8.65	0.86	0	32x32	ssda
0	0	0	0	0	0	—	—	—	—	—	—	—	—	—	0	32x32	2ds

Table 6.12: Wind vectors results by tracking clustered images on 20th March.

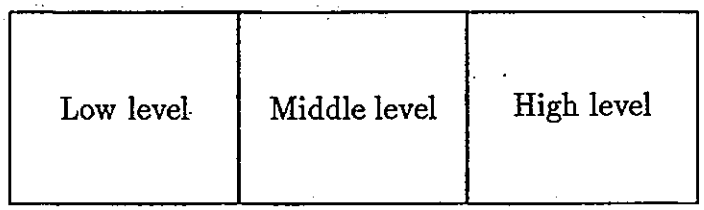


Figure 6.14: Reference wind field of 5th March interpolated on 16 x 16 target size grid.

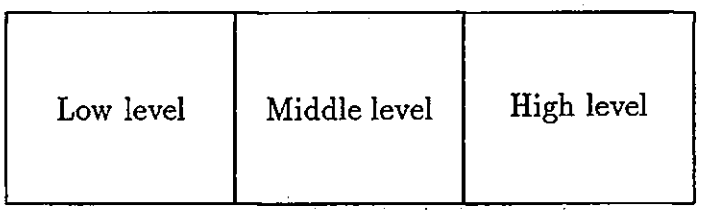


Figure 6.15: Reference wind field of 8th March interpolated on 16 x 16 target size grid.

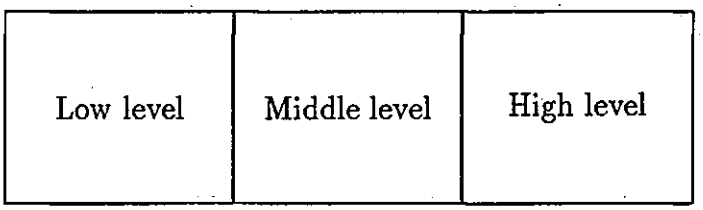


Figure 6.16: Reference wind field of 11th March interpolated on 16 x 16 target size grid.

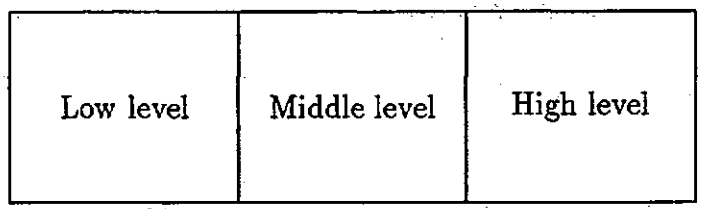


Figure 6.17: Reference wind field of 15th March interpolated on 16 x 16 target size grid.

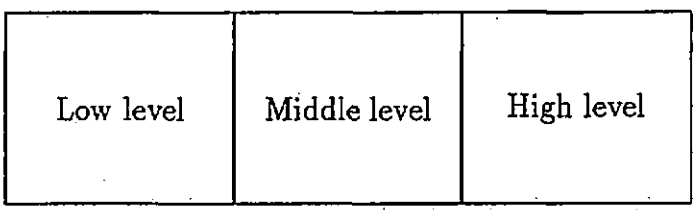


Figure 6.18: Reference wind field of 18th March interpolated on 16 x 16 target size grid.

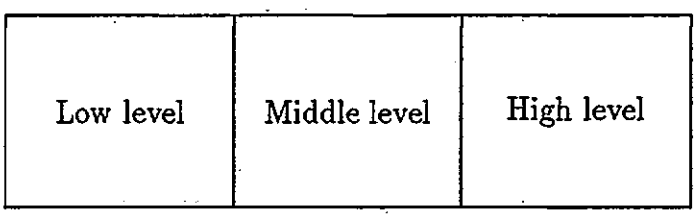


Figure 6.19: Reference wind field of 20th March interpolated on 16 x 16 target size grid.

Figures 6.20— 6.23 show the mean rms speed deviation of 'valid' vectors obtained by tracking using raw images. For low level wind vectors a clear minimum speed deviation for all three tracking methods occurs between 16 x 16 and 24 x 24 target sizes. However, a minimum can not be found for middle level and high level wind. Fig. 6.23 is the overall mean rms speed deviation for the three tracking method again a clear minimum for low level wind vector occurs at target size of 24 x 24. Lunnon and Lowe (1990) use target sizes of 4 x 4, 4 x 8, 8 x 8, 8 x 16, 16 x 16, 16 x 32 and 32 x 32, and found the optimum target size for METEOSAT images to be 16 x 16 for low level vectors.

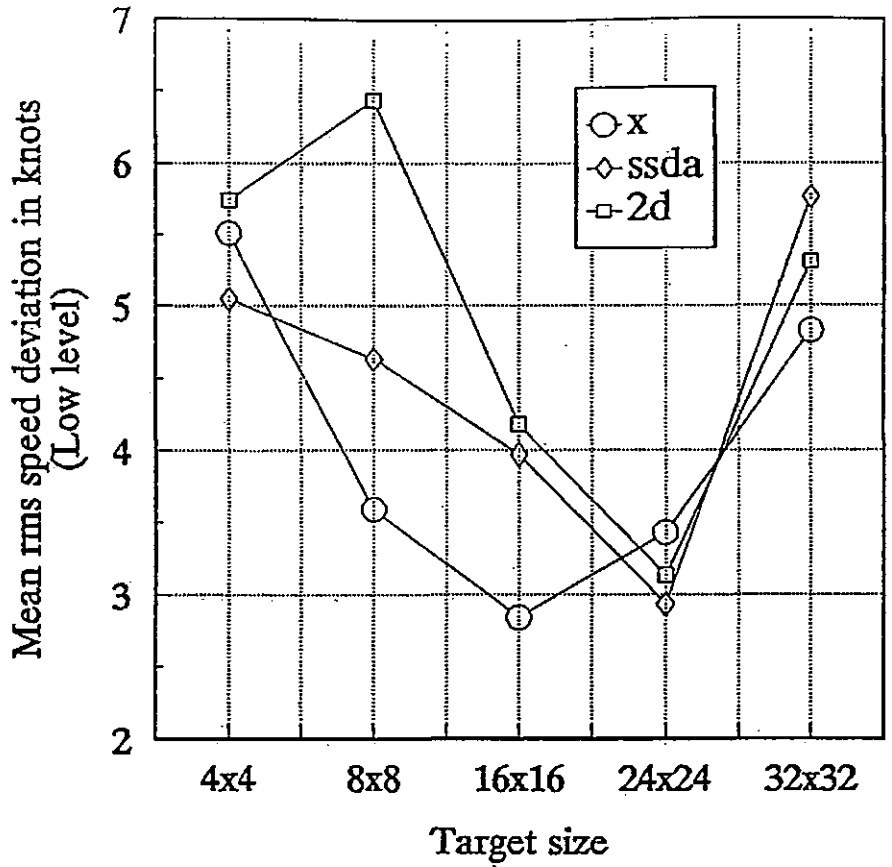


Figure 6.20: Mean rms speed deviation for different target sizes and tracking methods using raw images (Low level).

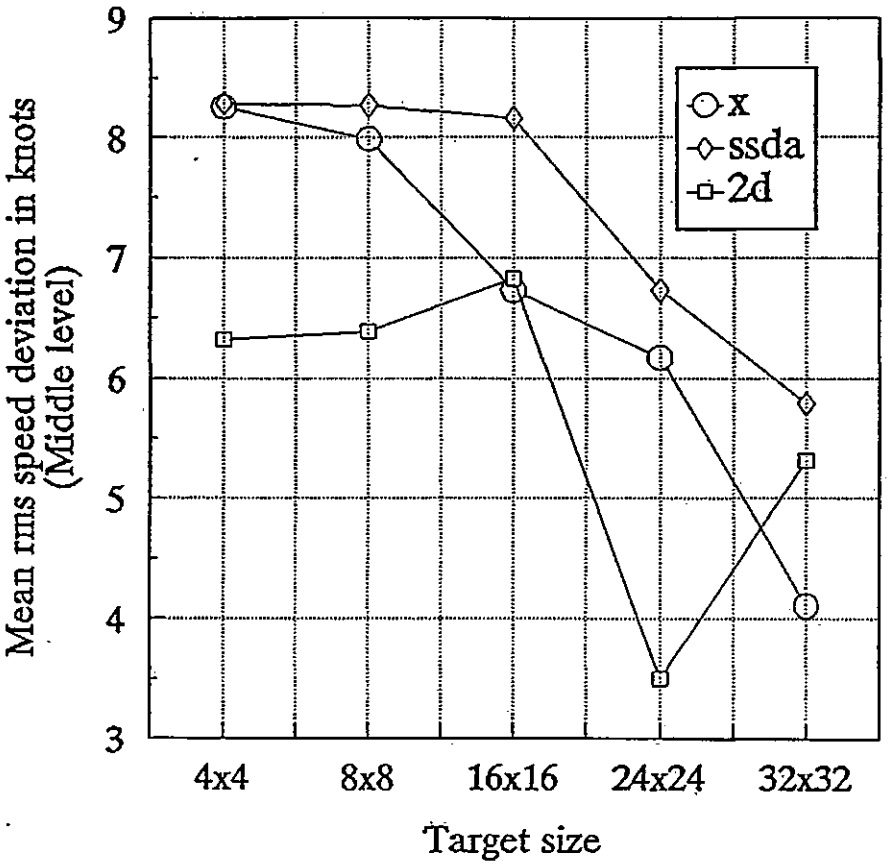


Figure 6.21: Mean rms speed deviation for different target sizes and tracking methods using raw images (Middle level).

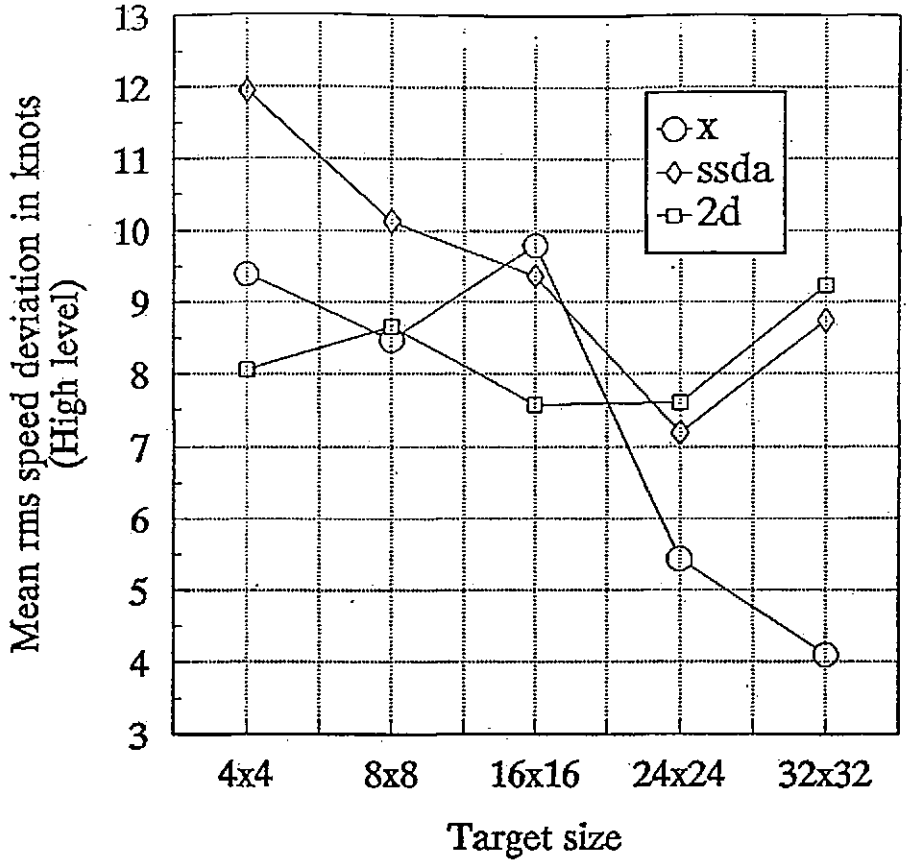


Figure 6.22: Mean rms speed deviation for different target sizes and tracking methods using raw images (High level).

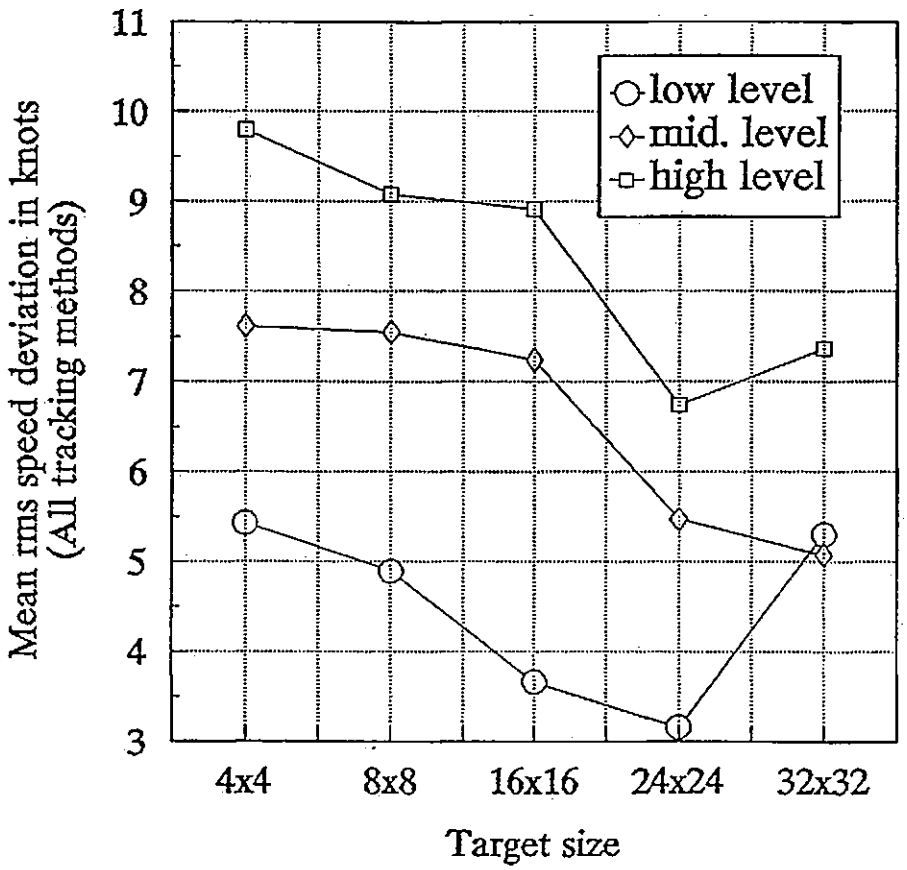


Figure 6.23: Mean rms speed deviation for different target sizes using raw images (All tracking methods).

Figures 6.24— 6.27 show the corresponding mean rms speed deviation for wind vectors obtained by tracking clustered images. It is noted that for clustered images the target window may not be fully filled with pixel especially with larger target window (say 16 x 16 and larger). Therefore the target sizes for clustered windows are undefined. No relationship of target size with speed deviation can be observed in these Figures. However, there is a tendency for the speed deviation to decrease with larger target windows for middle and high level wind vectors.

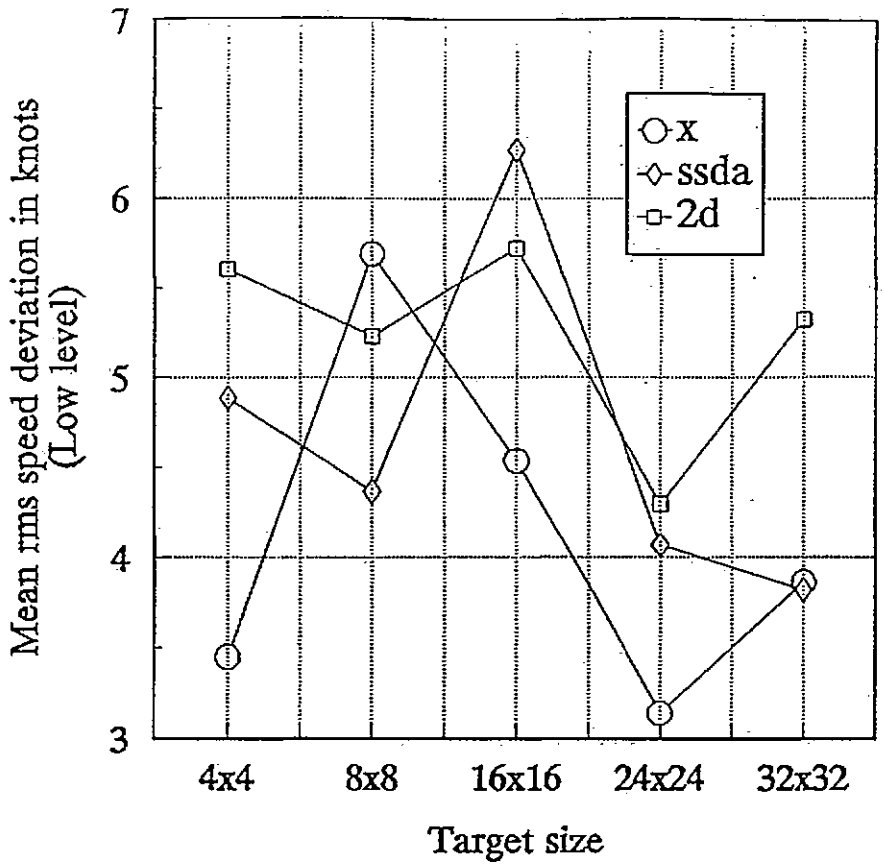


Figure 6.24: Mean rms speed deviation for different target sizes and tracking methods using clustered images (Low level).

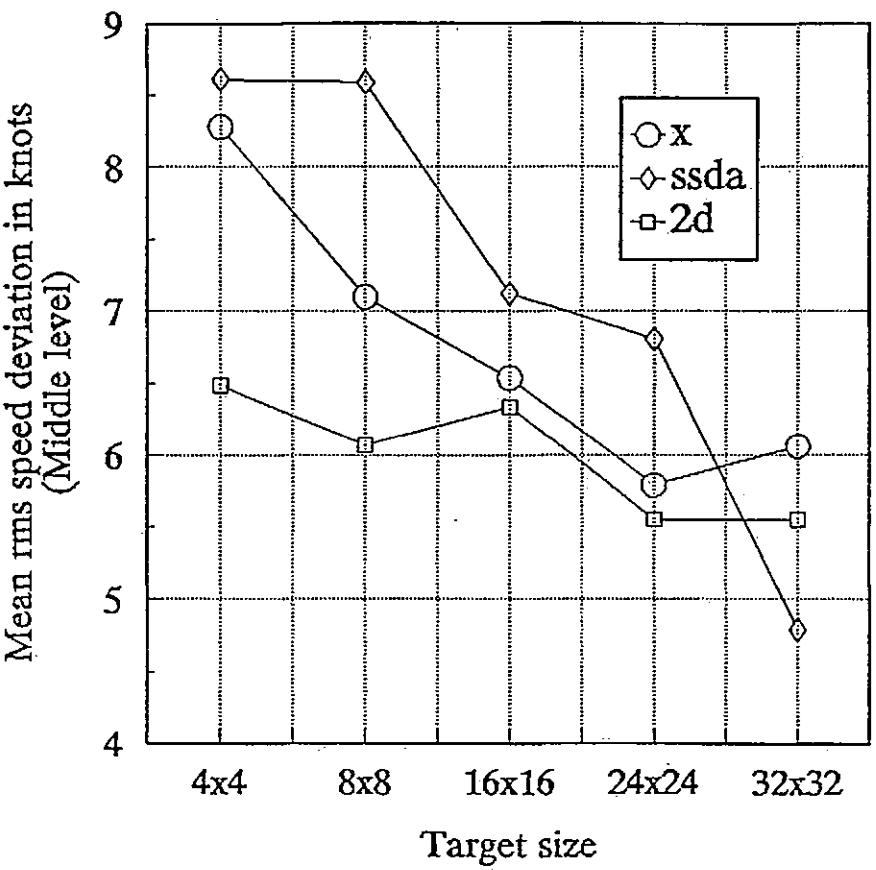


Figure 6.25: Mean rms speed deviation for different target sizes and tracking methods using clustered images (Middle level).

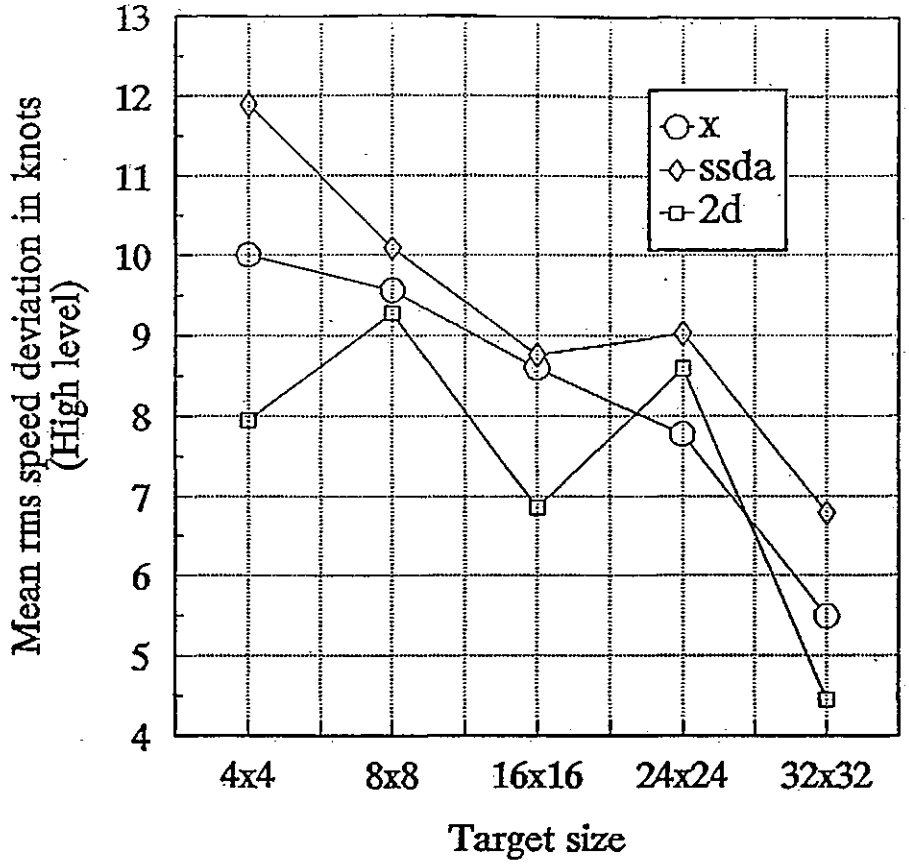


Figure 6.26: Mean rms speed deviation for different target sizes and tracking methods using clustered images (High level).

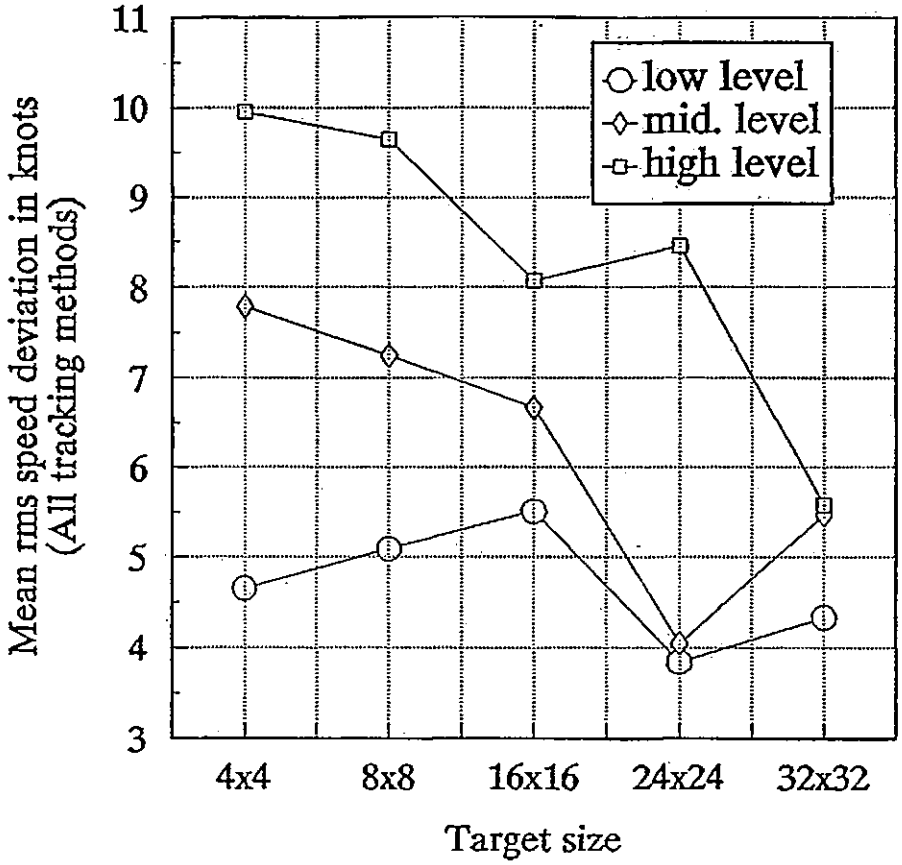


Figure 6.27: Mean rms speed deviation for different target sizes using clustered images (All tracking methods).

Figure 6.28 is the cumulative error for all 'valid' vectors. For low level wind 93% of vectors have a speed deviation less than 10 knots, and the error is almost identical for raw and cluster tracking. For middle level wind 85% of vectors have a speed deviation less than 10 knots, and for high level wind 76.5% have a speed deviation less than 10 knots.

The error for middle and high level vectors using clustered tracking is slightly ($\approx 1\%$) less than that using the raw tracking approach. Figure 6.28 also confirms the fact that high level winds have much larger error than low level vectors.

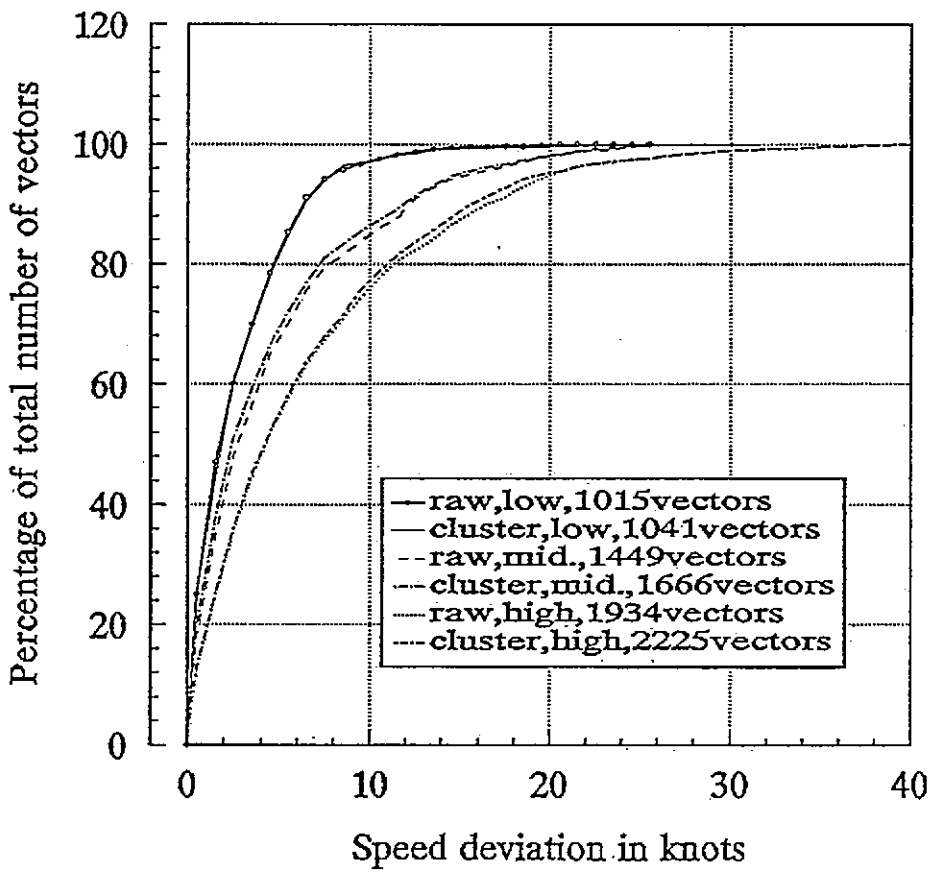


Figure 6.28: Cumulative speed deviation for low, middle, high level wind vectors obtained by tracking of raw and clustered images.

Figure 6.29— 6.31 shows the number of 'valid' vectors for different target size, and tracking methods. This reveals that, for a target size greater than 8 x 8, the number of 'valid' wind vectors computed using clustered images are significantly more than for raw images. For example, for 24 x 24 and larger targets, clustering gives approximately 50% more 'valid' vectors using cross correlation or SSDA. The diminishing advantage of using clustered image tracking for target sizes below 16 x 16 may be partly due to the diminishing chance of selecting a target covering more than one type of cloud when using raw images, and at the same time 'holes' are generated into the target window after removal of other clusters. The general increase in the number of vectors using clustered image tracking strongly suggests that multi-layers cloud motion can be tracked better by first separating different cloud types and tracking them individually.

Fig. 6.31 also reveal that clustering method generates more vectors for all target sizes. In order to obtain a correct match point the 2d search requires matching surfaces to have a dominant peak or valley. This suggests clustering improves the general shape of matching surfaces, such that more consistent matching can be achieved.

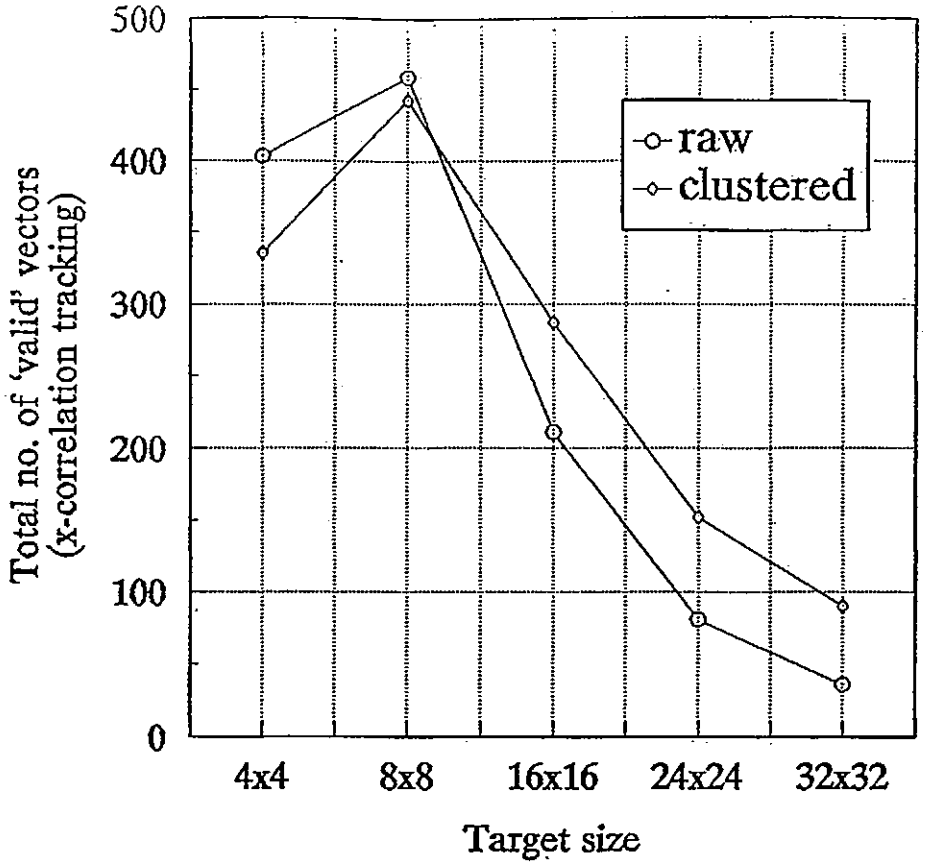


Figure 6.29: Total number of 'valid' vectors using different target sizes, cross correlation tracking with raw and clustered images.

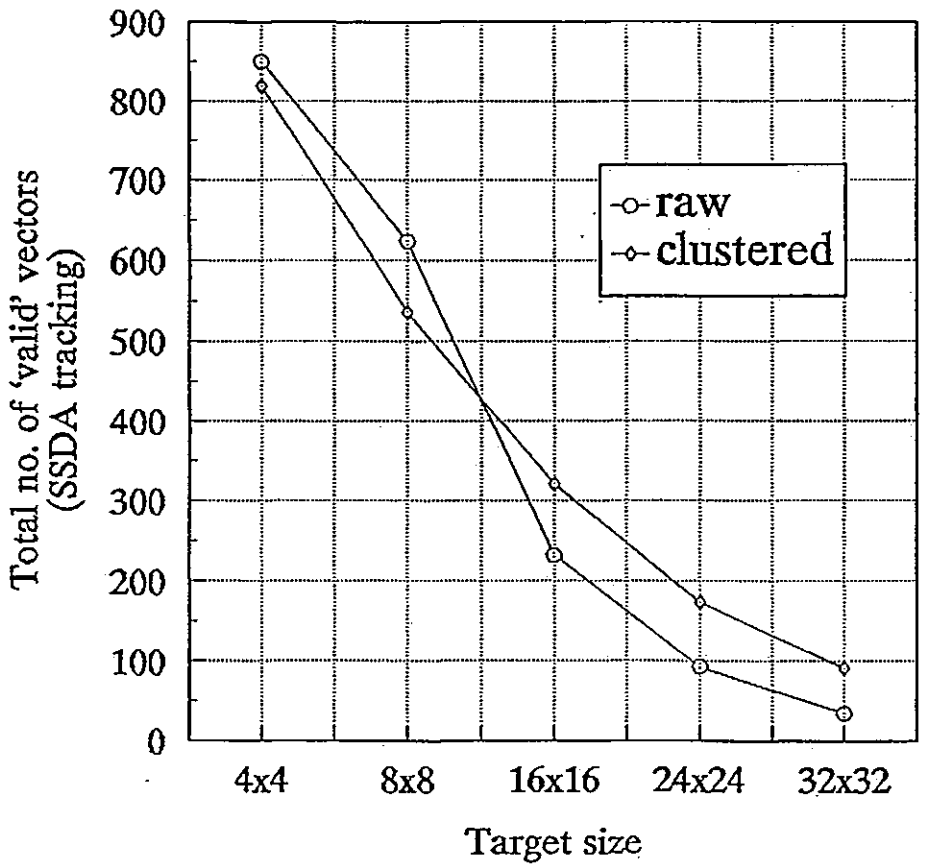


Figure 6.30: Total number of 'valid' vectors using different target sizes, SSDA tracking with raw and clustered images.

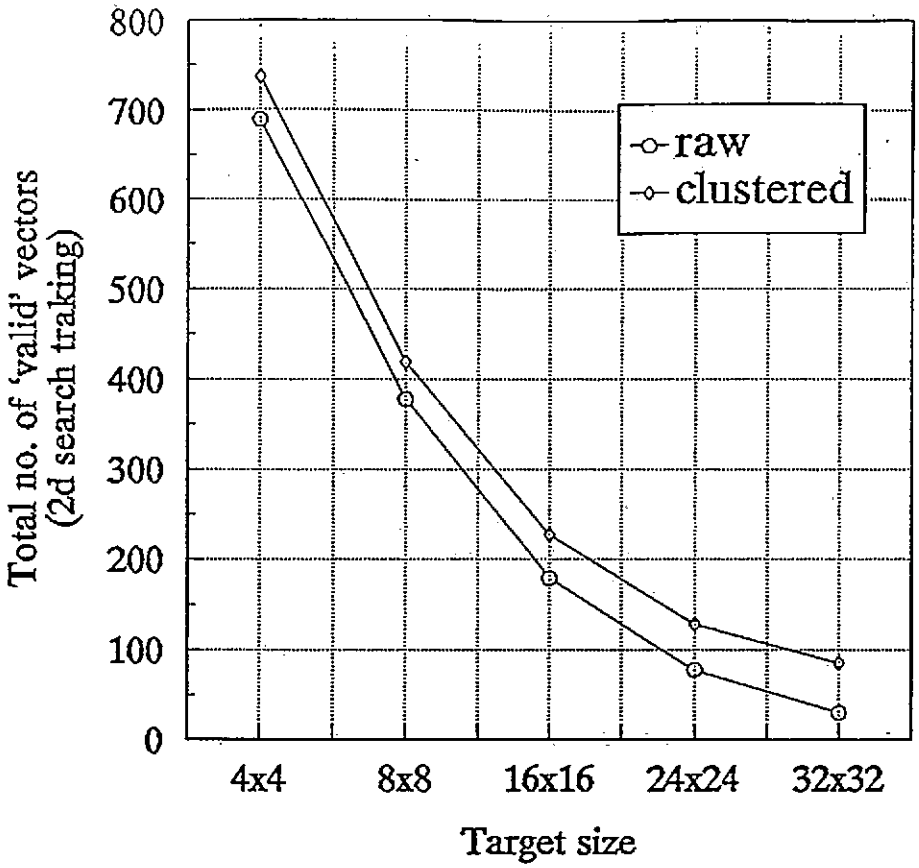


Figure 6.31: Total number of 'valid' vectors using different target sizes, 2d search tracking with raw and clustered images.

The 18th March image is a very difficult case for cloud tracking, this is shown in Table 6.9 and 6.10 that wind vectors can only be obtained using SSDA, and cross correlation has produced none at all, this is mainly because the whole image is covered by a big lump of bright featureless frontal cloud. The tracking ability of the three tracking methods can be seen in Fig. 6.32— 6.33 for raw and clustered tracking respectively. SSDA has been able to produce more wind vectors that cross correlation tracking for all target sizes. Therefore it should be used for cloud motion tracking instead of cross correlation for its higher efficiency and tracking ability. The higher tracking ability can be explain by the fact that the matching surfaces of SSDA are generally less 'spiky' than cross correlation (Fig. 6.5 and 6.6).

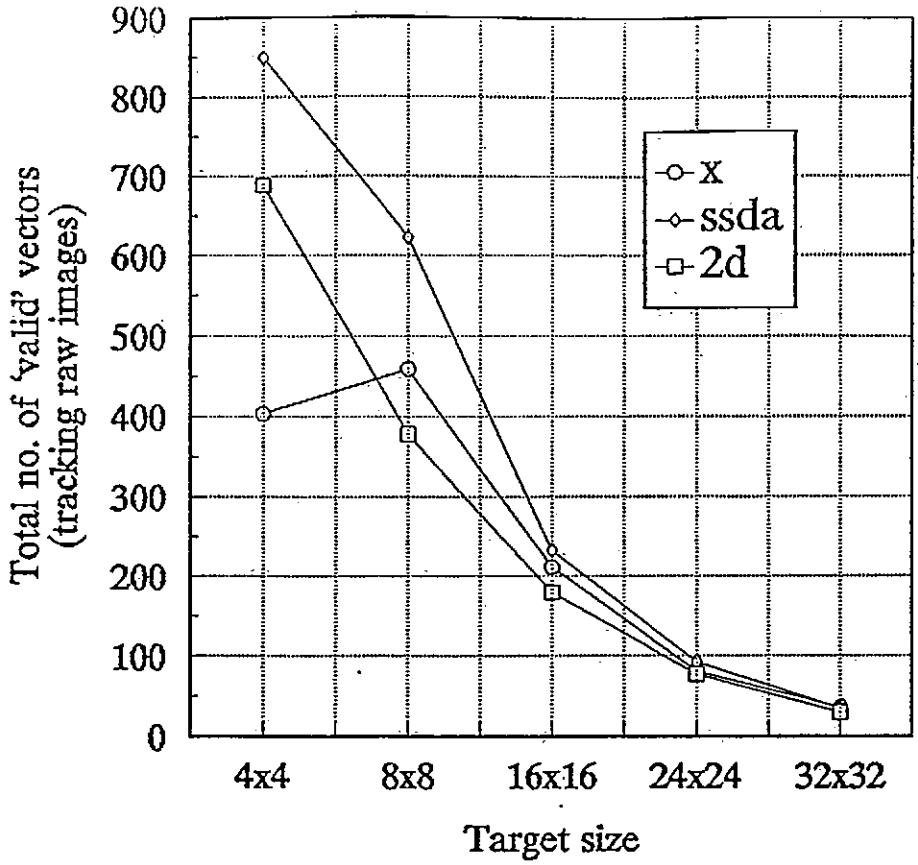


Figure 6.32: Total number of 'valid' vectors using different target sizes, tracking methods with raw images.

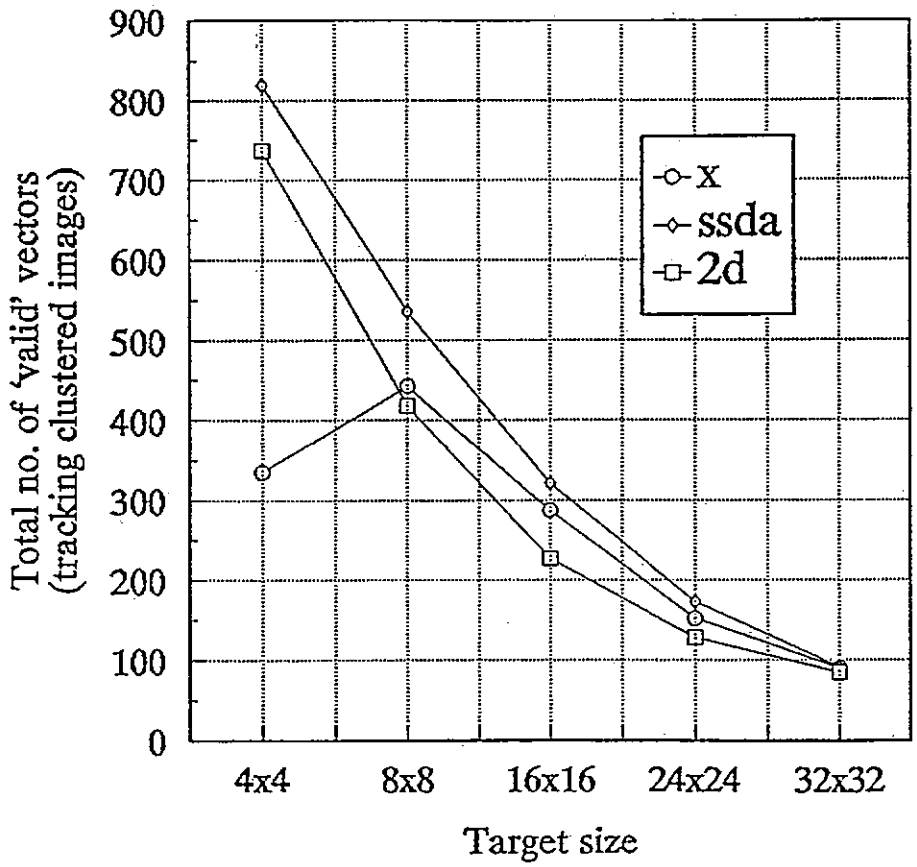


Figure 6.33: Total number of 'valid' vectors using different target sizes, tracking methods with clustered images.

Figures 6.34— 6.39 show the wind field computed by SSDA tracking of raw and clustered images with a target size of 24 x 24.

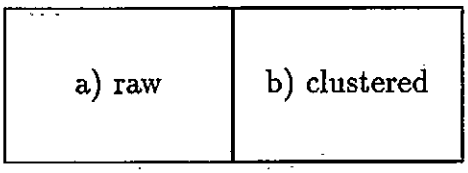


Figure 6.34: Wind field of 5th March, a) raw tracking, and b) clustered tracking (24 x 24 target size).

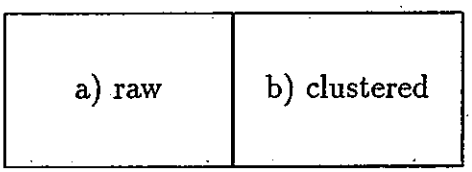


Figure 6.35: Wind field of 8th March, a) raw tracking, and b) clustered tracking (24 x 24 target size).

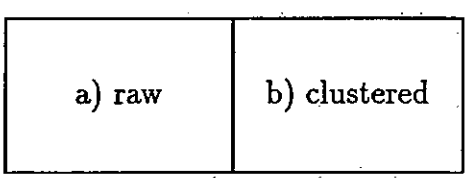


Figure 6.36: Wind field of 11th March, a) raw tracking, and b) clustered tracking (24 x 24 target size).

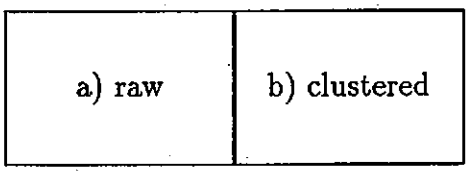


Figure 6.37: Wind field of 15th March, a) raw tracking, and b) clustered tracking (24 x 24 target size).

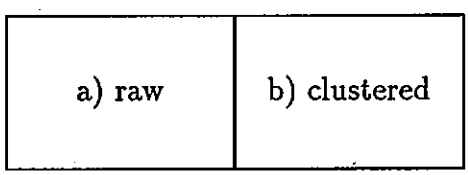


Figure 6.38: Wind field of 18th March, a) raw tracking, and b) clustered tracking (24 x 24 target size).

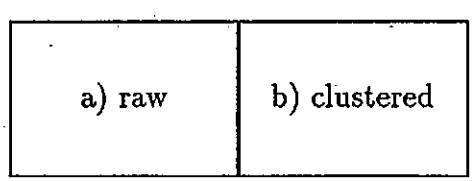


Figure 6.39: Wind field of 20th March, a) raw tracking, and b) clustered tracking (24 x 24 target size).

The 2d search method although produces least number of 'valid' vectors, but it is much more efficient that cross correlation and SSDA, if large wind field is required then it can be used with overlap target window to increase the total number of wind vectors.

6.5 Discussion

The clustering approach for cloud wind tracking has proved to increased the yield of wind vector by as much as 50% for target size of 24 x 24. The advantage stems from the ability of clustering to select natural data patterns, which in turn tends to correspond to different cloud types and by tracking a single cluster, features such as cloud boundaries can improve the matching process. Clustering approach is found to neither increase nor decrease the wind vector error.

It is also found that there is an optimum target size for a given wind resolution, and it is 24 x 24 for low level wind using raw images. However, this criterion does not apply to clustered image tracking, since, in this case, the effective target size is variable, also no relationship of target size and speed deviation can be observed at all.

The SSDA is found to be more reliable method for target tracking (in this study an infinity threshold is used, therefore SSDA degenerate to mean absolute error method), than cross correlation; it is also much faster since it avoids the need for normalisation. The 2d search is most efficient and can be used when a large wind field is required.

Chapter 7

Conclusion

Clustering techniques have proved to be effective for improving cloud motion tracking in multi-layer regions. Two clustering techniques have been developed, 1) The Global-Local clustering algorithm and 2) Spatial-Spectral clustering algorithms. Both of them assumed clusters to have a Gaussian distribution, since this assumption is found to be reasonable for METEOSAT imagery.

7.1 Global-Local Clustering Algorithm

All iterative partitional clustering algorithms require an initial partition or set of centres to start and it is well known that different initial conditions generate different clustering. In conventional clustering of image data initial partitions or centres are obtained manually and therefore bias is introduced. The bias can be eliminated by choosing some random initial centres or partition, but it is also well known that clustering can be trapped in some local optimum which may not be the global optimum.

A classical and almost universal solution to this problem is to allow clusters which are likely to contain points from more than one class to split, or conversely, to permit clusters which are very similar to merge. In this study it has been shown that split and merge does not solve the problem completely. Ideally the initial partition should be as close to the best solution (global optimum) as possible

to avoid trapping in a local optimum. This is achieved by using a Global-Local clustering approach. Essentially the first stage is to identify modes in the feature space and then generate an initial partition which is close to the best solution using a histogram clustering algorithm. The second stage is to optimize the initial partition using an iterative clustering algorithm. The Global-Local clustering approach is found to be effective in improving the clustering.

The second problem of using clustering techniques is the choice of distance function. The cluster mean has been used as a cluster model for a long time, but the deficiency of its ability to recover clusters with large difference in variance has long been realised. The classical solution to the cluster variation problem is to normalise the variables, such that cluster variation can be reduced. The normalisation can be performed using simple scaling of variables, clustering transform, and principle component transform etc. All these techniques do not require any prior knowledge of the data structure, and therefore if the data structure does not conform to the normalisation model, the discrimination between clusters may be decreased. However, if a parametric model can be assumed, then it is possible to derive a distance function which is optimum with respect to certain criteria (e.g. Minimum Sum of Error probability) and this implies the data structure are known beforehand.

Dynamic clustering allows the latter approach to be implemented. The commonly used cluster mean model is replaced by a kernel, and this may take any model which represents the clusters best. In this study a Gaussian model is used, which allows clusters to have different size and variance. Several distance functions can be derived from the Gaussian model with different levels of simplification, and in this way the cluster mean model can be related to the Gaussian model.

The number of patterns in an multi-spectral image is very large, so any clustering algorithm should perform efficiently. The Global-Local algorithm achieves high efficiency by exploiting the multiple occurrence of distinct vectors. Distinct vectors are clustered using the dynamic algorithm and if a distinct vector is assigned to a particular cluster so are the other copies of it. Since the histogram

clustering is also very efficient the combined algorithm remains so.

7.2 Spatial-Spectral Clustering Algorithm

Contextual information exists in all imagery, although it is usually ignored in image analysis. Classification of a pixel may be ambiguous if it is a mixture of different objects, usually these pixels lie on the boundaries of images. The contextual information can be exploited by considering a pixel and it neighbours. The Spatial-Spectral clustering algorithm is based on the assumption that if a pixel has uncertain classification then it should be assigned to the class with the majority of its neighbours belongs to.

The contextual information has been exploited by clustering of the spatial space using Graph Theoretic Hierarchical Segmentation (GTHS). This results in a number of homogeneous regions which corresponds to objects in the image and these objects are then grouped based on their spectral similarity.

The spatial clustering is a stepwise optimum approach. It is also a generalization of the hierarchical clustering approach which usually clusters patterns based on their spectral similarities. The spatial clustering algorithm constructs an image graph based on two types of distance function: local and global. Local type distances only use the information of the nearest neighbours, while global type distances use information of other pixels by considering the statistics of a neighbouring region.

Segmentation is usually compared visually but it is desirable to have a measure to indicate the goodness of a segmentation. To this end a mutual information model has been developed to monitor the segmentation process and the entropy loss is used as an indicator of the generation of the basic features in an image. A good segmentation should use the least number of segments to retain the maximum amount of information of the original image, and so the segmentation should stop when the basic features have been recovered and the entropy loss starts to decrease at a constant rate. Although this criterion is heuristic, it has found to provide

good segmentation.

The clustering of segments is also based on the agglomerative approach: a statistical hypothesis is used as a distance measure between two segments and every time the two most similar segments are merged. Clustering stops when the number of groups equals the number of clusters required.

The Spatial-Spectral clustering algorithm therefore requires two user parameters; 1) number of segments, 2) number of clusters. It is found that the first parameter is quite independent of the data once the windows size is fixed. Spatial-Spectral clustering is computation expensive so it is not used for testing of cloud wind tracking. However, it is able to produce clusters with 'clean' boundaries, and this ability should be valuable if further machine analysis of cluster shape is required.

Due to the lack of ground truth, it was not able to access the accuracy of the two clustering algorithms quantitatively. However, the clusterings compare favourably with the original observation.

7.3 Generation of Cloud Motion Vectors

Cloud motion wind has been generated by tracking raw and clustered images. The separation of cloud type prior to cloud tracking has been proposed a long time ago (Hubert 1971). However most operational wind systems do not take advantage of the multi-spectral information and only used simple IR thresholding to segment a target window before tracking cloud targets. The efficient Global-Local clustering algorithm has been designed for operational use, and even a single band of infrared can be used at night time when other channels are not available. It is the author's experience that most of the control parameters can be fixed once the first set of images have been processed. Specifically, the next clustering process (providing the two sets of images are similar) can use the previous clusters as a starting partition. Also other parameters for split or merge can be kept unchanged. Therefore the algorithm is only used interactively when necessary to

generate initial partition and specify the clustering parameters. At present the clustering only generates unlabelled classes, but the study by Seddon (1985) for example can be used to produces labelled classes for better use of the wind vectors.

The clustering approach has been able to increase the wind vector yield by as much as 50%, for target sizes of 16 x 16 or larger. The advantage for larger targets is mainly due to the fact that mixed cloud types in a small target are less probable. Also, clustering introduces 'holes' into the target which confuses the matching process.

Although more vectors can be generated by the clustering approach, it does not seem to improve the speed deviation significantly. It must be noted that only a small set of images are used in this study, and ideally more statistics are required to firm the results.

The effect on varying the target size on the mesoscale wind speed deviation has been investigated. There is a tendency for larger target size to produce smaller speed deviation, and no obvious minimum for high and middle level wind can be found between 4 x 4 and 32 x 32 target size. On the other hand, an obvious minimum for low level wind with a target size of 24 x 24 can be found for all tracking methods, and this result agrees with the Meteorological Office findings (Lunnon and Lowe 1991).

Cross correlation has been used for image tracking since the very first application, since it possesses the matched filter characteristic and provides an optimum result when the image is corrupted by white noise. In this study the simple SSDA (absolute mean error) is found to be more reliable than the classical cross correlation and it is highly recommended as a substitute for cross correlation tracking. If even higher efficiency is required the two-dimensional search method can be employed.

Any future work should be concentrated on using a more sophisticated classification algorithm to separate cloud types before tracking cloud targets. Large high level wind error is a major problem, and it is hoped that it can be improved by better classification methods. It is noted that tracking ability is not the only

factor to affect high level wind error.

Bibliography

[1] Abidi,M.A., and Gonzalez,R.C., "Motion detection in radar images", International Conference of Pattern Recognition, Montreal, Canada, 787-790, 1984.

[2] Abidi,M.A., and Gonzalez,R.C., "Cloud motion measurement from radar image sequences", Proc. SPIE Vol.846, Digital image processing and visual communications technologies in Meteorology, 54-60, 1987.

[3] Aggarwal,J.K., and Duda,R.O., "Computer analysis of moving polygonal images", IEEE Trans. on Computers, Vol.C-24, No.10, 966-976, 1975.

[4] Aggarwal,J.K., Davis,L.S., and Martin,W.N., "Correspondence processes in dynamic scene analysis", Proceedings of IEEE, Vol.69, No.5, 562-572, 1981.

[5] Aitchison,J., Habbema,J.D.F., and Kay,J.W., "A critical comparison of two methods of statistical discrimination", Applied Statistics, 26, 15-25, 1977.

[6] Amadasun,M., and King,R.A., "Low-level segmentation of multispectral images via agglomerative clustering of uniform neighborhoods", Pattern Recognition, Vol.21, No.3, 261-268, 1988.

[7] Anderberg,M.R., Cluster analysis for applications, Academic Press, Inc. New York, 1973.

[8] Andrews,H.C., Introduction to mathematical techniques in pattern recognition, John Wiley and Sons, Inc., New York, 1972.

[9] Anderson, R.K., and Veltishchev, N.F., ed., "The use of satellite pictures in weather analysis and forecasting", World Meteorological Organization, Technical Note No.124, 1973.

[10] Anuta,P.E., "Spatial registration of Multispectral and Multitemporal Digital Imagery Using Fast Fourier Transform Techniques", IEEE Tran. Geoscience Electronics, Vol.GE-8, No.4, 353-368, 1970.

[11] Arcese,A., Mengert,P.H., and Trombini,E.W., "Image detection through bipolar correlation", IEEE Tran. on Information Theory, Vol.IT-16, No.5, 534-541, 1970.

[12] Arking, A., and Chlids, J.D., "Retrieval of cloud parameters from multispectral satellite images", Journal of Climate and Applied Meteorology, Vol.24, 322-333, 1985.

[13] Arking,A, Lo,R.C., and Rosenfeld,A., "A Fourier approach to cloud motion estimation", Journal of Applied Meteorology, Vol.17, 735-744, 1978.

[14] Ashby,W.R., "Measuring the internal informational exchange in a system", Cybernetica, Vol.1, 5-22, 1965.

[15] Astrahan, M.M., "Speech analysis by clustering, or the hyperphoneme method", Stanford Artificial Intelligence Proj. Men. AIM-124, AD 709067. Stanford Univ., Standford, California, 1970.

[16] Ayres,F., Schaum's outline of theory and problems of plane and spherical trigonometry, Schaum Publishing Company, New York, 1954.

[17] Ball,G.H., and Hall,D.J., "A clustering technique for summarizing multivariate data", Behavioural Science, Vol.12, 1967, 153-155.

[18] Ball,G.H., "Data analysis in the social sciences:What about the details?", Proc. of Fall Joint Comput. Conf., 27, part 1, 533-559, 1965.

[19] Bader, M., and Waters, T., "Satellite and Radar Imagery Interpretation", Preprints for a Workshop on Satellite and Radar Imagery Interpretation, Meteorological Office College, England, 1987, EUMETSAT.

[20] Barnea,D.I., and Silverman,H.F., "A class of algorithms for fast digital image registration", IEEE Transactions on Computers, Vol.C-21, 1972, 179-186.

[21] Bauer,K.G., "A comparison of cloud motion winds with coinciding radiosonde winds", Monthly Weather Review, Vol.104, 922-931, 1976.

[22] Beaulieu,J-M., and Goldberg,M., "Hierarchy in Picture Segmentation: A Stepwise Optimization Approach", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.11, No.2, 150-163, 1989.

[23] Bernstein,R., "Digital image processing of earth observation sensor data", IBM Journal of Research and Development, Vol.20, 40-57, 1976.

[24] Bos,A.M., de Waard,J., and Adamson,J., "Real-time rectification of Meteosat images", European Space Agency Journal 1990, Vol.14, 179-191.

[25] Bowen,R.A., and Saunders,R.W., "The semi-transparency correction as applied operationally to Meteosat infrared data: A remote sensing problem", European Space Agency Journal 1984, Vol.8, 125-131.

[26] Bowen,R.A., Fusco,L.,Morgan,J.,and Roska,K.O., 1979, "Operational production of cloud motion vectors(satellite winds) from METEOSAT image data: Use of data from meteorological satellites", Paper ESA SP 143, European Space Agency, Paris.

[27] Bowen,R.A., "The meteorological product: cloud-top height", European Space Agency Bulletin, No.30, 16-20, 1982.

[28] Bowen,R.A., and Saunders,R., "The semi-transparency correction as applied operationally to Meteosat IR data", ESA Journal 8, 125-131.

[29] Bow,S.T., Pattern recognition, Marcel Dekker, Inc., New York, 1984.

[30] Bradford, R., Leese,J., and Novak,C., "An experimental model for the automated detection, measurement, and quality control of low-level cloud motion vectors from geosychronous satellite data", Proc. 8th International Symposium on Remote Sensing of Environment, 441-462, 1972.

[31] Bristor,C.L., (ED.), GREEN,R., HUGHES,G., NOVAK,C., and SCHREITZ,R., "The automatic extraction of wind estimates from VISSR data", NOAA technical memoradum NESS 64, U.S.DEPT. of COMMERCE, WASHINGTON, D.C., 94-110, 1975.

[32] Bruce,J., Thomas,B., and Dubes,R., "A variation on a nonparametric clustering method", IEEE Transcations on Pattern Analysis and Machine Intelligence, Vol.PAMI-1, No.4, 400-408, 1979.

[33] Bryant,J., "A fast classifier for image data", Pattern Recognition, Vol.22, No.1, 45-48, 1989.

[34] Bryant,J., "On the clustering of multidimensional pictorial data", Pattern Recognition, Vol.11, 115-125, 1979.

[35] Bryant,J., "AMOEBA clustering revisited", Photogrammetric Engineering and Remote Sensing, Vol.56, No.1, 41-47, 1990.

[36] Butler,G.A., "Evaluating feature spaces for the two class problem", IEEE Annual Symposium Record, IEEE Systems, Man and Cybernetics, 119-125, 1971.

[37] Butler,G.A., Ritea,H.B., "Estimation of mutual information in two class pattern recognition", IEEE Trans. on Computers, 410-420, April 1974.

[38] Byrne,G.F., Crapper,P.F., and Mayo,K.K., "Monitoring land-cover change by principal component analysis of multitemporal Landsat data", Remote Sensing of Environment, 10, 175-184, 1980.

[39] Campbell,J.B., Introduction to remote sensing, The Guildford Press, London, 1987.

[40] Cannon,R.L, Dave,J.V., Bezdek,J.C., and Trivedi,M.M., "Segmentation of a thematic mapper image using the fuzzy c-means clustering algorithm", IEEE Trans. on Geoscience and Remote Sensing, Vol.GE-24, No.3, 400-408, 1986.

[41] Casey, R.G., and Nagy, G., "An autonomous reading machine", IEEE Trans. Computers, vol.C-17, 492-503, 1968.

[42] Cayla,F.R., and Tomassini,C., "Determination of the temperature of semi-transparent cirrus", ESA Special Publication, SP-143, Oct., 1979.

[43] Chen,Z., and Fu,K.S., "On the connectivity of clusters", Information Science, 8, 283-299, 1975.

[44] Chidananda Gowda,K., and Krishna,G., "Agglomerative clustering using the concept of mutual nearest neighbourhood", Pattern Recognition, Vol.10, 105-112, 1978.

[45] Chou,M.D., Childs,J., and Dorian,P., "Cloud cover estimation using bispectral satellite measurements", Journal of Climate and Applied Meteorology, Vol.25, 1281-1292, 1986.

[46] Coakley,J.A., and Baldwin,D.G., "Towards the objective analysis of clouds from satellite imagery data", Journal of Climate and Applied Meteorology, Vol.23, 1065-1099, 1984.

[47] Coakley, J.A., and Bretherton, F.P., "Cloud Cover From High-Resolution Scanner Data: Detecting and Allowing for Partially Filled Fields of View", Journal of Geophysical Research, Vol.87, No.C7, 4917-4932, 1982.

[48] Coggins,J.M., and Jain,A.K., "A Spatial Filtering Approach to Texture Analysis", Pattern Recognition Letters, 3, 195-203, 1985.

[49] Coleman,G.B., and Andrews,H.C., "Image segmentation by clustering", Proc. of The IEEE, Vol.67, No.5, 773-785, 1979.

[50] Conrow,E.H., and Ratkovic,J.A., "Almost everything one needs to know about image matching systems", Image Processing for Missile Guidance, SPIE Proc. 238, 426-453, 1980.

[51] Cover,T.M., and Hart,P.E., "Nearest neighbour pattern classification", IEEE Trans. Info. Theory, IT-13, 21-27, 1967.

[52] Cracknell, A.P., Remote Sensing in Meteorology, Oceanography and Hydrology, Ellis Horwood Limited, Chichester, 1981.

[53] Daskalakis,T.N., Heaton,A.G., and Daskalakis,C.N., "A graph-theoretic algorithm for unsupervised image segmentaion", Signal Processing IV: Theories and Applications, North Holland, 1621-1624, 1988a.

[54] Daskalakis,T.N., Heaton,A.G., and Daskalakis,C.N., "Minimax variance entropy-based image segmentation", IERE Fifth Int. Conf. on Digital Processing of Signals in Communications, Loughborough University of Technology, 20th-30rd Sep., 1988b, 291-297.

[55] Decotiis,A.G., and Conlan,E., "Cloud information in the three spatial dimensions using IR thermal imagery and vertical temperature profile data", Proc. 7th Symposium on Remote Sensing of the Environment. ANN ARBOR/MI, 595-606, 1971.

[56] Desbois,M., and Seze,G., "Use of space and time sampling to produce representative satellite cloud classifications", Annales de Geophysique, 1984, 2, 5, 599-605.

[57] Desbois,M., Seze,G., and Sewjwach,G., "Automatic classification of clouds on METEOSAT imagery: Application to high-level clouds", Journal of Applied Meteorology, 21, 401-412, 1982.

[58] Devijver,P.A., and Kittler,J., Pattern Recognition, A statistical approach, Prentic-Hall, International, Inc., Englewood Cliffs., N.J., 1982.

[59] Diday,E., "Optimization in non-hierarchical clustering", Pattern Recognition, Vol.6, 17-33, 1974.

[60] Diday,E., Simon,J.C., "Clustering analysis", Digital Pattern Recognition, K.S.Fu (ed.), Springer-Verlag, 47-94, 1976.

[61] Drake,K.W., and Gerhardt,L.A., "A class of pdf modeling algorithms", IEEE Trans. on Systems, Man, and Cybernetics, Vol.SMC-2, No.3, 402-407, 1972.

[62] Dubes,R., and Jain,A.K., "Validity studies in clustering methodologies", Pattern Recognition, Vol.11, 235-254, 1979.

[63] Dubes,R., and Jain,A.K., "Clustering techniques: The user's dilemma", Pattern Recognition, Vol.8, 247-260, 1976.

[64] Duda,R.O., and Hart,P.E., Pattern classification and scene analysis, John Wiley and Sons, New York, 1973.

[65] Ebert, E., "A pattern recognition technique for distinguishing surface and cloud types in the polar regions", Journal of Climate and Applied Meteorology, Vol.26, 1412-1427, 1987.

[66] Ebert, E., "Analysis of polar clouds from satellite imagery using pattern recognition and a statistical cloud analysis scheme", Journal of Applied Meteorology, Vol.28, 382-399, 1989.

[67] Eigen, D.J., Fromm, F.R., and Northouse, R.A., "Cluster analysis based on dimensional information with applications to feature selection and classification", IEEE Trans. on Systems, Man, and Cybernetics, Vol.SMC-4, No.3, 284-294, 1974.

[68] Eigenwilling,N., and Fischer,H., "Determination of midtropospheric wind vectors by tracking pure water vapour structures in METEOSAT water vapour image sequences", Bull. Amer. Meteor. Soc., Vol.63, No.1, 44-58, 1982.

[69] Endlich,R.M., Wolf,D.E., Hall,D.J., and Brain,A.E., "Use of a pattern recognition technique for determining cloud motions from sequences of satellite photographs", Journal of Applied Meteorology, Vol.10, 105-117, 1971.

[70] Endlich,R.M., and Wolf,D.E., "Automatic cloud tracking applied to GOES and METEOSAT observations", Journal of Applied Meteorology, Vol.20, 309-319, 1981.

[71] Everitt, B.S., and Hand, D.J., Finite Mixture Distributions, Chapman and Hall, London, 1981.

[72] Farag,R.F.H., "An information theoretic approach to image partitioning", IEEE Trans. on Systems, Man and Cybernetics, Vol.SMC-8, No.11, 829-833, 1978.

[73] Fischer,H., Eigenwilling,N., and Muller,H., "Information content of METEOSAT and Nimbus/THRR water vapour channel data: Altitude association of observed phenomena", Journal of Applied Meteorology, Vol.20, 1344-1352, 1981.

[74] Fix,E., and Hodges,J.L., "Discriminatory Analysis; Non-parametric Discrimination: Consistency Properties", USAF School of Aviation Medicine Project Number 21-49-004, Rept. No. 4, Randolph Field, Texas, 1951.

[75] Forgy, E.W., "Cluster analysis of multivariate data:efficiency verus interpretability of classifications", Biometrics 21, No.3, 768, 1965, (Abstract).

[76] Freeman,J., "Experiments in discrimination and classification", Pattern Recognition, Vol.1, 207-218, 1969.

[77] Friedman, H.P., and Rubin, J., "On some invariant criteria for grouping data", Amer. Stat. Assoc. J., vol.62, 1159-1178, 1967.

[78] Fromm,F.R., and Northouse,R.A., "CLASS: A nonparametric clustering algorithm", Pattern Recognition, Vol.8, 107-114, 1976.

[79] Fu,K.S., and Mui,J.K., "A Survey on Image Segmentation", Pattern Recognition, Vol.13, 3-16, 1981.

[80] Fu,K.S., "Pattern recognition in remote sensing of the earth's resources", IEEE Trans. on Geoscience Electronics, Vol.GE-14, No.1, 10-18, 1976.

[81] Fu,K.S., and Rosenfeld,A., "Pattern recognition and image processing", IEEE Trans. on Computers, Vol.C-25, No.12, 1336-1346, 1976.

[82] Fukunaga,K., and Koontz,W.L.G., "A Criterion and an Algorithm for Grouping Data", IEEE Trans. Computers, Vol.C-19, 917-923, 1970.

[83] Fujita,T.T., and Pearl,E.W., "Satellite-tracked cumulus velocities", Journal of Applied Meteorology, Vol.14, 407-413, 1975.

[84] Garand, L., "Automated recognition of oceanic cloud patterns. Part I: Methodology and Application to cloud climatology", Journal of Climate, Vol.1, 20-39, 1988.

[85] Garrett,G.S. Reagh,E.L., and Hibbs,E.B.Jr., "Detection threshold estimation for digital area correlaton", IEEE Tran. on Systems, Man and Cybernetics, 65-70, 1976.

[86] Gersting,J.L., Mathematical structures for computer science, 2nd ed., W.H.Freeman and Co., New York, 1987.

[87] Ghaffary,B., "Review of image matching techniques", SPIE Vol.596, Architectures and Algorithms for Digital Image Processing, 164-172, 1986.

[88] Gitman,I., "A parameter-free clustering model", Pattern Recognition, Vol.4, 307-315, 1972.

[89] Goldberg,M., and Shlien,S., "A clustering scheme for multispectral images", IEEE Trans. on Systems, Man and Cybernetics, Vol.SMC-8, No.2, 86-92, 1978.

[90] Gonzalez,R.C., and Wintz,P., Digital image processing, Addison-Wesley, Reading Massachusetts, 1977.

[91] Goodman,A.H., and Henderson-Sellers,A., "Cloud detection and analysis: A review of recent progress", Atmospheric Research, 21, 203-228, 1988.

[92] Gool,L.V., Dewaele,P., and Oosterlinck,A., "Survey: Texture analysis anno 1983, Computer vision, Graphics, and Image processing", Vol.29, 336-357, 1984.

[93] Goshtasby,A., Stockman,G.C., and Page,C.V., "A region-based approach to digital image registration with subpixel accuracy", IEEE Transactions on Geoscience and Remote Sensing, Vol.GE-24, No.3, 390-399, 1986.

[94] Gowda,K.C., "A feature reduction and unsupervised classification algorithm for multispectral data", Pattern Recognition, Vol.17, No.6, 667-676, 1984.

[95] Gowda, K.C., and Krishna, G., "Agglomerative clustering using the concept of mutual nearest neighbourhood", Pattern Recognition, vol.10, 105-112, 1978.

[96] Green,R., Hughes,G., Novak,C., and Schreitz,R., "The automatic extraction of wind estimates from VISSR data", NOAA technical memoradum NESS 64, U.S.DEPT. of COMMERCE, WASHINGTON, D.C. BRISTOR, C.L., (ED), 94-110, 1975.

[97] Haass,U.L., and Brubaker,T.A., "Estimation of cloud motion from satellite pictures", International conference on Acoustics, Speech and Signal processing. 422-425, April, 1980.

[98] Hall,D.J., "An adaptive process for tracking clouds from satellite images", Image Science Mathematics, Symposium, California, 1976, in Western Periodicals Corp. North Holland, 118-122, 1977.

[99] Hall,D.J., Endlich,R.M., Wolf,D.E., and Brain,A.E., "Objective methods for registering landmarks and determining cloud motions from satellite data", IEEE Trans. on Computers, 768-776, July 1972.

[100] Hall,E.L., Davies,D.L., and Casey,M.E., "The selection of critical subsets for signal, image, and scene matching", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-2, No.4, 313-322, 1980.

[101] Halpern,D., "Surface wind measurements and low-level cloud motion vectors near the intertropical convergence zone in the central pacific ocean from November 1977 to March 1978", Monthly Weather Review, Vol.107, 1525-1534, 1979.

[102] Halpern,D., "Comparison of low-level cloud motion vectors and moored buoy winds", Journal of Applied Meteorology, Vol.17, 1866-1871, 1978.

[103] Hand, D.J., Discrmination and Classification, John Wiley and Sons, Chichester, 1981.

[104] Hartigan, J.A., Clustering Algorithms, John Wiley and Sons, New York, 1975.

[105] Haralick,R.M., "Statistical and structural approaches to texture", Proc. of the IEEE, Vol.67, No.5, 786-803, 1979.

[106] Haralick,R.M., Shanmugam,K., and Dinstein,I., "Textural features for image classification", IEEE Trans. on Systems, Man, and Cybernetics, Vol.SMC-3, No.6, 610-621, 1973.

[107] Haralick,R.M., and Shapiro,L.G., "Image Segmentation Techniques", Computer Vision, Graphics, and Image Processing, 29, 100-132, 1985.

[108] Haralick,R.M., and Kelly,G.L., "Pattern recognition with measurement space and spatial clustering for multiple images", Proc. IEEE Vol.57, No.4, 654-665, 1969.

[109] Haralick, R.M., and Dinstein, I., "An iterative clustering procedure", IEEE Trans. on Systems, Man, and Cybernetics, Vol.SMC-1, No.3, 275-289, 1971.

[110] Haralick,R.M., and Dinstein, I., "A spatial clustering procedure for multi-image data", IEEE Trans. on Circuits and Systems, Vol.CAS-22, No.5, 440-450, 1975.

[111] Hasselblad,V., "Estimation of parameters for a mixture of normal distributions", Technometrics, 8, 431-444, 1966.

[112] Hasler,A.F., Shenk,W.E., and Skillman,W.C., "Wind estimates from cloud motions — Phase I of an in situ aircraft verification experiment", J. Appl. Meteor., 15, 10-15, 1976.

[113] Hasler,A.F., Shenk,W.E., and Skillman,W.C., "Wind estimates from cloud motions: preliminary results of Phases I, II, and III of an in situ aircraft verification experiment", J. Appl. Meteor., 16, 812-815, 1977.

[114] Hasler,A.F., Skillman,W.C., and Shenk,W.E., "In situ aircraft verification of the quality of satellite cloud winds over oceanic regions", Vol.18, 1481-1489, 1979.

[115] Helmuth Spath, Cluster Analysis Algorithms for data reduction and classification of objects, Ellis Horwood Ltd., Chichester, 1980.

[116] Henderson-Sellers, A., Satellite Sensing of a Cloudy Atmosphere: Observing the third planet, Taylor and Francis, London and Philadelphia, 1984.

[117] Hoffman,J., "Cloud motion wind retrieval in multilayered areas", Submitted to J.Geophysics Research. 1990.

[118] Hubert,L.F., "Wind derivation from geostationary satellites", in Quantitative Meteorological Data from Satellites, World Meteorological Organization Technical Note No.66, edited by Winston,J.S., 1979, 33-59.

[119] Hubert,L.F., and Whitney,L.F.Jr., "Wind estimation from geostationary satellite pictures", Monthly Weather Review, 99, 665-672, 1971.

[120] Hubert,L.J., "Some applications of graph theory to clustering", Psychometrika, Vol.39, No.3, 283-309, 1974.

[121] Hughes,G., Novak,C., and Schreitz, R., "Automatic techniques for the detection and displacement measurement of selected cloud imagery observed in geostationary satellite data", Proc. 8th Annual Automatic Imagery Pattern Recognition Symposium, Graithersburg, MD, 81-90, 1978.

[122] Ichida,K., and Kiyono,T., "Estimation of a probability density function of very many variables", IEEE Trans. on Systems, Man and Cybernetics, 463-466, July, 1975.

[123] Ince,F., "The application of the coalescence clustering algorithm to remotely sensed multispectral data", Pattern Recognition, Vol.14, No.1, 121-130, 1981.

[124] Izawa,T., and Fujita,T., "Relationship between observed winds and cloud velocities determined from pictures obtained by the ESSA III, ESSA V and ATS I satellites", SPACE RESEARCH IX - NORTH HOLLAND PUBLISHING COMP., AMSTERDAM, 571-579, 1969.

[125] Jain,A.K., Fundamentals of digital image processing, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1989.

[126] Jain,A.K., and Dubes,R.C., Algorithms for clustering data, Prentice-Hall, Englewood Cliffs, N.J., 1988.

[127] Jain,J.R., and Jain,A.K., "Displacement measurement and its application in interframe image coding", IEEE Transactions on Communications, Vol.COM-29, No.12, 1981, 1799-1808.

[128] Jain,A.K., and Moreau, J.V., "Bootstrap technique in cluster analysis", Pattern Recognition, Vol.20, No.5, 547-568, 1987.

[129] Jain,A.K., Smith,S.P., and Backer,E., "Segmentation of Muscle Cell Pictures: A Preliminary Study", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.PAMI-2, No.3, 232-242, 1980.

[130] Jarvis,R.A., "Shared near neighbour maximal spanning trees for cluster analysis", Proc. 4th Int. Joint Conf. on Pattern Recognition, Kyoto University, Kyoto, 308-313, 1978.

[131] Jarvis,R.A., and Patrick,E.A., "Clustering using a similarity measure based on shared near neighbors", IEEE Trans. on Computers, Vol.C-22, No.11, 1025-1034, 1973.

[132] Jayroe, R.R.Jr., "Unsupervised spatial clustering with spectral discrimination", NASA Technical Note, NASA TN D-7312, NASA, Washington,D.C., May, 1973.

[133] Jenson,S.K., Loveland,T.R., and Bryant,J., "Evaluation of amoeba: a spectral-spatial classification method", Journal of Applied Photographic Engineering, 159-162, Vol.8, 1982.

[134] Jensen,J.R., Introductory digital image processing, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1986.

[135] Johnston,B., Bailey,T., and Dubes,R., "A Variation on a Nonparametric Clustering Method", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.PAMI-1, No.4, OCTOBER 1979, 400-408.

[136] Jones,D.S., Elementary information theory, Clarendon press, Oxford, 1979.

[137] Karlsson,K., "Development of an operational cloud classification model", Int.J.Remote Sensing, Vol.10, No.4, 687-693, 1989.

[138] Kashef,B.G., and Sawenuk,A.A., "A survey of new techniques for image registration and mapping", Applications of Digital Image Processing VI. Proc.SPIE (SAN DIEGO, CA), 222-239, 1982.

[139] Kauth,R.J., Pentland,A.P., and Thomas,G.S., "BLOB: An Unsupervised Clustering Approach to Spatial Preprocessing of MSS Imagery", 11th Symp. on Remote Sensing of Environment, Vol.2 1309-1317, 1977.

[140] Kettig,R.L., and Landgrebe,D.A., "Classification of multispectral image data by extraction and classification of homogeneous objects", IEEE Trans. on Geoscience Electronics, Vol.GE-14, No.1, 19-26, 1976.

[141] Key,J.R., Maslanik,J.A., and Barry,R.G., "Cloud classification from satellite data using a fuzzy sets algorithm: A polar example", International Journal of Remote Sensing, Vol.10, No.12, 1823-1842, 1989.

[142] Kittler,J., "A locally sensitive method for cluster analysis", Pattern Recognition, Vol.8, 23-33, 1976.

[143] Kittler,J., and Illingworth,J., "Minimum error thresholding", Pattern Recognition, Vol.19, No.1, 41-47, 1986.

[144] Kittler,J., and Pairman,D., "Contextual pattern recognition applied to cloud detection and identification", IEEE Trans. on Geoscience and Remote Sensing, Vol.GE-23, No.6, 855-863, 1985a.

[145] Kittler,J., and Pairman,D., "Segmentation of multispectral imagery using iterative clustering", Proc. of the 4th Scandinavian conference on image analysis, TRONDHEIN, Vol.I, 39-49, 1985b.

[146] Kittler,J., and Pairman,D., "Optimality of reassignment rules in dynamic clustering", Pattern Recognition, Vol.21, No.2, 169-174, 1988.

[147] Koontz,W.L.G., and Fukunaga,K., "A nonparametric valley-seeking technique for cluster analysis", IEEE Trans. on Computers, Vol.C-21, No.2, 171-178, 1972.

[148] Koontz,W.L.G., Narendra,P.M., and Fukunaga,K., "A branch and bound clustering algorithm", IEEE Trans. on Computers, Vol.C-24, No.9, 908-915, 1975.

[149] Koontz,W.L.G., Narendra,P.M., and Fukunaga,K., "A graph-theoretic approach to nonparametric cluster analysis", IEEE Transaction on Computers, Vol.C-25, No.9, 936-944, 1976.

[150] Kruskal, J.B.,Jr., "On the shortest spanning tree of a graph", Proc. of the Amer. Math. Soc., 7, 48-49, 1956.

[151] Kuglin,C.D., and Eppler,W.G., "Map-matching techniques for use with multispectral/multitemporal data, Image processing for missile guidance", SPIE Proc.238, 146-155, 1980.

[152] Kuglin,C.D., Blumenthal,A.F., and Pearson,J.J., "Map-matching techniques for terminal guidance using Fourier phase information", SPIE Vol.186, Digital Processing of Aerial Images, 21-29, 1979.

[153] Landgrebe, D.A., "The Development of a Spectral-Spatial Classifier for Earth Observational Data", Pattern Recognition, Vol.12, 165-175, 1980.

153a > Lau,K.S., and G. Wade, "Spatial-spectral clustering using recursive spanning trees", IEE Proc I, Vol. 138, No. 4, 1991 pp 232-238

[154] Lee,D.H., "Level assignment in the assimilation of cloud motion vectors", Monthly Weather Review, Vol.107, 1055-1074, 1979.

[155] Leese,J.A., and Novak,C.S., "An automated technique for obtaining cloud motion from geosynchronous satellite data using cross-correlation", Journal of Applied Meteorology, Vol.10, 119-132, Feb., 1971.

[156] Leese,J.A., Novak,C.S., and Taylor,V.R., "The determination of cloud pattern motions form geosynchronous satellite image data", Pattern Recognition, Vol.2, 279-292, 1970.

[157] Leese,J.A., Novak,C.S., and Clark,B.B., "An automated technique for obtaining cloud motion from geosynchronous satellite data using cross correlation", Journal of Applied Meteorology, Vol.10, 118-132, 1971.

[158] Lindeman,R.H., Merenda, P.F., and Gold,R.Z., Introduction to Bivariate and Multivariate Analysis, Scott, Foresman and Company, 1980.

[159] Lo,R., and Parikh,J.A., "Application of Fourier transforms to cloud movement estimation from satellite photographs", Technical Report 242, Computer Science Center, Univ. of Maryland, 1973.

[160] Lumelsky, V.L., "A combined algorithm for weighting the variables and clustering in the clustering problem", Pattern Recognition, vol.15, No.2, 53-60, 1982.

[161] Lunnon,R.W., and Lowe,D.A., "Spatial scale dependency of errors in satellite cloud track winds", COSPAR Id No MA 1.1.7, 26th June 1990.

[162] MacQueen, J.B., "Some methods for classification and analysis of multivariate observations", Proc. of 5th Berkeley Symposium, 1, 281-297, 1967.

[163] MacRae, D.J., "MIKCA:A FORTRAN IV iterative k-means cluster analysis program", Behavioral Sci., 16, No.4, 423-424, 1971.

[164] Maddox,R.A., and Haar,H.V., "Covariance analyses of satellite-derived mesoscale wind fields", Journal of Applied Meteorology, Vol.18, 1327-1334, 1979.

[165] Mandel,I.D., and Chernyi,L.B., "Experimental comparison of cluster analysis of algorithms", Automation and Remote Control(English Translation of Automatika i Telemekhanika) Journal of the Institution of Electronics and Telecommunication Engineers, Vol.34, Part 3, 258-265, 1988.

[166] Mandel,I.D., and Mirkin,B.G., "Cluster analysis and related problems (Brief review of basic trends)", Soviet Journal of Automation and Information Sciences :(English Translation of Avtomatyoka), Vol.20, Part 3, 74-84, 1987.

[167] Mausel,P.W., Kramber,W.J., and Lee,J.K., "Optimum band selection for supervised classification of multispectral data", Photogrammetric Engineering and Remote Sensing, Vol.56, No.1, 55-60, 1990.

[168] Menzel,W.P., Smith,W.L., and Stewart,T.R., "Improved cloud motion wind vector and Altitude Assignment using VAS", Journal of Climate and Applied Meteorology, Vol.22, 377-384, 1983.

[169] METEOSAT SYSTEM GUIDE, Vol.5 - Meteorological Products, 1980, European Space Agency, ESOC, Darmstadt, W.Germany.

[170] Michael,M., and Lin ,W.C., "Experimental study of information measure and inter-intra class distance ratios on feature selection and orderings", IEEE Trans. on Systems, Man, and Cybernetics, Vol.SMC-3, No.2, 172-181, 1973.

[171] Milgram,M., Dubuisson,B., and Vachon,B., "A computationally efficient clustering algorithm", IEEE Trans. on Systems, Man, and Cybernetics, 99-104, Feb., 1977.

[172] Mittchum,G.T., "A bias in the satellite observed low-level cloud motion winds over the central tropical pacific", Journal of Geophysical Research, Vol.29, No.C4, 3861-3865, 1987.

[173] Mitchell,O.R., and Carlton,S.G., "Image Segmentation Using a Local Extrema Texture Measure", Pattern Recognition, Vol.10, 205-210, 1978.

[174] Mizoguchi,R., and Shimura,M., "A nonparametric algorithm for detecting clusters using hierarchical structure", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-2, No.4, 292-300, 1980.

[175] Morris,O.J., Lee,M., and Constantinides,A.G., "Graph theory for image analysis: an approach based on the shortest spanning tree", IEE Proc., Vol.133, Pt. F, No.2, 146-152, 1986.

[176] Mucciardi,A.N., and Gose,E.E., "An algorithm for automatic clustering in N-dimensional spaces using hyperellipsoidal cells", Proc. IEEE Systems Science and Cybernetics Conf. 201-209, 1970.

[177] Nack,M.L., "Rectification and registration of digital images and the effect of cloud detection", IEEE, Machine Processing of Remotely Sensed Data Symposium, 12-23, 1977.

[178] Nagy, G., and Tolaba, J., "Nonsupervised crop classification through airborne multispectral observations", IBM J. RES. DEVELOP., 16, 2, 138-153, 1972.

[179] Narendra,P.M., and Goldberg,M., "A non-parametric clustering scheme for LANDSAT", Pattern Recognition, Vol.9, 207-215, 1977.

[180] Narendra,P.M., and Goldberg,M., "Image segmentation with directed trees", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-2, No.2, 185-191, 1980.

[181] Ney,H., "Dynamic programming as a technique for pattern recognition", Proc. 6th Int. Conf. on Pattern Recognition, Munich, IEEE New York, CH 1801-0, 1119-1125, 1982.

[182] Novak,C., and Young,M., "The operational processing of wind estimates from cloud motions: past, present and future", Proc. of 11th International Symposium on Remote Sensing of the Environment, ANN ARBOR, MICHIGAN, 1589-1598, 1978.

[183] Pairman, D., "Image Processing in Geophysics", Jesus college, Oxford, D.Phil Thesis, Trinity Term, 1986.

[184] Pairman,D., and Kittler,J., "Clustering algorithms for use with images of clouds", International Journal of Remote Sensing, Vol.7, No.7, 855-866, 1986.

[185] Parikh,J.A., "An approach to selection of wind tracers from tropical maritime geosynchronous satellite cloud imagery", Computer Science Tec.Rep.Ser., Univ. of Maryland, TR-450 F-44620-72C-0062, March, 1976.

[186] Parikh,J.A., "Automatic wind velocity estimation from multispectral geosynchronous satellite data: a proposal", Computer Science Tec.Rep.Ser., Univ. of Maryland, TR-328 F-44620-72C-0062, Sept,1974.

[187] Parikh,J.A., "Cloud classification from visible and infrared SMS-1 data", Remote Sens. of Environ., No.7, 85-92, 1978.

[188] Parikh,J.A., "A comparative study of cloud classification techniques", Remote Sensing of Environment, 6, 67-81, 1977.

[189] Parikh,J.A., and Rosenfeld,A., "Automated segmentation and classification of infrared meteorological satellite data", IEEE Transaction System, Man and Cybernetic, SMC-8, 736-743, 1978.

[190] Parzen,E., "On estimation of a probability density function and mode", Ann.Math.Stat., Vol.33, 1065-1076, 1962.

[191] Pelleg,S., Naor,J., Hartley,R., and Avnir,D., "Multiple resolution texture analysis and classification", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-6, No.4, 518-523, 1984.

[192] Peslen,C.A., Koch,S.E., and Uccellini,L.W., "The effect of the arbitrary level assignment of satellite cloud motion wind vectors on wind analyses in the pre-thunderstorm environment", Journal of Climate and Applied Meteorology, Vol.25, 615-632, 1986.

[193] Phulpin,T, Derrien,M., and Brard,A., "A two-dimensional histogram procedure to analyze cloud cover form NOAA satellite high-resolution imagery", Journal of Climate and Applied Meteorology, Vol.22, 1333-1345, 1983.

[194] Platt,C.M.R., "On the bispectral method for cloud parameter determination from satellite VISSR data: Separating broken cloud and semitransparent cloud", Journal of Climate and Applied Meteorology, Vol.22, 429-439, 1983.

[195] Pratt,W.K., "Correlation techniques of image registration", IEEE Trans. on Aerospace and Electronic systems, Vol.AES-10, No.3, 353-358, 1974.

[196] Prim, R.C., "Shortest connection networks and some generalizations", Bell System Technical J. 36, 1389-1401, 1957.

[197] Ray,K.S., and Majumder,D.D., "An intelligent search for clustering", J. Inst Electronics and Telecom. Engrs., Vol.34, No.3, 1988.

[198] Rayn,H.F., "The information content measure as a performance criterion for feature selection", IEEE Proc. 7th Symp. Adaptive Processes, 2-c-1 - 2-c-11, 1968.

[199] Reynolds,D.W., and Vonder Haar,T.H., "A bispectral method for cloud parameter determination", Monthly Weather Review, Vol.105, 446-457, 1977.

[200] Robertson, T.V., "Extraction and classification of objects in multispectral images", Conf. on Machine Processing of Remotely Sensed Data, Purdue Uni., West Lafayette, section 38, 27-34, 1973.

[201] Rosenfeld,A., and Vanderbrug,G.J., "Coarse-fine template matching", IEEE Tran. on Systems,Man and Cybernetics, 104-107, Feb., 1977.

[202] Rosenfeld,A., "Image pattern recognition", Proceedings of The IEEE, Vol.69, No.5, 596-605, 1981.

[203] Rosenfeld, A., and Kak, A.C., Digital Picture Processing, Vol.1, 2, 2nd ed., Academic Press, INC., New York, 1982.

[204] Rossow,W.B., "Measuring cloud properties from space: A Review, Journal of Climate", Vol.2, 201-213, 1989.

[205] Rossow,W.B., Mosher,F., Kinsella,E., Arking,A., Desbois,M., Harrison,E., Minnis,P., Ruprecht,E., Seze,G., Simmer,C., and Smith,E., "ISCCP cloud algorithm intercomparison", Journal of Climate and Applied Meteorology, Vol.24, No.9, 877-903, 1985.

[206] Sadler,J.C., and Kilonsky,B.J., "Deriving surface winds from satellite observations of low-level cloud motions", Journal of Climate and Applied Meteorology, Vol.24, 758-769, 1985.

[207] Saunders,R.W., and Kriebel,K.T., "An improved method for detecting clear sky and cloudy radiances from AVHRR data", Int. J. Remote Sensing, Vol.9, No.1, 123-150, 1988.

[208] Schachter,B.J., Davis,L.S., and Rosenfeld,A., "Some Experiments in Image Segmentation by Clustering of Local Feature Values", Pattern Recognition, Vol.11, 19-28, 1979.

[209] Schmetz,J., "An atmospheric-correction scheme for operational application to Meteosat infrared measurements", ESA Journal, Vol.10, 145-159, 1986.

[210] Schmetz,J., and Nuret,M., "Automatic tracking of high-level clouds in Meteosat infrared images with a radiance windowing technique", European Space Agency Journal, 1987, Vol.11, 275-286.

[211] Schmetz,J., and Holmlund,K., "Operational cloud motion winds from Meteosat and the use of cirrus clouds as tracers", COSPAR 1990.

[212] Scott,A.J., and Symons,M.J., "Clustering Methods Based on Likelihood Ratio Criteria", Biometrics 27, 387-397, 1971.

[213] Schwartzmann,D.H., and Vidal,J.J., "An algorithm for determining the topological dimensionality of point clusters", IEEE Trans. on Computers, Vol.C-24, No.12, 1175-1182, 1975.

[214] Sebestyen,G. and Edie, J., "An algorithm for non-parametric pattern recognition", IEEE Transactions on Electronic Computers, EC-15, 908-915, 1966.

[215] Seddon, A.M., "The application of scene analysis techniques to automatic classification of atmospheric data from multispectral satellite imagery", PhD thesis, University College London, 1983.

[216] Seddon,A.M., and Hunt,G.E., "Segmentation of clouds using cluster analysis", International Journal of Remote Sensing, Vol.6, No.5, 717-731, 1985.

[217] Seze, G., and Desbois, M., "Cloud cover analysis from satellite imagery using spatial and temporal characteristics of the data", Journal of Climate, Vol.26, 287-303, 1987.

[218] Shlien,S., and Smith,A., "A rapid method to generate spectral theme classification of LANDSAT imagery", Remote Sensing of Environment, 4, 67-77, 1975.

[219] Sheffield,C., "Selecting band combinations from multispectral data", Photogrammetric Engineering and Remote Sensing, Vol.51, NO.6, 681-687, 1985.

[220] Shenk,W.E., and Kreins,E.R., "A comparison between observed winds and cloud motions derived from satellite infrared measurements", Journal of Applied Meteorology, Vol.9, 703-710, 1970.

[221] Shenk,W.E., and Curren,R.J., "A multi-spectral method for estimating cirrus cloud top heights", Journal of Applied Meteorology, Vol.12, 1213-1216, 1973.

[222] Shen,H.C., and Wong,A.K.C., "Generalized texture representation and metric", Computer vision, Graphics, and Image Processing, Vol.23, 187-206, 1983.

[223] Sibson,R., "Order invariance methods for data analysis (with discussion)", Journal Royal Stat. Soc. B, 34, 311-349, 1972.

[224] Simon,J.C., "Clustering and digital image analysis", Inst. Phys. Conf. Ser. No.44, 20-39.

[225] Smith,E.A., and Phillips,D.R., "Automated cloud tracking using precisely aligned digital ATS pictures", IEEE Trans. on Computers, Vol.C-21, No.7, 715-729, 1972.

[226] Sokal, R.R., and Michener, C.D., "The statistical method for evaluating systematic relationships", Univ. Kansas Sc. Bull., 38, 1409-1438, 1958.

[227] Spann,M., and Wilson,R., "A Quad-tree Approach to Image Segmentation which Combines Statistical and Spatial Information", Pattern Recognition, Vol.18, Nos 3/4, 257-269, 1985.

[228] Specht,D.F., "Generation of polynomial discriminant functions for pattern recognition", IEEE Trans. on Electronic Computers, Vol.EC-16, No.3, 308-319, 1967.

[229] Stockman,G., Kopstein,S., and Benett,S., "Matching images to models for registration and object detection vis clustering", IEEE Tran. on Pattern Analysis and Machine Intelligence, Vol.PAMI-4, No.3, 229-241, 1982.

[230] Stromberg,W.D., and Farr,T.G., "A Fourier-based textural feature extraction procedure", IEEE Trans. on Geoscience and Remote Sensing, Vol.GE-24, No.5, 722-731, 1986.

[231] Sun,C., and Wee,W., "Neighboring gray level dependence matrix for texture classification", Computer vision,Graphics, and Image processing, Vol.23, 341-352, 1983.

[232] Suomi,V.E., ed., Wind determination from geostationary satellites in Meteorological observations from space. Their contribution to the first GARP global experiment. Philadelphia Symposium Proc. 188-250, 1976.

[233] Suomi,V.E., Fox,R., Limaye,S.S., and Smith,W.L., "McIDAS III: A modern interactive data access and analysis system", Journal of Climate and Applied Meteorology, Vol.22, 767-778, 1983.

[234] Svedlow,M, McGillem,C.D., and Anuta,P.E., "Image registration:Similarity measure and preprocessing method comparisons", IEEE Trans. on Aerospace and Electronic systems, Vol.AES-14, No.1, 141-149. 1978.

[235] Swain,P.H., Vardeman,S.B., and Tilton,J.C., "Contextual classification of multispectral image data", Pattern Recognition, Vol.13, No.6, 429-441, 1981.

[236] Symons,M.J., "Clustering criteria and multivariate normal mixtures", Biometrics, 37, 35-43, 1981.

[237] Szejwach,G., "Determination of semi-transparent cirrus cloud temperature from infrared radiances: application to METEOSAT", Journal of Applied Meteorology, Vol.21, 384-393, 1982.

[238] Tanimoto,S.L., "Regular hierarchical image and processing structures in machine vision", in Computer Vision System. New York: Academic, 1978, pp.165-174.

[239] Tanimoto, S., and Klinger,A., Structured Computer Vision. New York: Academic, 1980.

[240] Torn,A., "Cluster analysis using seed points", Proc. 3rd Int. Joint Conf. on Pattern Recognition, Colorado, CA, 394-398, 1976.

[241] Torn,A., "Cluster analysis using seed points and density determined hyperspheres as an aid to global optimization", IEEE Trans. on Systems, Man, and Cybernetics, Vol.SMC-7, No.8, 610-616, 1977.

[242] Townshend,J.R.G., and Justice,C.O., "Unsupervised classification of MSS Landsat data for mapping spatially complex vegetation", International Journal of Remote Sensing, Vol.1, No.2, 105-120, 1980.

[243] Tou,J.T., and Gonzalez,R.C., Pattern recognition principles, Addison-Wesley, Reading, Massachusetts, 1974.

[244] Tsonis,A.A., "On the separability of various classes form the GOES satellite data", Journal of Climate and Applied Meteorology, Vol.23, No.10, 1393-1410, 1984.

[245] Turner, J., and Warren,D.E., "Cloud track winds in the polar regions from sequences of AVHRR images", International Journal of Remote Sensing, 1989, Vol.10, Nos.4 and 5, 695-703.

[246] Umesh,R.M., "A technique for cluster formation", Pattern Recognition, Vol.21, No.4, 393-400, 1988.

[247] Unser,M., "Sum and difference histograms for texture classification", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-8., No.1, 118-125, 1986.

[248] Urquhart,R., "Graph theoretical clustering based on limited neighbourhood sets", Pattern Recognition, Vol.15, No.3, 173-187, 1982.

[249] Vonder Haar,T.H., and Reynolds,D.W., "A bi-spectral method for inferring cloud amount and cloud-top temperature using satellite data", 6th conference on aerospace and aeronautical meteorology, Amer. Met. Soc., 190-193, 1974.

[250] Ward, Jr., J.H., "Hierarchical grouping to optimise an objective function", J. Amer. Statist. Assoc. 58, No.301, 236-244, 1963.

[251] Warnecke,G., Zick,C., Carus,B., Doring,R., Eriksson,A., and Voellger,C., "Information extraction from meteorological satellite image sequences", R.A.Vaughan(ed.), Remote Sensing Applications in Meteorology and Climatology, 259-283, D.Reidel Publishing company, 1987.

[252] Wharton,S.W., "A generalised histogram clustering scheme for multidimensional image data", Pattern Recognition, Vol.16, No.2, 193-199, 1983.

[253] Wharton,S.W., "A Contextual Classification Method for Recognizing Land Use Patterns in High Resolution Remotely Sensed Data", Pattern Recognition, Vol.15, No.4, 317-324, 1982.

[254] Wharton,S.W., and Turner,B.J., "ICAP:An interactive cluster analysis procedure for analyzing remotely sensed data", Remote Sensing of Environment, 11, 279-293, 1981.

[255] Widger,W.K., Jr., and Touart, C.N., "Utilization of satellite observations in weather analysis and forecasting", Bull. Amer. Meteor. Soc., 38,.521-533, 1957.

[256] Wilson,G.S., "Automated mesoscale wind fields derived from GOES satellite imagery", American Meteorology Society Conference on Satellite/Remote Sensing Application. 1984, 164-171.

[257] Wilson,T., and Houghton,D., "Mesoscale wind fields for a severe storm situation determined from SMS cloud observations", Monthly Weather Review,1198-1209, 1979.

[258] Wilks,S.S., Mathematical statistics, Wiley, London, 1963.

[259] Winston,J.S. ed., Quantitative meteorological data from satellites, in World Meteorological Organization, Technical note No.166, Chapter 2., 1979.

[260] Wishart, D., "An algorithm for hierarchical classifications", Biometrics, 25, 165-170, 1969.

[261] WMO. 1956. "International cloud atlas", preface to the 1939 edition Vol.1 World Meteorological Office, Paris.

[262] Wolfe,J.H., "Pattern clustering by multivariate mixture analysis", Multivariate Behav. Res., 5, 329-350, 1970.

[263] Wolf,D.E., Hall,D.J., and Endlich,R.M., "Experiments in automatic cloud tracking using SMS-GOES data", Journal of Applied Meteorology, Vol.16, 1219-1230, 1977.

[264] Wolf,R., Meteosat high resolution image dissemination, Meteosat System Guide, Vol.9, Issue 4, August, 1984, ESA.

[265] Wong,R.Y., "Sequential Scene Matching Using Edge Features", IEEE Tran. on Aerospace and Electronic systems, Vol.AES-14,No.1, 128-140, 1978.

[266] Wong,R.Y. and Hall,E.L., "Scene matching with invariant moments", Computer Graphics and Image Processing 8, 16-24, 1978.

[267] Wong,R.Y., and Hall,E.L., "Sequential Hierarchical Scene Matching", IEEE Tran. on Computers, Vol.C-27, No.4, 359-366, 1978.

[268] Woodroffe, A., The Operational Use of Satellite Imagery in the Central Forecasting Office, Braknell, Preprints for a Workshop on Satellite and Radar Imagery Interpretation, Meteorological Office College, England, 3-20, 1987.

[269] Wright,W.E., "Gravitational clustering", Pattern Recognition, Vol.9, 151-166, 1977.

[270] Wright,W.E., "A Formalization of Cluster Analysis", Pattern Recognition, Vol.5, 273-282, 1973.

[271] Wylie,D.P., Hinton,B.B., and Millett,K.M., "A comparison of three satellite-based methods for estimating surface winds over oceans", Journal of Applied Meteorology, Vol.20, 439-449, 1981.

[272] Yau,S.S., and Chang,S.C., "A direct method for cluster analysis", Pattern Recognition, Vol.7, 215-224, 1975.

[273] Zahn,C.T., "Graph-theoretical methods for detecting and describing gestalt clusters", IEEE Trans. on Computers, Vol.C-20, No.1, 68-86, 1971.

[274] Zucker,S.W., "Region Growing: Childhood and Adolescence", Computer Graphics and Image Processing, 5, 382-399, 1976.

Appendix A

Maximum Likelihood Estimator

Suppose the set $\mathbf{X}$ contains n samples, $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Then since the samples were drawn independently,

$$p(\mathbf{X} \mid \theta) = \prod_{k=1}^{n} p(\mathbf{x}_k \mid \theta) \tag{A.1}$$

$p(\mathbf{X} \mid \theta)$ is called the likelihood function of θ with respect to the set of samples. The maximum likelihood estimator of θ is, by definition, the value $\hat{\theta}$ that maximizes $p(\mathbf{X} \mid \theta)$. The classical approach is to differentiate $p(x \mid \theta)$ with respect to θ, equate $\partial p(\mathbf{X} \mid \theta)/\partial\theta = 0$, and solve for $\hat{\theta}$.

We want to find the maximum likelihood estimate of the mean μ and covariance matrix Σ of a sample set $\mathbf{X}$ form a multi-variate normal pdf, so $\theta = (\mu, \Sigma)$, we have

$$p(\mathbf{x} \mid \theta) = (2\pi)^{-d/2}|\Sigma|^{-1/2} \exp\left[-\tfrac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)\right] \tag{A.2}$$

$$\begin{aligned} p(\mathbf{X} \mid \theta) &= (2\pi)^{-dn/2}|\Sigma|^{-n/2} \exp\left[-\tfrac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i - \mu)^T \Sigma^{-1}(\mathbf{x}_i - \mu)\right] \\ &= (2\pi)^{-dn/2}|\Sigma|^{-n/2} \exp[-\frac{1}{2}\mathrm{tr}\Sigma^{-1}A] \end{aligned} \tag{A.3}$$

$$\text{where } A = \sum_{j=1}^{n}(\mathbf{x}_j - \mu)(\mathbf{x}_j - \mu)^T$$

Let $\mathbf{m}_n$ denotes the samples mean vector where $\mathbf{m}_n \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$

$$\text{But } A = \sum_{j=1}^{n}[(\mathbf{x}_j - \mathbf{m}_n) + (\mathbf{m}_n - \mu)][(\mathbf{x}_j - \mathbf{m}_n) + (\mathbf{m}_n - \mu)]^T$$

$$= \sum_{j=1}^{n}(\mathbf{x}_j - \mathbf{m}_n)(\mathbf{x}_j - \mathbf{m}_n)^T + n(\mathbf{m}_n - \mu)(\mathbf{m}_n - \mu)^T \qquad \text{(A.4)}$$

$$\text{Since } \sum_{j=1}^{n}(\mathbf{x}_j - \mathbf{m}_n)(\mathbf{m}_n - \mathbf{x}_j)^T = \sum_{j=1}^{n}(\mathbf{m}_n - \mu)(\mathbf{x}_j - \mathbf{m}_n)^T = 0$$

Substitute A.4 into A.3 and let $V = \sum_{j=1}^{n}(\mathbf{x}_j - \mathbf{m}_n)(\mathbf{x}_j - \mathbf{m}_n)^T$ be the sample covariance matrix, we have

$$p(\mathbf{X} \mid \theta) = (2\pi)^{-dn/2}|\Sigma|^{-n/2}\exp\{-\frac{1}{2}\text{tr}\Sigma^{-1}[V + n(\mathbf{m}_n - \mu)(\mathbf{m}_n - \mu)^T]\} \qquad \text{(A.5)}$$

It is more convenient to work with the logarithm of the likelihood function, than with the likelihood function itself.

$$\begin{aligned} L &= \log p(\mathbf{X} \mid \theta) \\ &= \frac{n}{2}\log|\Sigma|^{-1} - \frac{1}{2}\text{tr}\Sigma^{-1}V \\ &\quad -\frac{n}{2}\text{tr}\Sigma^{-1}(\mathbf{m}_n - \mu)(\mathbf{m}_n - \mu)^T + \frac{dn}{2}\log 2\pi \end{aligned} \qquad \text{(A.6)}$$

Differentiating with respect to μ gives

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{n}{2}\frac{\partial}{\partial \mu}\text{tr}\Sigma^{-1}(\mathbf{m}_n - \mu)(\mathbf{m}_n - \mu)^T \\ &= -\frac{n}{2}2\Sigma^{-1}(\mu - \mathbf{m}_n) \end{aligned} \qquad \text{(A.7)}$$

Multiply by Σ and rearranging, we obtain

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \qquad \text{(A.8)}$$

Similarly, using the matrix identity

$$\frac{d}{dY}|Y| = |Y|Y^{-1}, \frac{d}{dY}\text{tr}(B^TY) = B$$

we have

$$\frac{\partial L}{\partial \Sigma^{-1}} = \frac{n}{2} - \frac{1}{2}A \tag{A.9}$$

Setting the derivative equal to zero and solving gives the estimator

$$\hat{\Sigma} = \frac{1}{n}\sum_{j=1}^{n}(\mathbf{x}_j - \hat{\mu})(\mathbf{x}_j - \hat{\mu})^T \tag{A.10}$$

Appendix B

Formula for Updating Gaussian Kernel Parameters for Post Transfer Advantage Rule.

The parameters which have to be updated after each reallocation of points are μ_i and Σ_i the mean and covariance matrix of the ith cluster respectively. Suppose that k copies of pattern $\mathbf{x}_l$ are transferred form cluster i to j with parameters μ_j and Σ_j (Kittler and Pairman 1988).

By definition

$$\mathbf{x}_l \in \text{ cluster } \omega_i$$

$$\begin{aligned}
\hat{\mu}_i &= \frac{1}{n_i - k}\left[\sum_{r=1}^{n_i} \mathbf{x}_r - k\mathbf{x}_l\right] \\
&= \frac{1}{n_i - k}[n_i\mu_i - k\mathbf{x}_l] \\
&= \mu_i + \frac{k}{n_i - k}(\mu_i - \mathbf{x}_l) \\
&= \mu_i + \Delta\mu_i
\end{aligned} \tag{B.1}$$

Similarly

$$\hat{\Sigma}_i = \frac{1}{n_i - k}\left[\sum_{r=1}^{n_i}(\mathbf{x}_r - \hat{\mu}_i)(\mathbf{x}_r - \hat{\mu}_i)^T - k(\mathbf{x}_r - \hat{\mu}_i)(\mathbf{x}_r - \hat{\mu}_i)^T\right] \tag{B.2}$$

Substitute eqn. B.1 into eqn. B.2 we obtain

$$\hat{\Sigma}_i = \frac{1}{n_i - k}\left[\sum_{r=1}^{n_i}(\mathbf{x}_r - \mu_i - \Delta\mu_i)(\mathbf{x}_r - \mu_i - \Delta\mu_i)^T - k(\mathbf{x}_l - \mu_i - \Delta\mu_i)(\mathbf{x}_l - \mu_i - \Delta\mu_i)^T\right] \quad \text{(B.3)}$$

Since $\sum_{r=1}^{n_i}(\mathbf{x}_r - \mu_i) = 0$

Eqn. B.3 simplifies to

$$\begin{aligned}\hat{\Sigma}_i &= \frac{1}{n_i - k}\left[n_i\Sigma_i + \Delta\mu_i \Delta \mu_i^T - \frac{n_i^2}{k}\Delta\mu_i \Delta \mu_i^T\right] \\ &= \frac{n_i}{n_i - k}\left[\Sigma_i - \frac{k}{n_i - k}(\mu_i - \mathbf{x}_l)(\mu_i - \mathbf{x}_l)^T\right] \quad \text{(B.4)}\end{aligned}$$

The updating formulas for cluster j can be obtained similarly.

$$\hat{\mu_j} = \mu_j + \frac{k}{n_j - k}(\mu_j - \mathbf{x}_l) \quad \text{(B.5)}$$

$$\hat{\Sigma_j} = \frac{n_j}{n_j + k}\left[\Sigma_j + \frac{k}{n_j + k}(\mu_j - \mathbf{x}_l)(\mu_j - \mathbf{x}_l)^T\right] \quad \text{(B.6)}$$

Appendix C

Efficient Algorithms for Constructing a Spanning Tree and Minimax Segmentation.

When graph is implemented on computer, it is stored using a data structure called linked list, and the traversal algorithm can be either Depth First Search or Breath First Search. These algorithm can be found in numerous text book (e.g. Gerstling 1982).

The construction of spanning tree and minimax segmentation in Chapter 5 can be constructed using graph theory algorithm proposed by Daskalaskis et al. (1988a, 1988b).

C.1 Efficient Implementation of Spanning Tree Algorithms

The complexity of the algorithm to construct the spanning tree is $O(n^{\frac{3}{2}})$ and therefore perform reasonably well even for large n, where n is the number of nodes in the graph. This is based on the Kruskal's algorithm but using a special data structure.

The spanning tree algorithm starts with n components and each components

is a binary tree data structure (Daskalakis et al. 1988a). A binary tree is a special kind of tree in which every node has at most two children. In a binary tree, each child of a node is designated as either the left child or the right child. The left child would consists of the nodes in this component and the right child of the links of the original graph incident to these nodes, which do not belong to the forest. So initially each left child consists of a single node and each right child of 8 or 4 edges of its nearest neighbours.

The spanning tree algorithm is:

Step 1. Initialise the data structures of the image graph described above.

Step 2. Arrange the edges in each structures in ascending order of weight.

Step 3. Create a list of pointer p, and point to the first edge in each structure.

Step 4. Save the edge with the lightest weight by searching all n pointers.

Step 5. Merge the binary trees which are joined by this edge and rearrange pointers.

Step 6. Remove edges in each tree which form cycles.

Step 7. Recompute the weight of the edges of the newly formed binary tree.

Step 8. Find the least weighted edge and label the rest in this tree to prevent them from selected.

Step 9. Rearrange pointer and point to the least weighted edge.

Step 10. If the number of saved links is less than $n - 1$ goto Step 4.

Step 11. Form the spanning tree from the saved links.

C.2 Efficient Implementation of Minimax Segmentation

A efficient implementation of minimax segmentation must avoid repeat computation of the intraset distance of a subtree. This is achieved by transforming the spanning tree into its directed counterpart (arborescense) by choosing arbitrary a node as its root. A branch b_i of the arborescense is represented by its parents node v_i whilst the root represents the whole arborescence.

The sum, the sum of squares of each variables and the number of nodes of each branch are attribute to its parents node. Therefore the intraset distance of a branch b_i can be computed by visiting node v_i. After this transformation the minimax segmentation can be implemented using algorithm described in Chapter 5.

Appendix D

Intraset Distance

In d-dimensional Euclidean space, the distance between two points $\mathbf{a}$ and $\mathbf{b}$ is given by (Tou and Gonzalez 1974 pp 248)

$$\begin{aligned} D(\mathbf{a},\mathbf{b}) &= ||\mathbf{a}-\mathbf{b}|| \\ &= \sqrt{(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})} \\ &= \sqrt{(\sum_{k=1}^{d}(\mathbf{a}_k-\mathbf{b}_k)^2} \end{aligned} \tag{D.1}$$

where $\mathbf{a}$ and $\mathbf{b}$ are d-dimensional vectors with the kth components equal to $\mathbf{a}_k$ and $\mathbf{b}_k$ respectively. The intraset distance for a set of pattern points $\{\mathbf{a}^i,\ i = 1,2,\ldots,K\}$ is given by

$$\overline{D^2(\{\mathbf{a}^j\},\{\mathbf{a}^i\})},\ i,j = 1,2,\ldots,K-1;\ \ i \neq j \tag{D.2}$$

define the square distance between $\mathbf{a}^j$ and $\mathbf{a}^i$ $i \neq j$

$$\begin{aligned} D^2(\mathbf{a}^j,\mathbf{a}^i) &= (\mathbf{a}^j-\mathbf{a}^i)^T(\mathbf{a}^j-\mathbf{a}^i) \\ &= \sum_{k=1}^{d}(\mathbf{a}_k^j-\mathbf{a}_k^i)^2 \end{aligned} \tag{D.3}$$

For fixed $\mathbf{a}^j$ and with $\mathbf{a}^i$ ranging over all of $K-1$ other points in the set $\{\mathbf{a}^i\}$, the partial average is

$$\overline{D^2(\mathbf{a}^j, \{\mathbf{a}^i\})} = \frac{1}{K-1} \sum_{i=1}^{K} \sum_{k=1}^{d} (\mathbf{a}_k^j - \mathbf{a}_k^i)^2 \tag{D.4}$$

It is noted that the contribution for $i = j$ is zero and may harmlessly be left in the expression. There are K terms, but only $K-1$ non-zero terms.

Following the same line of reasoning, we then take the average over all K points in set $\{\mathbf{a}^j\}$ to express the intraset distance as

$$\begin{aligned} \overline{D^2(\{\mathbf{a}^j\}, \{\mathbf{a}^i\})} &= \frac{1}{K} \sum_{j=1}^{K} \left[\frac{1}{K-1} \sum_{i=1}^{K} \sum_{k=1}^{d} (\mathbf{a}_k^j - \mathbf{a}_k^i)^2 \right] \\ &= \frac{1}{K(K-1)} \sum_{j=1}^{K} \sum_{i=1}^{K} \sum_{k=1}^{d} (\mathbf{a}_k^j - \mathbf{a}_k^i)^2 \end{aligned} \tag{D.5}$$

The intraset distance may also be expressed in terms of the variance associated with the components of the pattern points. Rearranging, we may write

$$\begin{aligned} \overline{D^2(\{\mathbf{a}^j\}, \{\mathbf{a}^i\})} &= \frac{K}{K-1} \sum_{k=1}^{d} \left[\frac{1}{K^2} \sum_{j=1}^{K} \sum_{i=1}^{K} (\mathbf{a}_k^j - \mathbf{a}_k^i)^2 \right] \\ &= \frac{2K}{K-1} \left[\overline{(\mathbf{a}_k^i)^2} - \overline{(\mathbf{a}_k^i)}^2 \right] \end{aligned} \tag{D.6}$$

The last step follows since $\overline{(\mathbf{a}_k^j)^2} = \overline{(\mathbf{a}_k^i)}^2$ and $\frac{1}{K} \sum_{i=1}^{K} \overline{(\mathbf{a}_k^j)^2} = \overline{(\mathbf{a}_k^j)^2}$ and $\frac{1}{K} \sum_{j=1}^{K} \overline{(\mathbf{a}_k^i)^2} = \overline{(\mathbf{a}_k^i)^2}$

Since $\overline{(\mathbf{a}_k^i)^2} - \overline{(\mathbf{a}_k^i)}^2 = (\sigma_k^*)^2$ is the biased sample variance of the kth component of the K pattern points in $\{\mathbf{a}^i\}$, the intraset distance is given by

$$\overline{D^2} = \frac{2K}{K-1} \sum_{k=1}^{d} (\sigma_k^*)^2 \tag{D.7}$$

Noting that

$$(\sigma_k)^2 = \frac{K}{K-1} (\sigma_k^*)^2 \tag{D.8}$$

The intraset distance can be written in terms of the unbiased sample variance

$$\overline{D^2} = 2 \sum_{k=1}^{d} (\sigma_k)^2 \tag{D.9}$$

Appendix E

The Entropy of a Gaussian Distribution

The entropy of a pdf $f(x)$ is defined as $H(x) = -\int_{-\infty}^{\infty} f(x) \log f(x)$, if $f(x)$ is n-dimensional Gaussian distributed, we have (Jones 1979 pp 151)

$$f(x) = \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right] \tag{E.1}$$

$$\log f(x) = \log \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} - \frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu) \log e \tag{E.2}$$

respectively, where $\triangle$ is the determinant of the matrix Σ^{-1}.

i.e. $H(x) =$

$$-\int_{-\infty}^{\infty} \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \log \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right] d\mathbf{x}$$
$$+\int_{-\infty}^{\infty} \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \frac{\log e}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu) \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right] d\mathbf{x} \tag{E.3}$$

Let $(\mathbf{x}-\mu) = \mathbf{u}$, since Σ the covariance matrix is symmetric and positive definite, a linear non-singular transform exists such that $L^T \Sigma^{-1} L = \Lambda$ where $\mathbf{u} = L\mathbf{v}$. It is noted that Λ is a diagonal matrix with elements $\lambda_1, \ldots, \lambda_n$.

$$
\begin{aligned}
H(x) &= -\int_{-\infty}^{\infty} \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \log \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \exp\left[-\frac{1}{2}\mathbf{v}^T\Lambda\mathbf{v}\right] d\mathbf{v} \\
&\quad + \int_{-\infty}^{\infty} \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \frac{\log e}{2} \mathbf{v}^T\Lambda\mathbf{v} \exp\left[-\frac{1}{2}\mathbf{v}^T\Lambda\mathbf{v}\right] d\mathbf{v} \\
&= -\frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \log \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \\
&\quad \exp\left[-\frac{1}{2}(\lambda_1\mathbf{v}_1^2 + \lambda_2\mathbf{v}_2^2 \ldots + \lambda_d\mathbf{v}_n^2)\right] d\mathbf{v}_1\mathbf{v}_2 \ldots d\mathbf{v}_n \\
&\quad + \frac{\triangle^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} \log e \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \frac{1}{2}(\lambda_1\mathbf{v}_1^2 + \lambda_2\mathbf{v}_2^2 \ldots + \lambda_n\mathbf{v}_n^2) \\
&\quad \exp\left[-\frac{1}{2}(\lambda_1\mathbf{v}_1^2 + \lambda_2\mathbf{v}_2^2 \ldots + \lambda_n\mathbf{v}_n^2)\right] d\mathbf{v}_1 d\mathbf{v}_2 \ldots d\mathbf{v}_n \\
&= \log \frac{(2\pi e)^{\frac{n}{2}}}{\triangle^{\frac{1}{2}}} \qquad \text{(E.4)}
\end{aligned}
$$

The last equation is obtained since the integral can be carried out on each $\mathbf{v}$ separately.

Appendix F

Least Square Method for Geometry Rectification

Rectification of image require fitting of a model to the ground control points (GCP) that satisfy a least-squares criteria (section 6.2.5).

Consider a single control point (see eqn. 6.27) and assume we are attempting to compute the c_k coefficients:

$$\begin{aligned}\gamma_i &= c_0x_{0i} + c_1x_{1i} + c_2x_{2i} + c_3x_{3i} + c_4x_{4i} \\ &\quad + c_5x_{5i} + c_6x_{6i} + c_7x_{7i} + c_8x_{8i} + c_9x_{9i}\end{aligned} \tag{F.1}$$

If we use n GCPs in total we shall have n such equations which we may write in matrix form as:

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix} = \begin{bmatrix} x_{01} & x_{11} & x_{21} & \cdots & x_{91} \\ x_{02} & x_{12} & x_{22} & \cdots & x_{92} \\ \vdots & & & & \\ x_{0n} & x_{1n} & x_{2n} & \cdots & x_{9n} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_9 \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{F.2}$$

We have to include the errors since there will not be a set of $c_0 \ldots c_9$ which will simultaneously provide us with exactly $\gamma_1 \ldots \gamma_n$ in the overdetermined case.

Equation F.2 can be abbreviated and generalized to m degree polynomial,

$$\Gamma = \mathbf{x}\mathbf{c} + \mathbf{e} \tag{F.3}$$

where Γ can be either the longitude or latitude to be calculated. We want to obtain $\mathbf{c}$ such that the sum of square error $\mathbf{e}^{\mathbf{T}}\mathbf{e}$ is minimized,

$$\mathbf{e} = \Gamma - \mathbf{x}\mathbf{c} \tag{F.4}$$

we form the sum of the square of each error by computing $\mathbf{e}^{\mathbf{T}}\mathbf{e}$:

$$\mathbf{e}^{\mathbf{T}}\mathbf{e} = (\Gamma - \mathbf{x}\mathbf{c})^{\mathbf{T}}(\Gamma - \mathbf{x}\mathbf{c}) \tag{F.5}$$

Differentiating $\mathbf{e}^{\mathbf{T}}\mathbf{e}$ with respect to Γ and equating to zero:

$$\begin{aligned}
\frac{\mathbf{d}(\mathbf{e}^{\mathbf{T}}\mathbf{e})}{\mathbf{d}(\mathbf{c})} &= (\mathbf{0} - \mathbf{x}\mathbf{I})^{\mathbf{T}}(\Gamma - \mathbf{x}\mathbf{c}) + (\Gamma - \mathbf{x}\mathbf{c})^{\mathbf{T}}(\mathbf{0} - \mathbf{x}\mathbf{I}) \\
&= (-\mathbf{x}\mathbf{I})^{\mathbf{T}}(\Gamma - \mathbf{x}\mathbf{c}) + (\Gamma^{\mathbf{T}} - (\mathbf{x}\mathbf{c})^{\mathbf{T}})(-\mathbf{x}\mathbf{I}) \\
&= -\mathbf{I}\mathbf{x}^{\mathbf{T}}(\Gamma - \mathbf{x}\mathbf{c}) + (\Gamma^{\mathbf{T}} - \mathbf{c}^{\mathbf{T}}\mathbf{x}^{\mathbf{T}})(-\mathbf{x}\mathbf{I}) \\
&= -\mathbf{I}\mathbf{x}^{\mathbf{T}}\Gamma + \mathbf{I}\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{c} - \Gamma^{\mathbf{T}}\mathbf{x}\mathbf{I} + \mathbf{c}^{\mathbf{T}}\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{I} \\
&= 2(\mathbf{I})(\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{c} - \mathbf{x}^{\mathbf{T}}\Gamma)
\end{aligned} \tag{F.6}$$

Since $\Gamma^{\mathbf{T}}\mathbf{x}\mathbf{I} = (\Gamma^{\mathbf{T}}\mathbf{x}\mathbf{I})^{\mathbf{T}}$ and $\mathbf{c}^{\mathbf{T}}\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{I} = (\mathbf{c}^{\mathbf{T}}\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{I})^{\mathbf{T}}$, because both are 1 x 1 matrix. Equating eqn. F.6 to zero thus:

$$\begin{aligned}
\mathbf{0} &= 2(\mathbf{I})(\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{c} - \mathbf{x}^{\mathbf{T}}\Gamma) \\
(\mathbf{x}^{\mathbf{T}}\mathbf{x})^{-1}\mathbf{x}^{\mathbf{T}}\mathbf{x}\mathbf{c} &= (\mathbf{x}^{\mathbf{T}}\mathbf{x})^{-1}\mathbf{x}^{\mathbf{T}}\Gamma \\
\mathbf{c} &= (\mathbf{x}^{\mathbf{T}}\mathbf{x})^{-1}\mathbf{x}^{\mathbf{T}}\Gamma
\end{aligned} \tag{F.7}$$

The elements in matrix $\mathbf{x}$ is substitute for variables given by equation 6.27. The matrix $\mathbf{x}$ for $m = 3$ is therefore:

$$\mathbf{x} = \begin{bmatrix} 1 & P_1 & L_1 & P_1^2 & P_1L_1 & L_1^2 & P_1^3 & P_1^2L_1 & P_1L_1^2 & L_1^3 \\ 1 & P_2 & L_2 & P_2^2 & P_2L_2 & L_2^2 & P_2^3 & P_2^2L_2 & P_2L_2^2 & L_2^3 \\ \vdots & & & & & & & & & \\ 1 & P_n & L_n & P_n^2 & P_nL_n & L_n^2 & P_n^3 & P_n^2L_n & P_nL_n^2 & L_n^3 \end{bmatrix} \tag{F.8}$$

and

$$\mathbf{x^T\Gamma} = \mathbf{x^T} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix} \tag{F.9}$$

Appendix G

Surface Chart of the Images Used in This Study

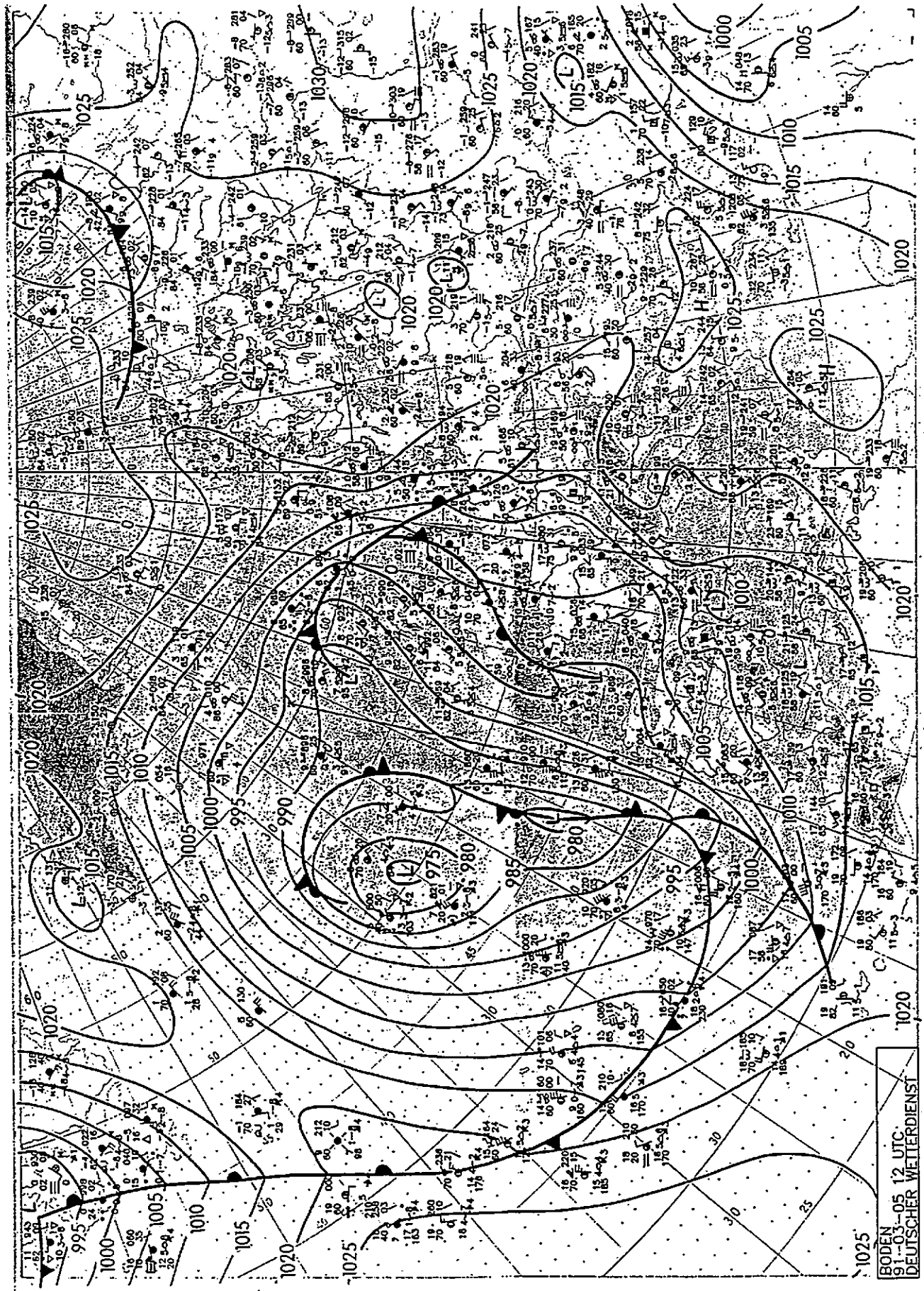


Figure G.1: Surface chart of 5th March.

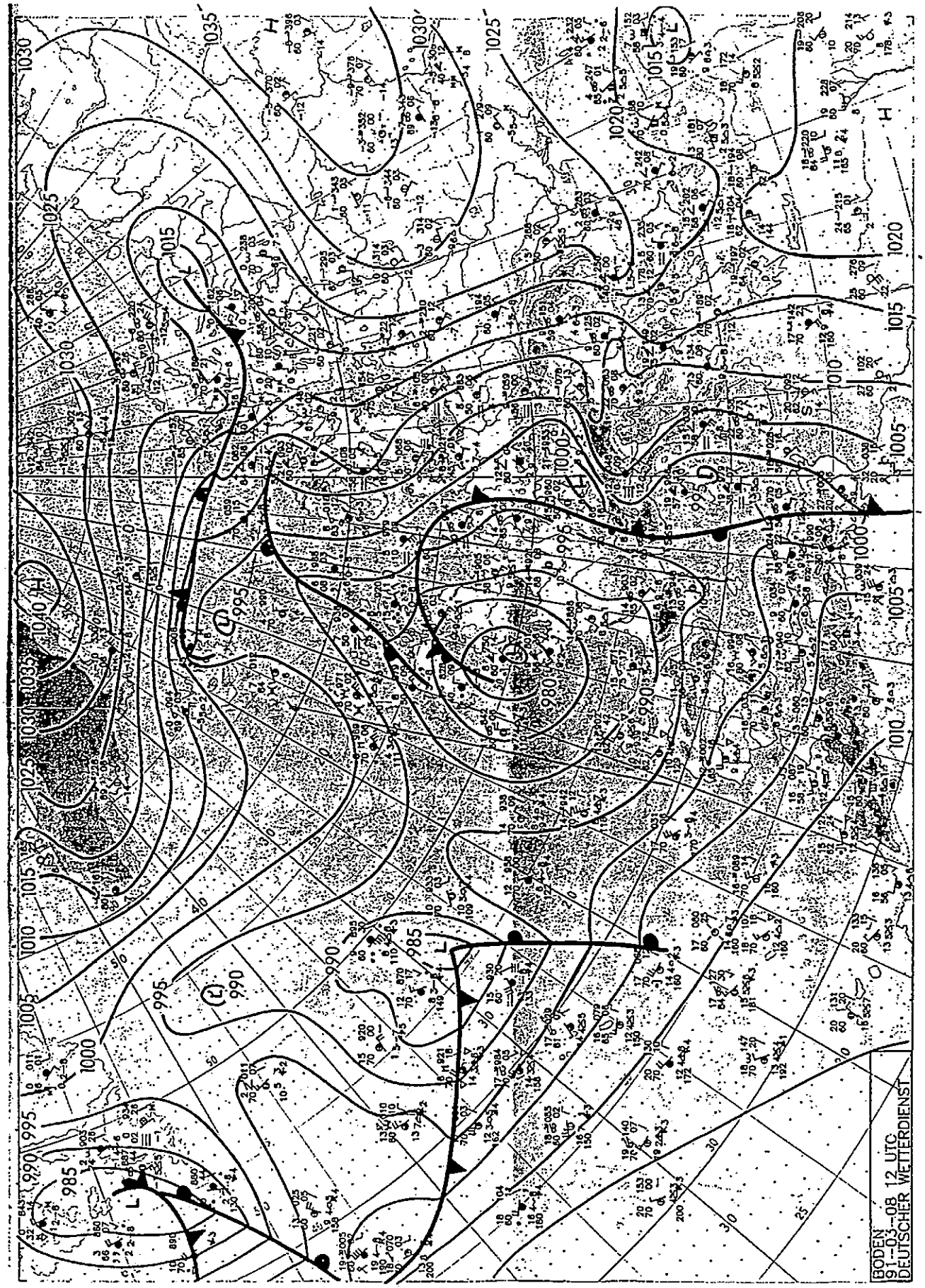


Figure G.2: Surface chart of 8th March.

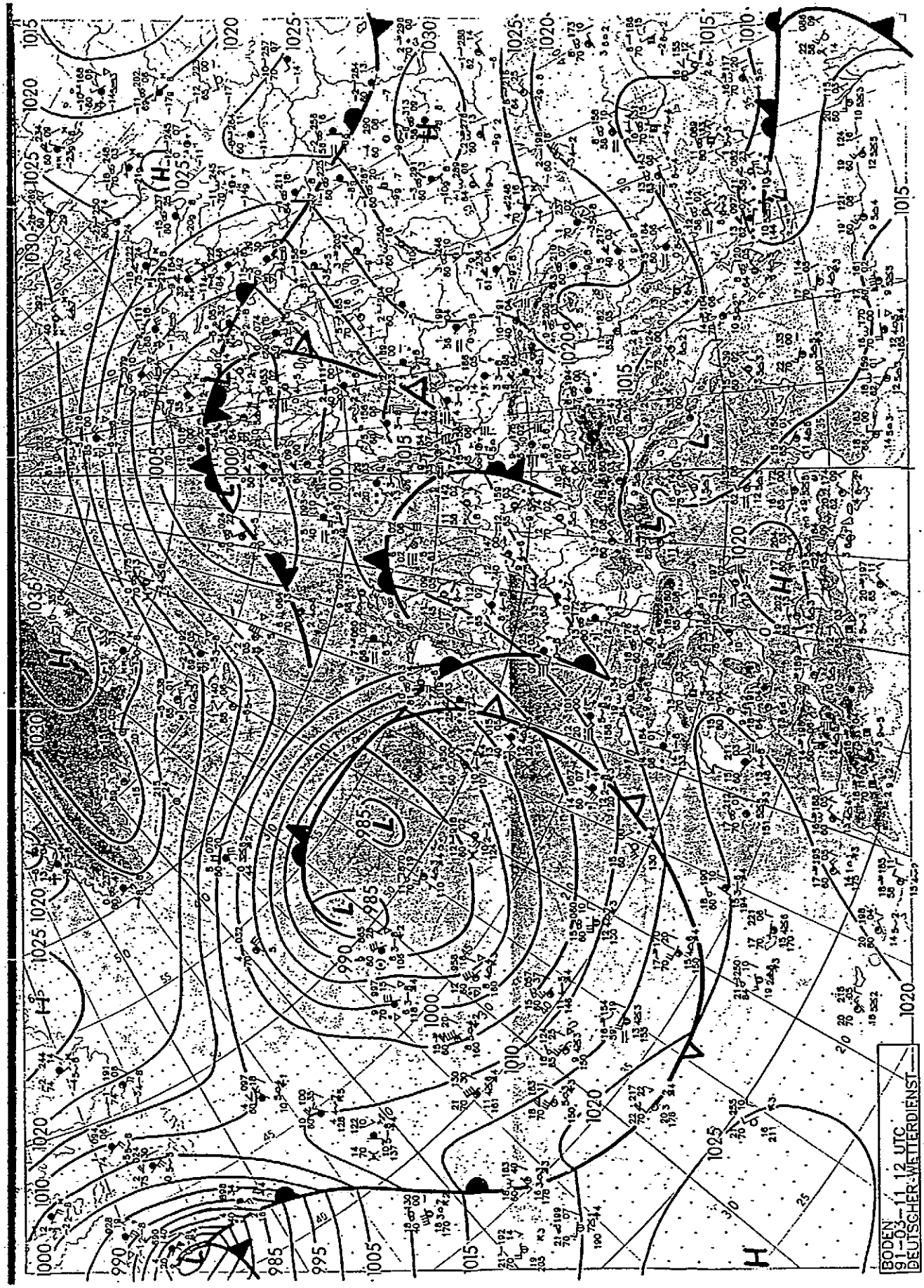


Figure G.3: Surface chart of 11th March.

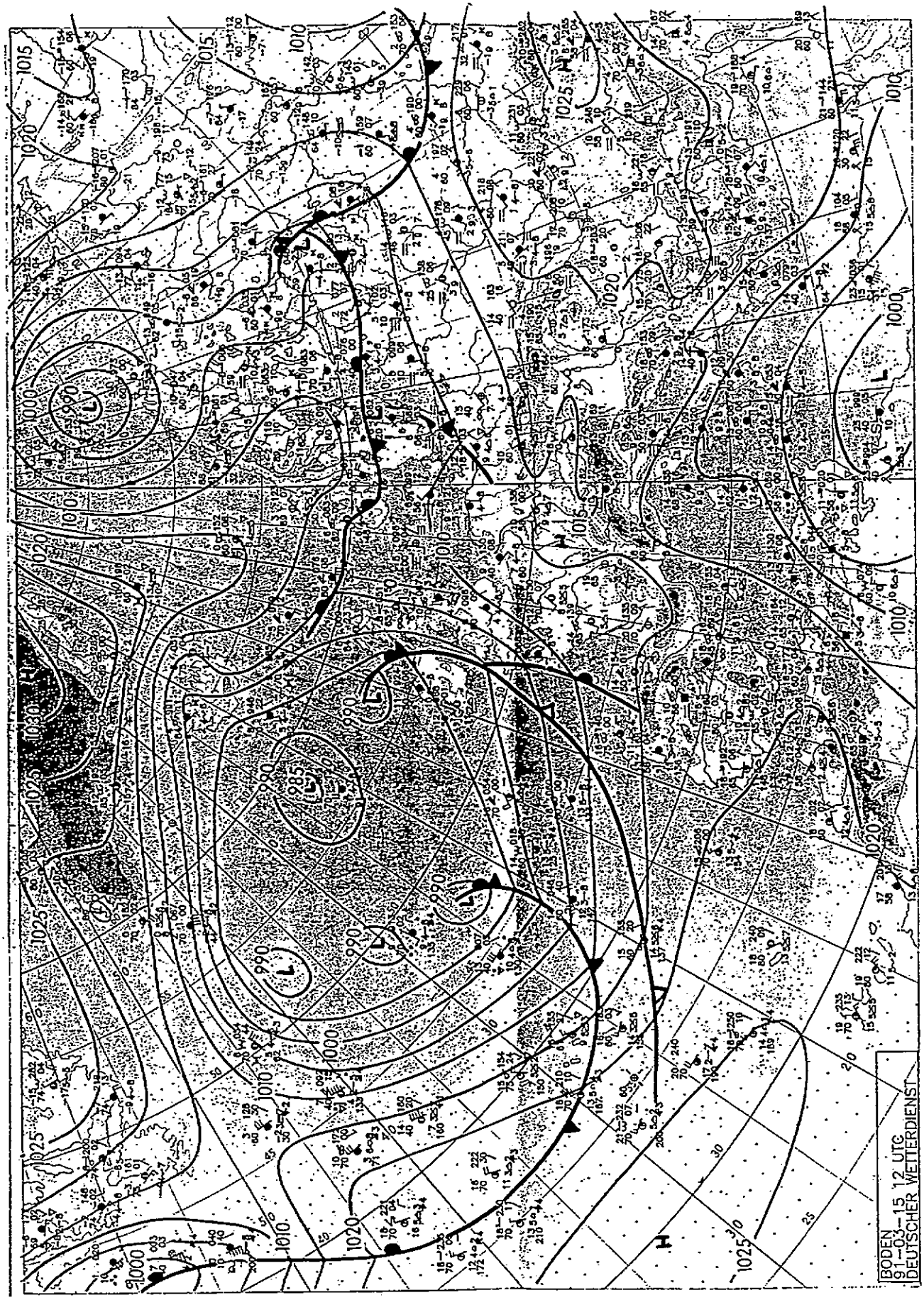


Figure G.4: Surface chart of 15th March.

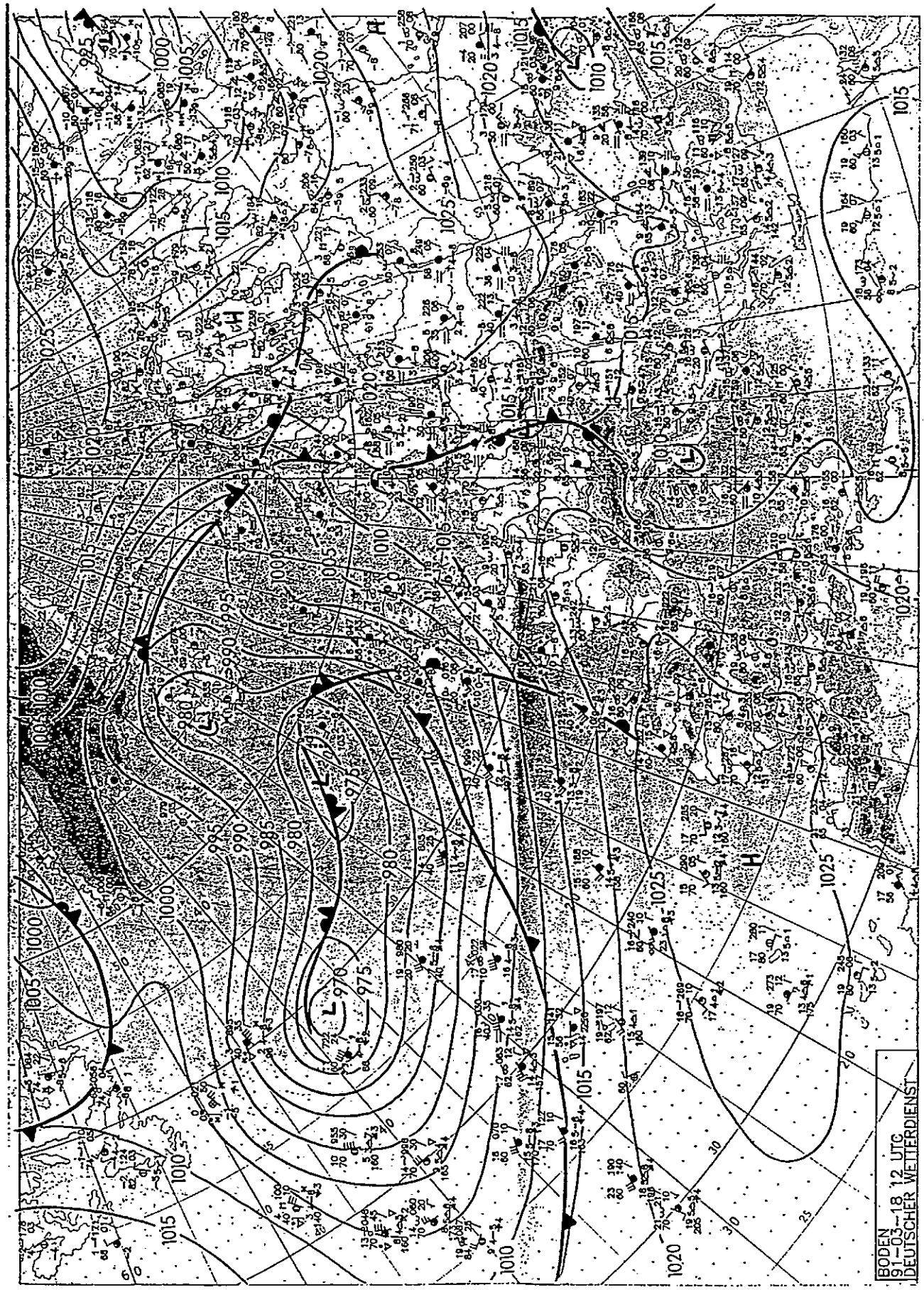


Figure G.5: Surface chart of 18th March.

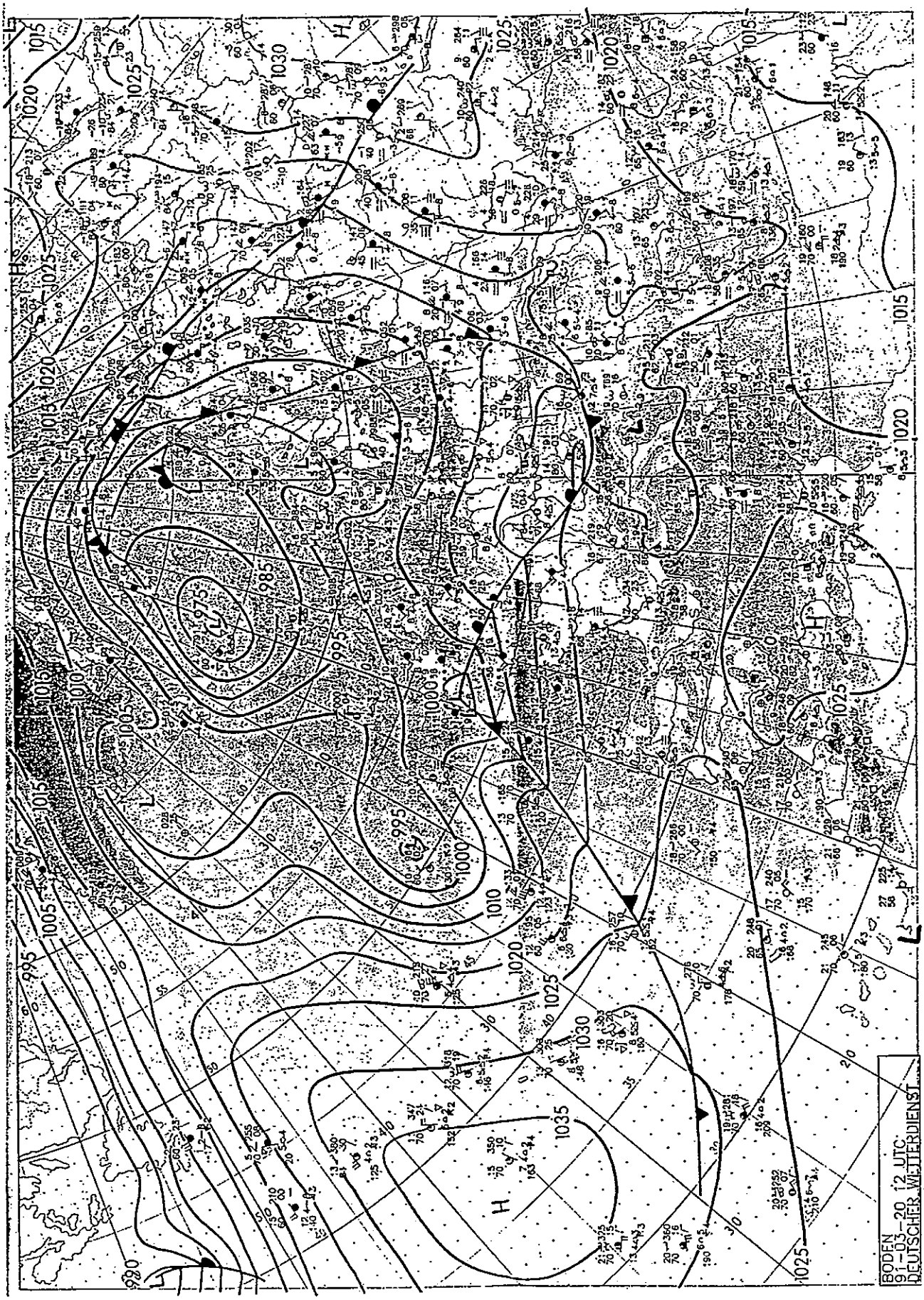


Figure G.6: Surface chart of 20th March.

Appendix H

Programs of the Global-Local Clustering Algorithm

```
/*****************************************************************************
 * Noted that this is a collection of files, so some extern variable         *
 * declaration has been deleted, to be compliable some changes has to        *
 * be made.                                                                  *
 * The Global-Local algorithm includes the following files:                  *
 * 1) isovar1.h                                                              *
 * 2) isodata1.h                                                             *
 * 3) graph.h                                                                *
 * 4) output.h                                                               *
 * 5) hicap.h                                                                *
 * 6) iso8.c                                                                 *
 * 7) isodata1.c                                                             *
 * 8) iso_io1.c                                                              *
 * 9) hicap1.c                                                               *
 * 10) smooth2.c                                                             *
 * 11) graph2.c                                                              *
 * 12) output1.c                                                             *
 *****************************************************************************/

/*****************************************************************************
 * You are reading isovar1.h                                                 *
 *****************************************************************************/
extern char  *optarg;
extern int  optind, opterr;

u_char  image[DIMENSION][MAX_YSIZE][MAX_XSIZE];/* store the image data */
char   label[MAX_YSIZE][MAX_XSIZE];/* label the samples */
int  dimension,/* number of channel used, vis, ir, wv */
     nos_center,/* the current number of clusters */
     nos_discard,/* number of discarded center */
     init_num_center,/* desired number of centers form histogram clustering */
     count[MAX_CENTER],/* the number of samples in a cluster */
     top_x, top_y,/* top left of the image window to be clustered */
     win_xsize, win_ysize,/* window sizes */
     gc, errflag;/* for getopt */
float  theta_s,/* standard deviation parameter */
       theta_l,/* lumping parameter */
       overall_id,/* overall average intraset distance */
       intra_d[MAX_CENTER],/* intraset distance */
       center[DIMENSION][MAX_CENTER],/* cluster center */
       sum[DIMENSION][MAX_CENTER],/* for standard deviation */
       sum_sq[DIMENSION][MAX_CENTER],/* for standard deviation */
       deviation[DIMENSION][MAX_CENTER];/* standard deviation of cluster */

/* these are for histogram clustering */
int  num_distinct,/* number of distinct vector */
     cell_size,/* the size of histogram cell */
     succlen,/* total length of hash */
     maxhash,/* record of longest hash */
     max_freq;/* record of maximum frequency */

HIST_TABLE  **hist_ptr,/* for sorting */
            table[TABLESIZE];/* histogram table */

struct rasterfile  header;/* info. of input image, for output result */

int  compare();
float  distance_to_center();

/*****************************************************************************
 * You are reading isodata1.h                                                *
 *****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <math.h>
```

```
#include <malloc.h>
#include <pixrect/pixrect_hs.h>

#define  MAX_XSIZE      512 /* maximum window size */
#define  MAX_YSIZE      512
#define  WIN_XSIZE      256
#define  WIN_YSIZE      256
#define  MAX_CENTER     20/* estimated max number of center during process */
#define  DIMENSION      3/* vis, ir, wv */
#define  MAX_GREY       255.0/* maximum greyscale */
#define  MIN_GREY       0.0/* minimum greyscale */
#define  OUT_CHAN       1/* the channel that will be used for output */

/* parameter for isodata */
#define  GAMMA          0.8/* parameter for newly splited center */
#define  THETA_N        0.01/* min number of samples */
#define  LUMP           2/* max number of pairs of center to be lumped */
#define  ITERATION      11/* number of iteration allowed */
/*#define  K              9 number of center desired */

#define  TABLESIZE      60013/* 12007 histogram table size */

#define  TRUE           1
#define  FALSE          !TRUE
#define  CHANGE         1
#define  UNCHANGE       !CHANGE
#define  SQUARE(x)      ((x)*(x))
#define  strsave(s)     (strcpy(malloc(strlen(s)+1), s))
#define  demand(fact, remark) {\
        if (!(fact)) {\
            fprintf(stderr, "demand not met: fact\n");\
            fprintf(stderr, "remark\n");\
            exit(1);\
        }\
}

typedef struct {/* for function lump cluster */
int  c1;
int  c2;
float  distance;
}INTER_CLUST;

typedef struct {/* for function lump cluster */
int  c1;
int  c2;
}CENTER_PAIR;

typedef struct {/* struct for histogram table */
float  prob;/* probablity */
int  freq;/* frequency count */
int  fpos;/* index in neighbor list file */
u_char  vec[4];/* the vector values */
char  label;/* class label */
}HIST_TABLE;

/*********************************************************************
 * You are reading graph.h                                           *
 *********************************************************************/

#define  NOTVISIT    0
#define  VISIT       !NOTVISIT

typedef struct tree {
int  key;/* the key for this node */
struct tree  *next;
char  status;
} TREE;/* memory is allocated in order of declaration */

typedef struct {
/*int  key;*/
struct tree  *node;
} TREE_HEAD;

typedef struct list {
int  key;
struct list  *next;
} LIST;


float  distance(),
       get_gradient();
TREE  *get_node(),
      *depth_first_next(),
      *find_alone_node(),
      *front_of_tree();
TREE_HEAD  *depth_first_search();
LIST  *front_of_list();

/*************************************************************
 * You are reading output.h                                  *
 *************************************************************/
```

```
#define   OUT_CHANNEL  0/* normally visible */
#define   LIMIT        10/* number of inter cluster distance to be shown */

float  intra_distance();

/*****************************************************************
 * You are reading hicap1.h                                      *
 *****************************************************************/
#include "isodata1.h"

#define  NOS_NEIGH      26/* number of neighbors for 3 dimension */
#define  BUFSIZE        ((int)(TABLESIZE/10))

#define  MAX_ROOT       50/* max number of root(clusters) allowed */

#define   SMALL      -1E30/* a very small number */

typedef struct {/* to compute the offset table */
char  vec[4];
} VECTOR;

typedef struct {/* struct for neighbor list buffer */
short  count;/* number of neighbors */
u_char  me[4];/* the vector itself */
u_char  neigh[NOS_NEIGH][4];/* neighbor of this vector */
}NEIGH;

typedef struct {/* use for statisic */
float  sum[DIMENSION];
float  ssq[DIMENSION];
int  num;
} STAT;


/*****************************************************************************
 * You are reading iso8.c                                                    *
 *                                                                           *
 * This is the main program of the Global-Local clustering algorithm         *
 * presented in Chapter 4 of this thesis. The first stage of the             *
 * Global-Local algorithm is a histogram clustering algorithm, and           *
 * consists of files: hicap.c smooth.c graph.c The second stage of the       *
 * Global-Local algorithm is a dynamic clustering algorithm based on         *
 * ISODATA and consists of files: iso8.c isodata1.c iso_io1.c output1.c      *
 *                                                                           *
 * hicap.c is to construct a multidimensional histogram (up to 4             *
 * variables).                                                               *
 * smooth.c is to smooth the histogram.                                      *
 * graph.c is the valley seeking algorithm which partition the histogram.    *
 * iso8.c is the main program.                                               *
 * isodata1.c, and iso_io1.c include all main functions of the second        *
 * stage.                                                                    *
 * output1.c is to compute the clustering statistics.                        *
 *                                                                           *
 * Detail implemenation can be found in ''Pattern recognition                *
 * principles'', J.T.Tou, R.C.Gonzalez Addision-Wesley Publishing            *
 * Co. Inc. 1974.                                                            *
 *                                                                           *
 * This programme is intended for clustering of METEOSAT images              *
 * up to 3 channels. visibel, infrared and water vapour respectively.        *
 *                                                                           *
 * The dynamic clustering uses J. Kittler's population weighted Gaussian     *
 * model. The multiple occurrence of pixel is exploited to increase          *
 * efficiency. The first stage partition the multidimensional histogram      *
 * and an initial partition is generated using methods described in          *
 * Chapter 4 of this thesis. The second stage use a dynamic clustering       *
 * algorithm which use a Gaussian cluster model and post transfer            *
 * advantage rule (see Chapter 4 for details).                               *
 *                                                                           *
 * K.S.LAU 1-12-90                                                           *
 *****************************************************************************/
#include "isodata1.h"
#include "isovar1.h"

float  w_covar[MAX_CENTER][DIMENSION][DIMENSION],/* within cluster covar */
       iw_covar[MAX_CENTER][DIMENSION][DIMENSION],/* inversed covar matrix */
       cosum[MAX_CENTER][DIMENSION][DIMENSION],/* for covariance matrix */
       log_covar[MAX_CENTER],/* log of covariance matrix */
       mahalanobis_distance();/* compute mahalanobis distance */

/*
 Main function of the Global-Local clustering algorithm.
 */
main(argc, argv)
int argc; char **argv;
{

    while ((gc = getopt(argc, argv, " ")) != EOF)
            switch (gc) { case '?': errflag++; break; }
    if (errflag) quit(argv);
```

```
    get_data();/* get the input image filename */
    initial_centers();/* the first stage of the Global-Local algorithm
                         */
    get_parameters();/* specify the parameters for the second stage */
    isodata_resolution();/* recompute the histogram with different
                            compression ratio */

    isodata();/* the main function for the second stage */

    map_pixel_to_histogram();/* label pixel to clusters */
    show_result();/* print statistics */
    get_objective_function_value();/* compute objective function */
    output_result();/* write output to files */
}

/*
 isodata_resolution is to recompute the histogram.
 */
isodata_resolution()
{
    char  answer[10];
    int  i;

    printf("Do you want to change the histogram cellsize for isodata? ");
    scanf("%s", answer);
    if (answer[0] != 'n')
    {   /* reconstruct histogram with new resolution */
        printf("Enter the histogram cellsize, ");
        scanf("%d", &cell_size);/* compression ratio, 1,2,4,... */
        clear_table();/* reset the hashing table */
        get_hist_table();/* compute the histogram */
        free((char *)hist_ptr);
        sort_histogram();
        /* reset the label in histogram table */
        for (i = 0; i < num_distinct; i++) hist_ptr[i]->label = 0;
    }
}

/*
 Main function of the dynamic clustering algorithm.
 */
isodata()
{
    int  iter,/* iteration counter */
         status;/* indicate any split or merge */
    /* use nearest mean assignment only after split or merge */

    status = CHANGE;
    for (iter = 1; iter <= ITERATION; iter++)
    {
        printf("\nIteration %d\n", iter);
        do {
            /* assign pixel to nearest center if iter=1, or after
               split or merge */
            if (status == CHANGE) nearest_center_assignment();
            /* assign distinct vector using post transfer advantage rule
             */
            population_weighted_Gaussian_reassignment(status);
            /* remove small cluster */
            status = discard_cluster();
        } while (status == CHANGE);
        print_statistic();

        if (iter == ITERATION) break;/* no lump or split for last iter. */

        if (iter % 2 == 0 || nos_center >= 2*init_num_center)
        { /* too many clusters */
            status = lump_cluster();
            continue;
        }

        if ((status = split_cluster()) == CHANGE) ;
        else if (LUMP > 0) status = lump_cluster();/* no split, do lump */
    }
}

/*
 lump_cluster is to merge any clusters which are closer than the
 specified threshold.
 */
int  lump_cluster()
{
     register int  i, j, k, class;
     int  lump_pair = 0,/* check any cluster to lump */
          merged_center = 0,/* record merged center */
          end,/* index of last center in table */
          used,/* label */
          status = UNCHANGE,/* any merge? */
          nos_pair,/* how many pair of center to be test */
          compare();
     float  inter_d,/* inter cluster distance */
```

```
        groups_var[DIMENSION],/* within groups sd of kth channel */
        new_center[DIMENSION];/* new center */
    INTER_CLUST  close_center[MAX_CENTER],/* record of qualified centers */
                 *ptr[MAX_CENTER];/* for sorting */
    CENTER_PAIR  used_center[LUMP];/* record of already merged center */

    for (i = 0; i < nos_center-1; i++)
        for (j = i+1; j < nos_center; j++)
        {   /* for all pairwise of clusters */
            /* check how many pair is to merge */
            if (lump_pair >= MAX_CENTER) break;/* cannot save any more */
            /* compute the within group variance of two clusters */
            get_within_groups_var(i, j, groups_var);
            inter_d = 0.0;/* initialise intercluster distance */
            for (k = 0; k < dimension; k++)
                inter_d += (((float)count[i]*(float)count[j])
                            *SQUARE(center[k][i]-center[k][j]))
                            /((float)(count[i]+count[j])*groups_var[k]);
            if (inter_d < theta_1)/* check lumping threshold */
            {
                close_center[lump_pair].c1 = i;
                close_center[lump_pair].c2 = j;
                close_center[lump_pair].distance = inter_d;
                lump_pair++;/* count how many pair to merge */
            }
        }
    if (lump_pair == 0) return(status);/* no lumping */

    for (k = 0; k < lump_pair; k++)/* initialise for sorting */
        ptr[k] = &close_center[k];
    /* select the closest pairs to merge */
    qsort((char *)ptr, lump_pair, sizeof(INTER_CLUST *), compare);
    end = nos_center-1;/* end of list position */
    nos_pair = (lump_pair > LUMP) ? LUMP : lump_pair;

    for (class = 0; class < nos_pair; class++)
    {
        used = FALSE;
        if (merged_center > 0)
            for (i = 0; i < merged_center; i++)
                /* check whether center has been merged */
                if (ptr[class]->c1 == used_center[i].c1
                 || ptr[class]->c1 == used_center[i].c2
                 || ptr[class]->c2 == used_center[i].c1
                 || ptr[class]->c2 == used_center[i].c2)
                    used = TRUE;
        if (used) continue;/* do not merge */
        printf("merge cluster %d %d\n", ptr[class]->c1, ptr[class]->c2);
        status = CHANGE;/* centers has been merged */
        for (k = 0; k < dimension; k++)/* merge */
        {/* compute new centre */
            new_center[k] =
            ((float)count[ptr[class]->c1]*center[k][ptr[class]->c1]
            +(float)count[ptr[class]->c2]*center[k][ptr[class]->c2])
            /(float)(count[ptr[class]->c1]+count[ptr[class]->c2]);
        }
        used_center[merged_center].c1 = ptr[class]->c1;
        used_center[merged_center].c2 = ptr[class]->c2;
        merged_center++;
        for (k = 0; k < dimension; k++)/* put into list */
            center[k][ptr[class]->c1] = new_center[k];
        if (ptr[class]->c2 < end)
            move_center(end, ptr[class]->c2);/* move upward */
        end--;
    }
    nos_center -= merged_center;/* update current no. of clusters */

    return(status);
}

/*
 check if cluster kernels have to be recomputed.
 */
population_weighted_Gaussian_reassignment(status)
int  status;/* any split or merge in previous iteration */
{
    int  class;

    if (status == CHANGE)
    {/* compute covariance matrix of every clusters */
        get_wcovar_matrix();/* within cluster covariance matrix */
        for (class = 0; class < nos_center; class++)
        {
            log_matrix_determinant(class);/* log covariance matrix */
            inverse_matrix(class);/* inverse of covariance matrix */
        }
    }

    /* assign distinct vector using post transfer advantage rule */
    population_weighted_Gaussian_assignment();
    /* get cluster statistics */
```

```
    get_cluster_parameters();
}

/*
 assign distinct vectors using post transfer advantage rule.
 */
population_weighted_Gaussian_assignment()
{
    register int  class, i, j;/* counter */
    int  wi, wr;/* cluster r and i */
    float  k,/* number of copies */
           dr, di,/* change in cerition function for cluster r and i */
           min_dr,/* min of change for r, r!=i */
           delta_sum, delta_sumsq,/* change in stat */
           total;/* total number of pixel */

    total = (float)(win_xsize*win_ysize);

    for (i = 0; i < num_distinct; i++)
    {
        wi = hist_ptr[i]->label;
        if (wi < 0) continue;/* this is a outlier */
        k = (float)hist_ptr[i]->freq;
        di = log_covar[wi]
             -(((float)count[wi]-k)/k)
             *log(1.0-(k*mahalanobis_distance(i, wi))
             /((float)count[wi]-k))
             -(2.0*log((float)count[wi]/total))
             -(((float)dimension+2.0)*(((float)count[wi]-k)/k)
             *log((float)count[wi]/((float)count[wi]-k)));
        min_dr = di;
        for (class = 0; class < nos_center; class++)
        {
            if (class == wi) continue;
            dr = log_covar[class]
                 +(((float)count[class]+k)/k)
                 *log(1.0+(k*mahalanobis_distance(i, class))
                 /((float)count[class]+k))
                 -(2.0*log((float)count[class]/total))
                 +(((float)dimension+2.0)*(((float)count[class]+k)/k)
                 *log((float)count[class]/((float)count[class]+k)));
            if (dr < min_dr)
            {
                wr = class;
                min_dr = dr;
            }
        }
        if (min_dr < di)
        {/* update statistic of wr and wi and reassign */
            wi = hist_ptr[i]->label;/* class i */
            hist_ptr[i]->label = wr;/* reassign to class r */

            update_wcovar_matrix(wr, i, 1.0);/* update the matrix */
            update_wcovar_matrix(wi, i, -1.0);
            log_matrix_determinant(wr);
            log_matrix_determinant(wi);
            inverse_matrix(wr);
            inverse_matrix(wi);

            /* update statistic */
            count[wi] -= hist_ptr[i]->freq;/* update count */
            count[wr] += hist_ptr[i]->freq;/* update count */
            for (j = 0; j < dimension; j++)
            {
                delta_sum = (float)hist_ptr[i]->freq
                            *(float)hist_ptr[i]->vec[j];
                delta_sumsq = (float)hist_ptr[i]->freq
                              *SQUARE((float)hist_ptr[i]->vec[j]);
                sum[j][wr] += delta_sum;/* update sum */
                sum_sq[j][wr] += delta_sumsq;
                sum[j][wi] -= delta_sum;
                sum_sq[j][wi] -= delta_sumsq;
                center[j][wr] = sum[j][wr]/(float)count[wr];/* update mean*/
                center[j][wi] = sum[j][wi]/(float)count[wi];
                deviation[j][wr] = ((float)count[wr]*sum_sq[j][wr]
                                    -SQUARE(sum[j][wr]))
                                    /((float)count[wr]*(float)(count[wr]-1));
                deviation[j][wr] = sqrt(deviation[j][wr]);
                deviation[j][wi] = ((float)count[wi]*sum_sq[j][wi]
                                    -SQUARE(sum[j][wi]))
                                    /((float)count[wi]*(float)(count[wi]-1));
                deviation[j][wi] = sqrt(deviation[j][wi]);
            }
        }
    }
}

/*
 Update a covariance matrix.
 */
update_wcovar_matrix(class, xl, sign)
```

```
int  class,/* the cluster which para are to be updated */
     xl;/* the vector which is being reassigned */
float  sign;/* add or subtract */
{
register int  i, j;
float  x[DIMENSION],/* a buffer matrix */
       k,/* the number of copy of this pixel */
       c0, c1;/* constant */

k = (float)hist_ptr[xl]->freq;

for (i = 0; i < dimension; i++)
    x[i] = center[i][class]-(float)hist_ptr[xl]->vec[i];

c0 = sign*(k/((float)count[class]+sign*k));
c1 = (float)count[class]/((float)count[class]+sign*k);
for (i = 0; i < dimension; i++)
    for (j = 0; j < dimension; j++)
    {
        w_covar[class][i][j] += c0*x[i]*x[j];
        w_covar[class][i][j] *= c1;
    }
}

get_wcovar_matrix()/* compute within cluster covariance matrix */
{
register int  a, j, k, class;

for (class = 0; class < nos_center; class++)/* initialise */
    for (j = 0; j < dimension; j++)
        for (k = 0; k < dimension; k++)
    cosum[class][j][k]= 0.0;

for (a = 0; a < num_distinct; a++)/* compute co product */
{
    class = hist_ptr[a]->label;
    if (class < 0) continue;/* outlier */
    for (j = 0; j < dimension; j++)
        for (k = j+1; k < dimension; k++)
            cosum[class][j][k]
            += ((float)hist_ptr[a]->vec[k]*
                (float)hist_ptr[a]->vec[j]*
                (float)hist_ptr[a]->freq);
}

for (class = 0; class < nos_center; class++)
    compute_comatrix(class);
}

compute_comatrix(class)/* compute covariance matrix */
register int  class;
{
    register int  j, k;

    for (j = 0; j < dimension; j++)
        for (k = j; k < dimension; k++)
            if (k != j)/* covariance */
            {
                w_covar[class][j][k]
                    = ((float)count[class]*cosum[class][j][k]
                      -sum[j][class]*sum[k][class])
                      /((float)count[class]*(float)(count[class]-1));
                w_covar[class][k][j] = w_covar[class][j][k];
            }
            else/* variance */
            {
                w_covar[class][j][k]
                    = ((float)count[class]*sum_sq[k][class]
                      -SQUARE(sum[k][class]))
                      /((float)count[class]*(float)(count[class]-1));
            }
}

log_matrix_determinant(class)/* log of a matrix's determinant */
int  class;
{
    int  iflag,
         ipivot[DIMENSION],
         istar,
         i, j, k;
    float  awikod,
           colmax,
           ratio,
           rowmax,
           temp,
           d[DIMENSION],
           w[DIMENSION][DIMENSION];

    /* copy covar matrix into w */
    for (i = 0; i < dimension; i++)
        for (j = 0; j < dimension; j++)
```

```
            w[i][j] = w_covar[class][i][j];

    iflag = 1;
    /* initialise ipivot, d */
    for (i = 0; i < dimension; i++)
    {
        ipivot[i] = i;
        rowmax = 0.0;
        for (j = 0; j < dimension; j++)
            if (rowmax < fabs(w[i][j])) rowmax = fabs(w[i][j]);
        if (rowmax == 0.0)
        {
            iflag = 0;
            rowmax = 1.0;
        }
        d[i] = rowmax;
    }
    /* factorisation */
    for (k = 0; k < dimension-1; k++)
    {
        /* determine pivot row, the row istar */
        colmax = fabs(w[k][k])/d[k];
        istar = k;
        for (i = k+1; i < dimension; i++)
        {
            awikod = fabs(w[i][k])/d[i];
            if (awikod > colmax)
            {
                colmax = awikod;
                istar = i;
            }
        }
        if (colmax == 0.0) iflag = 0;
        else
        {
            if (istar > k)
            {
                /* make k the pivot row by interchangeing
                   it with the chosen row istar */
                iflag = -iflag;
                i = ipivot[istar];
                ipivot[istar] = ipivot[k];
                ipivot[k] = i;
                temp = d[istar];
                d[istar] = d[k];
                d[k] = temp;
                for (j = 0; j < dimension; j++)
                {
                    temp = w[istar][j];
                    w[istar][j] = w[k][j];
                    w[k][j] = temp;
                }
            }
            /* eliminate  x[k] from rows k+1...n */
            for (i = k+1; i < dimension; i++)
            {
                w[i][k] /= w[k][k];
                ratio = w[i][k];
                for (j = k+1; j < dimension; j++) w[i][j] -= ratio*w[k][j];
            }
        }
    }
    if (w[dimension-1][dimension-1] == 0.0) iflag = 0;

    log_covar[class] = (float)iflag;
    for (i = 0; i < dimension; i++) log_covar[class] *= w[i][i];
    log_covar[class] = log(log_covar[class]);
}

inverse_matrix(class)/* find the inverse of a matrix */
register int  class;
{
    register int  i, j, k;

    for (i = 0; i < dimension; i++)
        for (j = 0; j < dimension; j++)
            iw_covar[class][i][j] = w_covar[class][i][j];

    /* compute elements of reduced matrix */
    for (k = 0; k < dimension; k++)
    {   /* new elements of pivot row */
        for (j = 0;  j < dimension; j++)
            if (j != k) iw_covar[class][k][j] /= iw_covar[class][k][k];
        /* element replacing pivot element */
        iw_covar[class][k][k] = 1.0/iw_covar[class][k][k];
        /* compute new elements not in pivot row or pivot column */
        for (i = 0; i < dimension; i++)
            if (i != k)
                for (j = 0; j < dimension; j++)
                    if (j != k)
                        iw_covar[class][i][j] = iw_covar[class][i][j]
```

```
                            -iw_covar[class][k][j]*iw_covar[class][i][k];
        /* compute replacement elements for
           pivot column-except pivot element */
        for (i = 0; i < dimension; i++)
            if (i != k)
                iw_covar[class][i][k] *= -iw_covar[class][k][k];
    }
}

/* compute mahalanobis distance of a distinct vector and a cluster */
float  mahalanobis_distance(i, class)
register int  i, class;
{
    register int  k, a, b;
    float  vector[DIMENSION],
           result[DIMENSION],
           distance = 0.0;

    for (k = 0; k < dimension; k++) result[k] = 0.0;

    for (k = 0; k < dimension; k++)
        vector[k] = (float)hist_ptr[i]->vec[k]-center[k][class];

    for (a = 0; a < dimension; a++)
        for (b = 0; b < dimension; b++)
            result[a] += vector[b]*iw_covar[class][b][a];

    for (k = 0; k < dimension; k++)
        distance += result[k]*vector[k];

    return(distance);
}

/*************************************************************************
 * You are reading isodata1.c                                            *
 * Functions for second stage of the Global-Local clustering algorithm.  *
 *************************************************************************/
#include "isodata1.h"

float  distance_to_center();

/*
 print the statistic of clusters after each iteration.
 */
print_statistic()
{
    int  i, j, k, class;
         sum_change;/* number of pixel changed center */
    float  inter_d, groups_var[DIMENSION];
    static  int  first_time = 1,/* counter */
                 last_count[MAX_CENTER];/* number of pixel in previous run */

    printf("\t***** START WITH CENTERS *****");
    printf("\n     ");
    for (class = 0; class < nos_center; class++)
        printf(" c%2d    ", class);
    for (k = 0; k < dimension; k++)
    {
        printf("\n"); printf("%2d: ", k);
        for (class = 0; class < nos_center; class++)
            printf("%5.1f,  ", center[k][class]);
    }printf("\n");

    printf("\t**** NUMBER OF OBJECTS PER CLUSTER ****\n");
    for (class = 0; class < nos_center; class++)
        printf("   c%2d  ", class);
    printf("\n");
    for (class = 0; class < nos_center; class++)
        printf("%6d  ", count[class]);
    printf("\n");

    printf("\t***** STANDARD DEVIATION *****");
    printf("\n     ");
    for (class = 0; class < nos_center; class++)
        printf("  c%2d   ", class);
    for (k = 0; k < dimension; k++)
    {
        printf("\n"); printf("%2d: ", k);
        for (class = 0; class < nos_center; class++)
            printf("%5.1f,  ", deviation[k][class]);
    }printf("\n");

    printf("\t***** INTER CLUSTER DISTANCE *****\n");
    for (class = 1; class < nos_center; class++)
        printf("     %2d ", class);
    printf("\n");

    for (i = 0; i < nos_center-1; i++)
    {
        printf("%2d  ", i);
        for (class = 0; class < i; class++) printf("        ");
```

```
        for (j = i+1; j < nos_center; j++)
        {
            get_within_groups_var(i, j, groups_var);
            inter_d = 0.0;
            for (k = 0; k < dimension; k++)
                inter_d += (((float)count[i]*(float)count[j])
                            *SQUARE(center[k][i]-center[k][j]))
                            /((float)(count[i]+count[j])*groups_var[k]);
            inter_d = sqrt(inter_d);
            printf("%5.1f,  ", inter_d);
        }
        printf("\n");
    }
    if (first_time)
    {/* first call of this function */
    for (i = 0; i < nos_center; i++)
        last_count[i] = count[i];
    first_time = 0;
    }
    else
    {
        sum_change = 0;
        for (i = 0; i < nos_center; i++)
            sum_change += abs(last_count[i]-count[i]);
        for (i = 0; i < nos_center; i++)
            last_count[i] = count[i];
        printf("Percentage of pixel transferred is %.2f\n",
               100.0*(float)sum_change/(float)(win_xsize*win_ysize));
    }
}

/*
 assign vectors to the nearest center
 */
nearest_center_assignment()
{

euclidean_assignment();
get_cluster_parameters();
}

euclidean_assignment()
{
     register int  i, class;
     int  min_class;/* the minimum distance class */
     float  distance, min_distance;

     for (class = 0; class < nos_center; class++)/* initialise */
         count[class] = 0;

     for (i = 0; i < num_distinct; i++)/* for all samples */
     {
         if (hist_ptr[i]->label < 0) continue;/* this is a outlier */
         min_class = 0;/* initialise */
         min_distance = distance_to_center(i, 0);
         for (class = 1; class < nos_center; class++)
         {
             distance = distance_to_center(i, class);
             if (distance < min_distance)
             {
                 min_class = class;
                 min_distance = distance;
             }
         }
         hist_ptr[i]->label = min_class;
         count[min_class] += hist_ptr[i]->freq;
     }
}

/*
 remove any cluster smaller than the threshold.
 threshold is defined in header file.
 */
int  discard_cluster()
{/* discarded samples will not be visit any more */
    register int  i, class;
    int  end,/* position index of center list end */
         discard = UNCHANGE,/* set to no discared cluster */
         current_center;/* number of center currently exist */

    end = nos_center-1;/* last index of array */
    current_center = nos_center;
    for (class = 0; class < current_center; class++)/* for all class */
        if (count[class] < win_xsize*win_ysize*THETA_N)/* discard */
        {
            nos_discard++;
            /* keep visit points
            for (i = 0; i < num_distinct; i++)
                if (hist_ptr[i]->label == class)
                    hist_ptr[i]->label = -nos_discard;
            */
```

```
            if (class < end)/* not the last center in list */
                move_center(end, class);/* shuffle center*/
            current_center--;/* one cluster less */
            end--;
        }
    if (nos_center > current_center)
    {
        discard = CHANGE;
        printf("Discard %d centers\n", nos_center-current_center);
    }
    nos_center = current_center;/* update */
    return(discard);
}

get_cluster_parameters()
{/* computer center, sd, intraset distance, and overall intraset distance */
 /* counting of number of pixel will be done in the reassignment routine */
    register int  k, i, class;

    for (k = 0; k < dimension; k++)
        for (class = 0; class < nos_center; class++)
            sum[k][class] = sum_sq[k][class] = 0.0;/* initialise */

    for (i = 0; i < num_distinct; i++)
    {
        class = hist_ptr[i]->label;
        if (class < 0) continue;/* outlier */
        for (k = 0; k < dimension; k++)/* sum all clusters */
        {
            sum[k][class] += (float)hist_ptr[i]->vec[k]*
                             (float)hist_ptr[i]->freq;
            sum_sq[k][class] += SQUARE((float)hist_ptr[i]->vec[k])*
                                (float)hist_ptr[i]->freq;
        }
    }

    for (k = 0; k < dimension; k++)/* compute center mean */
        for (class = 0; class < nos_center; class++)
            center[k][class] = sum[k][class]/(float)count[class];

    for (class = 0; class < nos_center; class++)/* standard deviation */
        for (k = 0; k < dimension; k++)
        {
            deviation[k][class] = ((float)count[class]*sum_sq[k][class]
                                   -SQUARE(sum[k][class]))
                                /((float)count[class]*(float)(count[class]-1));
            deviation[k][class] = sqrt(deviation[k][class]);
        }

    for (class = 0; class < nos_center; class++)/* intraset distance */
    {
        intra_d[class] = 0.0;
        for (k = 0; k < dimension; k++)
            intra_d[class] += ((float)count[class]*sum_sq[k][class]
                               -SQUARE(sum[k][class]))
                            /((float)count[class]*(float)(count[class]-1));
        intra_d[class] *= 2.0;/* intraset distance, use unbaised var */
        /*intra_d[class] = sqrt(intra_d[class]);*/
    }

    overall_id = 0.0;/* compute overall average within cluster distance */
    for (class = 0; class < nos_center; class++)
        overall_id += intra_d[class];
    overall_id /= (float)nos_center;
}

/*
 split any clusters whose variance larger than threshold.
 */
int  split_cluster()
{
    register int  class, k;
    int  end,/* position index of last center element */
         status = UNCHANGE,/* any split ? */
         current_center,/* number of center currently exist */
         max_channel;/* channel which has max deviation */
    float  max_sd;/* max standard deviation */

    end = nos_center-1;
    current_center = nos_center;/* take a record */
    for (class = 0; class < nos_center; class++)/* for all cluster */
    {
        max_channel = 0;/* initialise */
        max_sd = deviation[0][class];
        for (k = 1; k < dimension; k++)/* find max sd */
            if (deviation[k][class] > max_sd)
            {
                max_channel = k;
                max_sd = deviation[k][class];
            }
        if (max_sd > theta_s) ;/* do nothing, follow on */
```

```
        else          continue;/* not satisfy for splitting */

        if (nos_center <= init_num_center/2)
        /*if (nos_center <= init_num_center-1)*/
        {
            printf("split cluster %d\n", class);
            split(end, class, max_channel);
            status = CHANGE;
            current_center++;
            end++;
        }
        /* split even near limit */
        else if (intra_d[class] > overall_id
                 && count[class] > 2*(win_xsize*win_ysize*THETA_N+1))
        {
                printf("split cluster %d\n", class);
                split(end, class, max_channel);
                status = CHANGE;
                current_center++;
                end++;
        }
    }
    nos_center = current_center;/* update */

    return(status);
}

split(end, class, channel)/* split a cluster */
int  end,/* position index of last center */
     class,/* which cluster */
     channel;/* which channel */
{
    register int  k;
    float  center_plus[DIMENSION],
           center_minus[DIMENSION];

    for (k = 0; k < dimension; k++)/* compute new centers */
    {
        /*if (k == channel)*//* split on max sd channel */
        /*{*/
        if (deviation[k][class] > theta_s)
        {/* compute new cluster centers */
            center_plus[k] = center[k][class]+deviation[k][class]*GAMMA;
            center_minus[k] = center[k][class]-deviation[k][class]*GAMMA;
            if (center_plus[k] > MAX_GREY)
            {
                center_plus[k] = MAX_GREY;
                printf("Split center out of range, reduced\n");
            }
            if (center_minus[k] < MIN_GREY)
            {
                center_minus[k] = MIN_GREY;
                printf("Split center out of range, increased\n");
            }
        }
        /*}*/
        else/* other channels unchange */
        {
            center_plus[k] = center[k][class];
            center_minus[k] = center[k][class];
        }
    }
    /* move center */
    for (k = 0; k < dimension; k++)
    {
        center[k][class] = center_plus[k];/* insert into old position */
        center[k][end+1] = center_minus[k];/* append to end of list */
    }
}

/*
 compute within group variance.
 */
get_within_groups_var(c1, c2, groups_var)
int  c1, c2;/* which group */
float  *groups_var;
{
    register int  k;
    float  ssq1, ssq2;
    /*
 Ref:Pearson, K. 1936
     On the coefficient of racial likeness.
     Biometrika 18, 105.
     */


    for (k = 0; k < dimension; k++)
    {
        ssq1 = ((float)count[c1]*sum_sq[k][c1]-SQUARE(sum[k][c1]))
                        /(float)(count[c1]-1);
        ssq2 = ((float)count[c2]*sum_sq[k][c2]-SQUARE(sum[k][c2]))
```

```
                        /(float)(count[c2]-1);
        groups_var[k] = (ssq1+ssq2)
                        /((float)count[c1]+(float)count[c2]-2.0);
    }
}

int  compare(a, b)
INTER_CLUST **a, **b;
{/* ascending order of magnitude */
if ((*a)->distance > (*b)->distance)        return(1);
else                                        return(-1);
}

float  distance_to_center(i, class)
register int  i, class;
{
     register int  k;
     float  distance = 0.0;

     for (k = 0; k < dimension; k++)
         distance += SQUARE((float)hist_ptr[i]->vec[k]-center[k][class]);

     return(distance);/* squared Euclidean distance */
}

move_center(from, to)/* update array center[][] */
register int  from, to;
{
register int  k;

for (k = 0; k < dimension; k++)
    center[k][to] = center[k][from];
count[to] = count[from];
}

/*************************************************************************
 * You are reading iso_io1.c                                             *
 * Functions to show the result, store any results generated by second   *
 * stage of the Global-Local clustering algorithm.                       *
 * And some auxiliary functions.                                         *
 * K.S.LAU  13-8-90                                                      *
 *************************************************************************/

#include "isodata1.h"

/**
 label pixel.
 **/
map_pixel_to_histogram()
{
u_char  vector[4], r;
register  int  i, j, k, xi;

for (i = 0; i < win_ysize; i++)
    for (j = 0; j < win_xsize; j++)
    {
        for (k = 0; k < dimension; k++)
        {
            vector[k] = image[k][i][j];
            r = vector[k]%cell_size;
            vector[k] -= r;
        }
        for ( ; k < 4; k++) vector[k] = 0;
        xi = search(vector);
        label[i][j] = table[xi].label;
    }
}

output_result()
{
char  *output1 = "/home/image/output/hiso.clus",
      *output2 = "/home/image/output/hiso.map";
u_char  buffer[MAX_YSIZE][MAX_XSIZE];
int  i, j, a, b;
FILE  *f1, *f2;

if ((f1 = fopen(output1, "w")) == NULL)
{ fprintf(stderr, "Cannot open output file\n"); exit(1); }
if ((f2 = fopen(output2, "w")) == NULL)
{ fprintf(stderr, "Cannot open output file\n"); exit(1); }

fwrite((char *)&header, sizeof(struct rasterfile), 1, f1);

for (i = 0; i < header.ras_height; i++)/* initialise */
    for (j = 0; j < header.ras_width; j++)
        buffer[i][j] = 0;

for (a = 0, i = top_y; i < top_y+win_ysize; a++, i++)/* get value */
    for (b = 0, j = top_x; j < top_x+win_xsize; b++, j++)
        buffer[i][j] = center[OUT_CHAN][label[a][b]];
```

```
for (i = 0; i < header.ras_height; i++)
    fwrite((char *)&buffer[i][0], sizeof(char), header.ras_width, f1);

printf("The clustered image is stored in %s\n", output1); fclose(f1);

fwrite((char *)&nos_center, sizeof(int), 1, f2);
fwrite((char *)&dimension, sizeof(int), 1, f2);
fwrite((char *)&top_x, sizeof(int), 1, f2);
fwrite((char *)&top_y, sizeof(int), 1, f2);
fwrite((char *)&win_xsize, sizeof(int), 1, f2);
fwrite((char *)&win_ysize, sizeof(int), 1, f2);
fwrite((char *)count, sizeof(int), nos_center, f2);
for (i = 0; i < nos_center; i++)
    for (j = 0; j < dimension; j++)
        fwrite((char *)&center[j][i], sizeof(float), 1, f2);
for (i = 0; i < nos_center; i++)
    for (j = 0; j < dimension; j++)
        fwrite((char *)&deviation[j][i], sizeof(float), 1, f2);
for (i = 0; i < win_ysize; i++)
    fwrite((char *)&label[i][0], sizeof(char), win_xsize, f2);

printf("The cluster map is stored in %s\n", output2); fclose(f2);
}

show_result()
{
int i, j;

printf("CLUSTER RESULT\n");
for (i = 0; i < 20; i++)
{
    printf("\n");
    for (j = 0; j < 20; j++)
        printf("%3d ", label[i][j]);
}
printf("\n");
}

quit(argv)
char  **argv;
{
fprintf(stderr, "Usage: %s: interactive, no option\n", argv[0]);
exit(1);
}

get_data()/* specify input file names */
{
char  filename[DIMENSION][50];
int  i, j;
FILE  *f[DIMENSION];
struct  rasterfile  head[DIMENSION];

printf("Enter the number of channel to be used < 4, ");
scanf("%d", &dimension);

for (i = 0; i < dimension; i++)
{
    printf("Enter the channel filename %d, ", i);
    scanf("%s", (char *)&filename[i][0]);
    f[i] = fopen((char *)&filename[i][0], "r");
    demand(f[i], Cannot open file);
}

        /* store header in global area, for later use */
        fread((char *)&header, sizeof(struct rasterfile), 1, f[0]);
        rewind(f[0]);
        for (i = 0; i < dimension; i++)
        {
            fread((char *)&head[i], sizeof(struct rasterfile), 1, f[i]);
    demand(header.ras_length == head[i].ras_length,
          Make sure the images has the same size and coordinates);
}

        printf("The images size is %d\n", header.ras_width);
        printf("Enter the window width (xsize), ");
        scanf("%d", &win_xsize);
        printf("Enter the window height (ysize), ");
        scanf("%d", &win_ysize);

        printf("Enter the top left window coordinate of
               the image to be processed,\n");
        printf("\tx: "); scanf("%d", &top_x);
        printf("\ty: "); scanf("%d", &top_y);


demand(head[0].ras_width >= win_xsize+top_x,
       The window is out of range\, please reduce size);

for (i = 0; i < dimension; i++)
          fseek(f[i], (long)(top_y*head[i].ras_width+top_x), 1);
for (i = 0; i < dimension; i++)
```

```
    for (j = 0; j < win_ysize; j++)
    {
        fread((char *)&image[i][j][0], sizeof(char), win_xsize, f[i]);
        fseek(f[i], (long)(head[i].ras_width-win_xsize), 1);
    }

for (i = 0; i < dimension; i++) fclose(f[i]);
}

get_parameters()
{
printf("Enter the maximum standard deviation of cluster, ");
scanf("%f", &theta_s);
printf("Enter the minimum inter center distance, ");
scanf("%f", &theta_l); theta_l = SQUARE(theta_l);
init_num_center = nos_center;/* set desired number of centers */
}

get_objective_function_value()
{/* the value is only a indication, rather meaningless */
int  i, class,
     discard_count[MAX_CENTER];
float  sum_covar, sum_entropy, weight;

if (nos_discard > 0)
{
    printf("%d CLUSTER HAS BEEN DISCARD.\n", nos_discard);
    for (class = 0; class < nos_discard; class++)
        discard_count[class] = 0;
    for (i = 0; i < num_distinct; i++)
        if (hist_ptr[i]->label < 0)
            discard_count[-hist_ptr[i]->label-1] += hist_ptr[i]->freq;
    for (class = 0; class < nos_discard; class++)
        printf("   c%2d  ", class);
    printf("\n");
    for (class = 0; class < nos_discard; class++)
        printf("%5d,  ", discard_count[class]);
    printf("\n");
}

sum_covar = sum_entropy = 0.0;
for (i = 0; i < nos_center; i++)
{
    weight = (float)count[i]/(float)(win_xsize*win_ysize);
    sum_covar += log_covar[i]*weight;
    sum_entropy += -2.0*weight*log(weight);
}
printf("SUM WEIGHTED |COVARIANCE| %.1f\n", sum_covar);
printf("SUM ENTROPY %.1f\n", sum_entropy);
printf("OBJECTIVE FUNCTION VALUE IS %.1f\n", sum_covar+sum_entropy);
}

/*************************************************************************
 * You are reading hicap1.c                                              *
 * The first stage of Global-Local clustering algorithm.                 *
 * Contains machine depend code.                                         *
 * To cluster Meteosat images(upto 3 channels)using histogram clustering.
 * Ref: A NON-PARAMETRIC CLUSTERING SCHEME FOR LANDSAT                   *
       P.M.NARENDRA and M.GOLDBERG                                       *
       PATTERN RECOGNITION Vol.9 pp. 207-215 1977                        *
 * K.S.LAU  4-9-90                                                       *
 *************************************************************************/
#include "hicap.h"

#define   CENTER_THRES  200/* cluster less than this will not be selected */

int  num_clus,/* number of clusters found */
     smoothsize;/* window size for histogram smoothing */
float  thresh;/* threshold for histogram smoothing */

initial_centers()
{
char  answer[5];
int  i, k,
     done,
     threshold;
float  mean[DIMENSION][MAX_ROOT];
STAT  stat[MAX_ROOT];

printf("Enter the histogram compression ratio, ");
printf("must be 1 or power of 2, ");
scanf("%d", &cell_size);
do
{
            clear_table();/* initialise hashing table */
            done = get_hist_table();/* compute histogram */
    if (done)
    {
        printf("Compression ratio is %d, ", cell_size);
        printf("do you want to increase further? ");
        scanf("%s", answer);
```

```
        if (answer[0] == 'y') done = !done;
    }
            if (!done) cell_size *= 2;/* double compression ratio */
} while (!done);/* compress more if table is full */
printf("The compression ratio is %d\n", cell_size);
sort_histogram();

do
{
            get_smoothing_parameter();/* specify smoothing threshold */
            normalise_frequency();/* normalise frequency from 0 to 1 */
            smooth_histogram();
            done = density_graph();/* valley seeking clustering */
    if (done)
    {
        get_cluster_statistic(stat, mean, label);
        printf("Do you want to run with different parameters? ");
        scanf("%s", answer);
    }
    else
    {
        puts("Find too many clusters, try again!");
        answer[0] = 'y';
    }
} while (answer[0] != 'n');
/*
printf("Please enter a threshold,\n");
printf("clusters less than this threshold will not be selected, ");
scanf("%d", &threshold);
*/
threshold = CENTER_THRES;
printf("Centers with less than %d points are not selected\n",
      CENTER_THRES);
for (nos_center = i = 0; i < num_clus; i++)
        {/* take down the mean of the clusters selected */
    if (stat[i].num < threshold) continue;
    for (k = 0; k < dimension; k++)
        center[k][nos_center] = mean[k][i];
    nos_center++;
}
for (i = 0; i < num_distinct; i++) hist_ptr[i]->label = 0;
}

/***
 Histogram compress use the top left elements as the representive
 elements of the vectors fall in this cell.
 ***/
int  get_hist_table()
{
u_char  r,/* remainder */
        vector[4];/* must be 4 bytes long */
register int  i, j, k;
int  count,
     empty,
     done = TRUE,
     key;

maxhash = succlen = 0;
for (i = 0; i < win_ysize; i++)
    for (j = 0; j < win_xsize; j++)
    {
        for (k = 0; k < dimension; k++)
        {/* map vector into histogram cell */
            vector[k] = image[k][i][j];
            r = vector[k]%cell_size;
            vector[k] -= r;
        }
        for ( ; k < 4; k++) vector[k] = 0;
        key = insert(vector);
        if (key < 0) return(!done);
        for (k = 0; k < dimension; k++)/* put data */
            table[key].vec[k] = vector[k];
        table[key].freq++;/* count frequency */
    }

max_freq = count = empty = 0;
for (i = 0; i < TABLESIZE; i++)
{/* find maximum and capacity */
    count += table[i].freq;
    if (table[i].freq == 0) empty++;
    if (max_freq < table[i].freq) max_freq = table[i].freq;
    table[i].label = -1;/* initialise label */
}printf("\n");
num_distinct = TABLESIZE-empty;
printf("The mode is, ");
for (i = 0; i < TABLESIZE; i++)
    if (table[i].freq == max_freq)
    {
        for (j = 0; j < dimension; j++)
            printf("%d ", table[i].vec[j]);
        printf("frequency %d\n", max_freq);
```

```
    }
printf("Mean frequency is %.2f\n", (float)count/(float)num_distinct);
printf("Number of distinct vector %d, ", TABLESIZE-empty);
printf("Total %d\n", count);
printf("Loading factor %.2f\n", 1.0-(float)empty/(float)TABLESIZE);
printf("Average probe length %.2f\n", (float)succlen/(float)count);
printf("Longest hash %d\n", maxhash);

return(done);
}

clear_table()
{
register int  i, *to_int;

for (i = 0; i < TABLESIZE; i++)
{
    to_int = (int *)table[i].vec;
    to_int[0] = 0;
    table[i].freq = 0;
}
}

normalise_frequency()
{
int  i;

for (i = 0; i < TABLESIZE; i++)
{/* normalise to become probablity */
    if (table[i].freq == 0) continue;
    table[i].prob = (float)table[i].freq/(float)max_freq;
}
}

int  insert(val)/* find a empty place to put data */
u_char  val[];
{
register int  h1, h2, try, len;

try = h1 = hash1(val);
h2 = hash2(val);
len = 1;
do
{
    if (table[try].freq == 0 || find(try, val))
    {
        succlen += len;
        break;
    }
    /* TABLESIZE and rehash value should be relatively prime */
    /* variation of double rehashing, eliminate clustering */
    try = (try+h2)%TABLESIZE;
    len++;
} while (try != h1);

if (len > maxhash) maxhash = len;/* record the longest hash */
if (len >= TABLESIZE) return(-1);
else                  return(try);
}

int  search(val)/* search vector */
register u_char  val[];
{
register int  h1, h2, try;
int  len = 0;

try = h1 = hash1(val);
h2 = hash2(val);
do
{
    if (find(try, val)) break;
    /* TABLESIZE and rehash value should be relatively prime */
    /* variation of double rehashing, eliminate clustering */
    try = (try+h2)%TABLESIZE;
    len++;
    if (len == maxhash) return(-1);/* does not exist */
} while (try != h1);

return(try);
}

int  find(try, val)/* same vector already here */
register int  try;
register u_char  val[];
{
register u_int  *a, *b;
register int  same = TRUE;

a = (u_int *)val;
b = (u_int *)table[try].vec;
if (*a != *b) same = FALSE;
```

```
return(same);
}

int  hash1(val)
register u_char  val[];
{
register u_int  *int_ptr;
register int  remainder;

int_ptr = (u_int *)val;/* concatenate all channels value */
remainder = *int_ptr % TABLESIZE;

return(remainder);
}

int  hash2(val)/* rehash when collision occurred */
register u_char  val[];
{
register int  i;
register u_int  *int_ptr;
u_char  carry[4];

int_ptr = (u_int *)carry;
*int_ptr = 0;
for (i = 0; i < dimension; i++)/* accumulate the remainder */
{
    carry[0] = carry[1] = val[i];
    *int_ptr %= TABLESIZE;
}

return((int)*int_ptr);
}

sort_histogram()
{
     int  i, j;
     static int  tablecomp();

     hist_ptr = (HIST_TABLE **)calloc((u_int)num_distinct,
                                      sizeof(HIST_TABLE));
     demand(hist_ptr, calloc failed);
     for (i = j = 0; i < TABLESIZE; i++)
         if (table[i].freq > 0) hist_ptr[j++] = &table[i];
     qsort((char *)hist_ptr, num_distinct, sizeof(HIST_TABLE *), tablecomp);
     for (i = 0; i < num_distinct; i++)/* write file index */
         hist_ptr[i]->fpos = i;
}

static int  tablecomp(i, j)
HIST_TABLE **i, **j;
{
u_int  *a, *b;/* ascending order */

a = (u_int *)(*i)->vec;
b = (u_int *)(*j)->vec;
if (*a > *b) return(1);
else         return(-1);
}

get_smoothing_parameter()
{
      char  answer[10];

      thresh = 0.0;/* default no smoothing */
      smoothsize = 3;/* minimum smooth size */
      printf("Do you want to smooth the histogram ? ");
      scanf("%s", answer);
      if (answer[0] != 'n')
      {
          printf("Enter the probablity threshold for smoothing, 0 to 1 ");
          scanf("%f", &thresh);
          printf("Enter the window size for histogram smoothing, 3,5,7... ");
          scanf("%d", &smoothsize);
      }
}

/*************************************************************************
 * You are reading smooth2.c                                             *
 * Functions to smooth the histogram generated by hicap1.c               *
 * Neighborhood size are 3,5,7...etc.                                    *
 * K.S.LAU 13-9-90                                                       *
 *************************************************************************/
#include "hicap.h"

smooth_histogram()
{
u_char  plusoff[4];/* resultant vector(offseted) */
int  i, j, k,
     xi, xj,
     level;/* level = 0, for smoothsize = 3 */
```

```
     maxneigh;/* number of elements */
float  sum,
       wsize;
VECTOR  *offset;

for (level = 1, i = 3; i < smoothsize; i+=2) level++;
wsize = pow((double)smoothsize, (double)dimension);

maxneigh = (int)wsize -1;
offset = (VECTOR *)calloc((u_int)maxneigh, (u_int)sizeof(VECTOR));
demand(offset, offset calloc fail);

        /* generate offset vector table */
offset_table(smoothsize, level, maxneigh, offset);

for (i = 0; i < num_distinct; i++)
        {   /* compute mean for each distinct vector */
    xi = hist_ptr[i]-table;
        /* smooth only cell with frequency lower than thresh */
    if (table[xi].prob > thresh) continue;

    sum = 0.0;
    sum += table[xi].prob;
    for (j = 0; j < maxneigh; j++)
    {/* find all neighbor */
        for (k = 0; k < 4; k++)
            plusoff[k] = hist_ptr[i]->vec[k]+offset[j].vec[k];
        xj = search(plusoff);
        if (xj >= 0) sum += table[xj].prob;/* found */
    }
    table[xi].prob = sum/wsize;
}
free((char *)offset);
}

offset_table(size, celldist, maxneigh, off)
int  size,/* window size */
     celldist,/* greatest cell difference from center */
     maxneigh;/* number of neighbor */
VECTOR *off;
{
char  start, count;
int  i, j, cycle, period, channel;

for (i = 0; i < maxneigh; i++)/* initialise */
    for (j = 0; j < 4; j++)
        off[i].vec[j] = 0;

for (channel = 0; channel < dimension; channel++)
        {   /* start with channel 0 */
    start = celldist*cell_size;
    count = start;
    period = (int)pow((double)size, (double)(dimension-channel-1));
    cycle = period;

    for (i = 0; i < maxneigh/2; i++)
    {
        if (cycle >= period)
        {
            count += cell_size;
            cycle = 0;
        }
        cycle++;
        if (count > start) count = -start;
        off[i].vec[channel] = count;
    }
}

for (i = 0; i < maxneigh/2; i++)/* another half */
    for (j = 0; j < dimension; j++)
        off[maxneigh-1-i].vec[j] = -off[i].vec[j];
}

/*********************************************************************
 * You are reading graph2.c                                          *
 * This file contains functions of the valley seeking                *
 * clustering algorithm which used to cluster the histogram.         *
 * Functions to construct directed graph.                            *
 * graph.c by K.S.LAU 2-10-90                                        *
 *********************************************************************/

#include "hicap.h"
#include "graph.h"

static int  *set,/* vectors in the same cluster */
            *stack,/* for depth first search */
            count_node,/* number of node used */
            numcalloc;/* number of call to calloc */
static TREE_HEAD  *new, *old;/* for depth first search */
static TREE  *freelist[MAX_ROOT];/* for freeing of node */
```

```
/*
 Valley seeking algorithm.
 */
int  density_graph()
{
int  i, done = TRUE,
     maxneigh,/* max number of neigh */
     numneigh,
     xi, xj;
u_int  neighlist[MOS_NEIGH];/* contain the neighbors */
float  gradient;
TREE  *temp;
TREE_HEAD *vertex;
LIST  *head = NULL,/* head of root list */
      *root;
VECTOR  offset[MOS_NEIGH];
static LIST  rlist[MAX_ROOT];/* for root head link list */

num_clus = count_node = numcalloc = 0;/* initialise */
vertex = (TREE_HEAD *)calloc((u_int)num_distinct, sizeof(TREE_HEAD));
demand(vertex, vertex calloc fail);
set = (int *)calloc((u_int)num_distinct, sizeof(int));
demand(set, set calloc fail);
stack = (int *)calloc((u_int)num_distinct, sizeof(int));
demand(stack, stack calloc fail);

maxneigh = (int)pow(3.0, (double)dimension)-1;
offset_table(3, 1, maxneigh, offset);

for (i = 0; i < num_distinct; i++)
{    /* for every distinct vectors */
     xi = hist_ptr[i]-table;
             /* find the maximum density gradient of a neighbourhood */
     get_numneigh_maxgrad(maxneigh, hist_ptr[i]->vec, xi, &xj,
                          neighlist, &numneigh, &gradient, offset);

     if (numneigh == 0)
     {   /* this is a root */
         if (!(num_clus < MAX_ROOT))
             { finish(vertex, set, stack); return(!done); }
 root = &rlist[num_clus++];
         root->key = xi;
 head = front_of_list(root, head);
 continue;
     }

     if (gradient < 0.0)
     {/* this is a root */
         if (!(num_clus < MAX_ROOT))
             { finish(vertex, set, stack); return(!done); }
         root = &rlist[num_clus++];
         root->key = xi;
         head = front_of_list(root, head);
     }
     else if (gradient > 0.0)
     {   /* link xj to xi */
 temp = get_node();
 temp->key = xi;
 vertex[table[xj].fpos].node
 = front_of_tree(temp, vertex[table[xj].fpos].node);
     }
     else /* gradient = 0 */
     {
 xj = avoid_cycle(vertex, &head, xi, neighlist,
                  numneigh, rlist);
 switch (xj)
 {
     case -1:
         break;
     case -2:
         finish(vertex, set, stack);
         return(!done);
         break;
     default:
         /* link xj to xi */
         temp = get_node();
         temp->key = xi;
         vertex[table[xj].fpos].node
         = front_of_tree(temp, vertex[table[xj].fpos].node);
         break;
 }
     }
}

get_cluster_label(vertex, head);

finish(vertex, set, stack);
return(done);
}

get_numneigh_maxgrad(maxneigh, me, xi, xj, neighlist, numneigh, gradient, off)
```

```
u_char  *me;
int  xi, *xj,
     maxneigh,
     *numneigh;
u_int  neighlist[];
float  *gradient;
VECTOR off[];
{
u_char  *to_char1, *to_char2;
u_int  plusoff;
int  i, j,
     max_neigh;
float  max_grad;

*numneigh = 0; max_grad = SMALL;
to_char1 = (u_char *)&plusoff;
for (i = 0; i < maxneigh; i++)
{/* for all possible neighbor */
    for (j = 0; j < 4; j++) to_char1[j] = me[j]+off[i].vec[j];
    if ((*xj = search(to_char1)) < 0) continue;/* not exist */
    *gradient = get_gradient(xi, *xj);
    if (max_grad < *gradient)
    {
        max_grad = *gradient;
        max_neigh = *xj;
    }
    to_char2 = (u_char *)&neighlist[*numneigh];/* get neighbor */
    for (j = 0; j < 4; j++) to_char2[j] = table[*xj].vec[j];
    (*numneigh)++;
}
*gradient = max_grad;
*xj = max_neigh;
}

finish(vertex, set, stack)/* free all memory when finish */
TREE_HEAD  *vertex;
int  *set, *stack;
{
free((char *)vertex); free((char *)set);
free((char *)stack); free_node();
}

TREE *get_node()/* get some memory */
{
int  sizel;/* the number of elements get each call */
static TREE  *head;

sizel = num_distinct;
if (count_node >= sizel) count_node = 0;/* reset */
if (count_node == 0)
{
    head = (TREE *)calloc((u_int)sizel, sizeof(TREE));
    demand(head, get_node fail);
    demand(numcalloc < MAX_ROOT, freelist overflow);
    freelist[numcalloc++] = head;
}
count_node++;

return(&head[count_node-1]);
}

free_node()
{
int  i;

for (i = 0; i < numcalloc; i++) free((char *)freelist[i]);
}

/* avoid forming directed cycle */
int avoid_cycle(vertex, root_head, xi, neighlist, numneigh, rlist)
int  xi,
     numneigh;
u_int  neighlist[];
LIST  **root_head,/* point to head of root list */
      rlist[];
TREE_HEAD  *vertex;
{
int  i, count = 0,
     card,
     xj;
LIST *pi_head,
     *node,
     list[MOS_NEIGH];

pi_head = NULL; /* initialize */
for (i = 0; i < numneigh; i++)
{/* for all neighbor of xi */
    xj = search((u_char *)&neighlist[i]);
    if (get_gradient(xi, xj) == 0.0)
    {/* construct set pi */
        demand(count < MOS_NEIGH, avoid cycle fail);
```

```
        node = &list[count++];
        node->key = xj;
        pi_head = front_of_list(node, pi_head);
    }
}
/* eliminate xj that has a path to xi */
eliminate(vertex, &pi_head, xi);
card = check_cardinality(root_head, pi_head, xi, rlist);
switch (card)
{
    case -1: return(-2); break;/* return and try again */
    case  0: return(-1); break;/* return and do nothing */
}

return(minimum_distance_node(pi_head, xi, neighlist, numneigh));
}

/* eliminate path that already exists */
eliminate(vertex, pi_head, xi)
int  xi;
LIST  **pi_head;
TREE_HEAD  *vertex;
{
/* search tree with xi as a node for any path */
int stack_pos, num;
TREE_HEAD *temp;
TREE  *another;

/* delete node in trees */
stack_pos = num = 0;
/* xi is not a parent node yet */
if ((another = find_alone_node(&vertex[table[xi].fpos])) == NULL)
    return;
start_depth_first(vertex, &stack_pos, &num, xi, another);
do
{
    temp = depth_first_search(vertex, &stack_pos, &num);
    old = new;
    new = temp;
} while (stack_pos != 1);

check_cycle(pi_head, num);
reset_tree(vertex, num);  /* reset the tree status */
}

/* check number of nodes in a tree */
int check_cardinality(root_head, pi_head, xi, rlist)
int  xi;
LIST  *pi_head, **root_head, rlist[];
{
int  count;
LIST  *node;

count = 0;
node = pi_head;
while (node != NULL)
{       /* count number of elements */
        ++count;
        node = node->next;
}
if (count == 0)
{ /* made xi a root */
    if (!(num_clus < MAX_ROOT)) return(-1);
    node = &rlist[num_clus++];
    node->key = xi;
    *root_head = front_of_list(node, *root_head);
}

return(count);
}

check_cycle(pi_head, num)
int num;
LIST **pi_head;
{
int i;
LIST *this, *last, dummy;

dummy.next = *pi_head;
for (i = 0; i < num; i++)
{   /* for all node in this branch */
    this = dummy.next;
    last = &dummy;
    while (this != NULL)
    {   /* for all node in set pi */
        if (set[i] == this->key)
        {  /* delete an element */
    last->next = this->next;
    break;
}
last = this;
```

```
        this = this->next;
    }
}
*pi_head = dummy.next;  /* renew the head */
}

/* find the closest node */
int minimum_distance_node(pi_head, xi, neighlist, numneigh)
int  xi,
     numneigh;
u_int  neighlist[];
LIST  *pi_head;
{
int  i, first = TRUE,
     min_neigh,
     xj;
float  min_dist, dist;
LIST  *this;

for (i = 0; i < numneigh; i++)
{    /* for all node in neighborhood list */
    xj = search((u_char *)&neighlist[i]);
    this = pi_head;
    while (this != NULL)
    {
        if (this->key == xj)
        {     /* compare distance */
    if (first)
    {  /* initialise */
               min_dist = distance(xi, xj);
               min_neigh = xj;
        first = FALSE;
            }
       else
       {
           dist = distance(xi, xj);
               if (min_dist > dist)
                   {
            min_dist = dist;
            min_neigh = xj;
                }
            }
        }
        this = this->next;
    }
}

return(min_neigh);
}

/* trace all node in a tree */
get_cluster_label(vertex, root_head)
TREE_HEAD  *vertex;
LIST  *root_head;
{
int i,
    count,/* check result */
    stack_pos,
    num,
    label;
LIST *this;
TREE_HEAD  *temp;
TREE  *another;

count = label = 0;/* invalid label are -1 */
this = root_head;
while (this != NULL)
{   /* search all trees with known root */
    stack_pos = num = 0;
    another = find_alone_node(&vertex[table[this->key].fpos]);
    if (another == NULL)
    {   /* single node */
        table[this->key].label = label++;
this = this->next;
        count++;
continue;
    }
    start_depth_first(vertex, &stack_pos, &num, this->key, another);
    do
    {
temp = depth_first_search(vertex, &stack_pos, &num);
old = new;
new = temp;
    } while (stack_pos != 1);
    this = this->next;
    for (i = 0; i < num; i++) table[set[i]].label = label;
    label++;
    count += num;
}
demand(count == num_distinct, graph construction error);
}
```

```
float  get_gradient(xi, xj)
int  xi, xj;
{
float  grad;

grad = (table[xj].prob-table[xi].prob)/distance(xi, xj);

return(grad);
}

float  distance(xi, xj)
int  xi, xj;
{
int  i;
float  dist;

        dist = 0.0;/* square Euclidean distance */
for (i = 0; i < dimension; i++)
    dist += SQUARE((float)table[xi].vec[i]-(float)table[xj].vec[i]);

return(sqrt(dist));
}

static
reset_tree(vertex, num)
int num;
TREE_HEAD *vertex;
{
int i;
TREE *this;

for (i = 0; i < num; i++)
{    /* reset all status */
     this = vertex[table[set[i]].fpos].node;
     while (this != NULL)
     {
    this->status = NOTVISIT;
    this = this->next;
     }
}
}

/* initialise depth first search */
static
start_depth_first(vertex, stack_pos, num, root, node)
int *stack_pos, *num;
int root;
TREE_HEAD *vertex;
TREE *node;
{
stack[(*stack_pos)++] = root;    /* put root to stack */
set[(*num)++] = root;  /* put root to cluster */
stack[(*stack_pos)++] = node->key;  /* put node to stack */
set[(*num)++] = node->key;  /* put node to cluster */
node->status = VISIT;  /* visit node */
new = &vertex[table[node->key].fpos];  /* for back track */
old = &vertex[table[root].fpos];            /* for back track */
}

TREE_HEAD *depth_first_search(vertex, stack_pos, num)
int *stack_pos, *num;
TREE_HEAD *vertex;
{
TREE *node;

        if ((node = depth_first_next(vertex, stack_pos)) == NULL)
           return(NULL);

stack[(*stack_pos)++] = node->key;
set[(*num)++] = node->key;
node->status = VISIT;

return(&vertex[table[node->key].fpos]);
}

TREE * depth_first_next(vertex, stack_pos)
int *stack_pos;
TREE_HEAD *vertex;
{  /* for directed graph */
TREE  *node;

if((node = find_alone_node(new)) == NULL)
{
   while(node == NULL)
   {
 /* delete stack */
 *stack_pos -= 2;
 new = &vertex[table[stack[*stack_pos]].fpos];
 (*stack_pos)++;
 node = find_alone_node(new);
```

```
 if(*stack_pos == 1 && node == NULL)    return(NULL);
   }
}

return(node);
}

static
TREE * find_alone_node(head)
TREE_HEAD  *head;
{
     TREE *this;

     this = head->node;
     while(this != NULL)
     {
   if (this->status == NOTVISIT) return(this);
   this = this->next;
     }

     return(NULL);
}

static
LIST *front_of_list(new, head)
LIST *new, *head;
{
new->next = head;
head = new;

return(head);
}

static
TREE *front_of_tree(new, list)
TREE *new, *list;
{
new->next = list;
list = new;

return(list);
}

/*********************************************************************
 * You are reading output1.c                                         *
 * Functions to obtain cluster statistics generate by graph1.c       *
 * K.S.LAU 8-10-90                                                   *
 *********************************************************************/

#include "hicap.h"
#include "graph.h"
#include "output.h"

get_cluster_statistic(stat, mean, label)
STAT  stat[MAX_ROOT];
float  mean[][MAX_ROOT];
char  label[][MAX_XSIZE];
{
u_char  r,/* remainder to map vector */
        *to_char,
        vector[4];
register int  i, j, k, a, b;
int  xi,
     show,/* number of cluster to be print on screen */
     class, index,
     numcell[MAX_ROOT];/* number of distinct vector in each cluster */
float  intra_d, inter_d,
       intra_sum, inter_sum,
       sd,
       group_var[DIMENSION];

printf("class   numpix   numcell   ");
for (i = 0; i < dimension; i++) printf("mean[%d]   ", i);
printf("intra. dist.  ");
for (i = 0; i < dimension; i++) printf("sd[%d]   ", i);printf("\n");

to_char = (u_char *)stat;/* inititalise */
for (i = 0; i < sizeof(STAT)*MAX_ROOT; i++) to_char[i] = 0;
for (i = 0; i < MAX_ROOT; i++) numcell[i] = 0;

for (i = 0; i < TABLESIZE; i++)/* count cluster's distinct vector */
    if (table[i].label >= 0) numcell[table[i].label]++;

for (a = 0, i = top_y; i < top_y+win_ysize; a++, i++)
    for (b = 0, j = top_x; j < top_x+win_xsize; b++, j++)
    {/* obtain the label for each pixel */
        for (k = 0; k < dimension; k++)
        {/* map the pixel into histogram */
            vector[k] = image[k][a][b];
            r = vector[k]%cell_size;
            vector[k] -= r;
```

```
        }
        for ( ; k < 4; k++) vector[k] = 0;
        xi = search(vector);
        label[a][b] = table[xi].label;
    }

for (i = 0; i < win_ysize; i++)
    for (j = 0; j < win_xsize; j++)
    {/* compute stat for each clusters */
        index = label[i][j];
        for (k = 0; k < dimension; k++)
        {
            stat[index].sum[k] += (float)image[k][i][j];
            stat[index].ssq[k] +=
                        SQUARE((float)image[k][i][j]);
        }
        stat[index].num++;
    }

for (i = 0; i < num_clus; i++)/* compute cluster mean */
    for (j = 0; j < dimension; j++)
        mean[j][i] = stat[i].sum[j]/(float)stat[i].num;

intra_sum = 0.0;
for (i = 0; i < num_clus; i++)
{
    printf("%3d, %8d, %8d, ", i, stat[i].num, numcell[i]);
    for (j = 0; j < dimension; j++)
        printf("%8.1f, ", mean[j][i]);
    intra_d = intra_distance(stat[i]);
    printf("%8.1f      ", sqrt(intra_d));
    intra_sum += intra_d;
    for (k = 0; k < dimension; k++)
    {
                sd = ((float)stat[i].num*stat[i].ssq[k]
                    -SQUARE(stat[i].sum[k]))/
             SQUARE((float)stat[i].num);
        printf("%6.1f  ", sqrt(sd));
    }printf("\n");
}
printf("SUM INTRASET DISTANCE %5.2f\n\n", intra_sum);

show = (num_clus > LIMIT) ? LIMIT : num_clus;
if (show != num_clus)
    printf("Only the first %d clusters are shown.\n", LIMIT);
printf("INTER CLUSTER DISTANCE\n");printf("    ");
for (class = 1; class < show; class++)
    printf("%4d     ", class);
printf("\n");

inter_sum = 0.0;
for (i = 0; i < show-1; i++)
{
    printf("%2d ", i);
    for (class = 0; class < i; class++) printf("         ");
    for (j = i+1; j < show; j++)
    {
        get_within_groups_variance(i, j, stat, group_var);
        inter_d = 0.0;
        for (k = 0; k < dimension; k++)
            inter_d += (((float)stat[i].num*(float)stat[j].num)
                        *SQUARE(mean[k][i]-mean[k][j]))
                                /((float)(stat[i].num+stat[j].num)
                                *group_var[k]);
        inter_d = sqrt(inter_d);
        inter_sum += inter_d;
        printf("%6.1f, ", inter_d);
    }
    printf("\n");
}
printf("SUM INTER CLUSTER DISTANCE %5.2f\n\n", inter_sum);
}

get_within_groups_variance(c1, c2, stat, group_var)
int  c1, c2;/* which group */
STAT  *stat;
float *group_var;
{
     register int  k;
     float  ssq1, ssq2;
     /*************************************************************
      Ref:Pearson, K. 1936
          On the coefficient of racial likeness.
          Biometrika 18, 105.
      *************************************************************/


     for (k = 0; k < dimension; k++)
     {
         ssq1 = ((float)stat[c1].num*stat[c1].ssq[k]-SQUARE(stat[c1].sum[k]))
                            /(float)(stat[c1].num-1);
```

```
            ssq2 = ((float)stat[c2].num*stat[c2].ssq[k]-SQUARE(stat[c2].sum[k]))
                                /(float)(stat[c2].num-1);

            group_var[k] = (ssq1+ssq2)
                                /((float)stat[c1].num+(float)stat[c2].num-2.0);

        }
    }

float intra_distance(sub)
STAT sub;
{
int  i;
float  sum = 0.0;

if (sub.num == 1) return(0.0);

for (i = 0; i < dimension; i++)
    sum += ((float)sub.num * sub.ssq[i] - SQUARE(sub.sum[i]))
                    /((float)sub.num * ((float)sub.num-1.0));

return(2.0*sum);
}
```

Appendix I

Programs of the Spatial-Spectral Clustering Algorithm

```
/*************************************************************************
 * The Spatial-Spectral clustering algorithm (bottom-up approach)        *
 * includes:                                                             *
 * 1) bmirs1.c                                                           *
 * 2) buseg2.c                                                           *
 * 3) bhc3.c                                                             *
 * 4) bu.h                                                               *
 * bmirs1.c and buseg2.c is to perform bottom-up segmentation, and       *
 * segments are store in a file, this file is input to bhc3.c and the    *
 * segments clustered.                                                   *
 *                                                                       *
 * The Spatial-Spectral clustering algorithm (top-down approach)        *
 * includes:                                                             *
 * 1) rst3.c                                                             *
 * 2) mm3.c                                                              *
 * 3) tdseg3.c                                                           *
 * 4) bhc3.c                                                             *
 * The mst algorithm is not included because it is a simplification      *
 * of the rst3.c                                                         *
 * Use rst3.c to construct a CEST, then store spanning tree in a file.   *
 * mm3.c read a spanning tree file, and partition it using minimax method,
 * the removed links and the spanning tree is stored in a file.          *
 * tdseg3.c read in a file created by either rst3.c or mm3.c and generate*
 * a user specified number of segments then store this segments in a file.
 * bhc3.c read a file generated by tdseg3.c and cluter the segments.     *
 *************************************************************************/

/*****************************************************************
 * You are reading cluster.h                                     *
 *****************************************************************/

#include <malloc.h>
#include <stdio.h>
#include <math.h>
#include <pixrect/pixrect_hs.h>

#define  DIMENSION          3
#define  MAX_XSIZE          128
#define  MAX_YSIZE          128   /* maximum image size */
#define  MAX_CLUS           15

#define  DUPLICATE          1     /* find single node */
#define  UNIQUE             0

#define  NOTVISIT           0     /* depth first search */
#define  VISIT              1
#define  TMPVISIT           2

#define  SQUARE(x)          ((x)*(x))  /* macro for square */
#define  demand(fact, remark) {\
        if (!(fact)) {\
            fprintf(stderr, "demand not met: fact\n");\
            fprintf(stderr, "remark\n");\
            exit(1);\
        }\
}

/*****************************************************************
 * You are reading bu.h                                          *
 *****************************************************************/
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include <math.h>
#include <pixrect/pixrect_hs.h>
```

```
#define  ROOT          1/* label for root */
#define  NROOT         !ROOT
#define  VISIT         1    /* depth first search */
#define  NOTVISIT      0
#define  MAX_XSIZE     128
#define  MAX_YSIZE     128  /* maximum image size */

/************************** STRUCTURE DEFINITION ********************/
typedef struct link {
     int node1,node2;
     float weight;
     struct link *next;
     struct link *last;
     struct link *other;
}LINK;

typedef struct tree {
     int node_pos;
     struct tree *next;
     char status;
}TREE;

typedef struct {
     int node_tag;
     TREE *node;
}LIST_HEAD;

typedef struct {
     float sum[DIMENSION];
     int num_vec;
}STAT;

/************************** STRUCTURES DECLARAION ********************/
typedef struct tree {
int node_pos;
struct tree *next;
char status;
}TREE;   /* memory is allocated in order of declaration */

typedef struct {
int node_tag;
struct tree *node;
}LIST_HEAD;

typedef struct {
int node1;
int node2;
}UNIQUE_LINK;

typedef struct {
UNIQUE_LINK node;
float weight;       /* interset distance of segments */
}LINK;

typedef struct {
int mm_node;
float var;
char root;
}MINIMAX_NODE;/* for minimax only */

typedef struct {
float sum[DIMENSION];
float ssq[DIMENSION];
int num_vec;
int num_seg;
}NODE_INTREE;/* for minimax only */

typedef struct list {
        UNIQUE_LINK node;
        float weight;
struct list *next;
struct list *last;
}LIST;

typedef struct {
float sum[DIMENSION];
float ssq[DIMENSION];
int num_vec;
}STAT;

/************************** FUNCTION PROTOTYPES ********************/
/*                      FOR CONSTRUCTING RST AND SST                */
char *      get_memory();
float       get_intraset_distance();
float       single_single();
float       single_group();
float       group_group();
float       intra_distance();
STAT        add_struct();
```

```
STAT        segment_statistic();
LIST *      lightest_link();
TREE *      get_node();
TREE *      depth_first_next();
TREE *      find_alone_node();
TREE *      front_of_tree();
LIST_HEAD * depth_first_class();

/************************** FUNCTION PROTOTYPES ********************/
/*                              FOR  MINIMAX                       */
STAT  get_tree_variance();
TREE *          depth_first_attribute();
LIST_HEAD *  depth_first_re_attribute();
LIST_HEAD *  depth_first_link_sum_square();
LIST_HEAD *  depth_first_link();

/*****************************************************************
 *  You are reading programme rst3.c                             *
 *  This program is to construct a CEST which use global information.  *
 *  Image segmentation based on graph theoretic approach.        *
 *  At the beginning only local informations are used,           *
 *  when more and more vertices are merged, more and more        *
 *  global informations will be added.                           *
 *  To obtain m segments the heaviest m-1 links will be deleted. *
 *  To further improve the CEST, top-down minimax method can be  *
 *  used to select the links to be deleted.                      *
 *  There are eight neighbours to each pixels.                   *
 *  The pattern vectors are two dimensional.                     *
 *  ****** ONLY CALCULATE THE CEST,DOSE NOT OBTAIN THE SEGMENTS ****** *
 *  ****** THE TREE IS SAVE ON THE HARD DISK FILE ******         *
 *  K.S.LAU 20-1-92                                              *
 *****************************************************************/

#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include <pixrect/pixrect_hs.h>

/************************** STRUCTURE DEFINITION *******************/
typedef struct {/* to store links in the spanning tree */
     int node1,node2;
     float weight;
}UNIQUE_LINK;

typedef struct link {/* to store links in the image graph */
     int node1,node2;
     float weight;
     struct link *next;
     struct link *last;
     struct link *other;
}LINK;

typedef struct tree {/* structure to store spanning tree */
     int node_pos;
     struct tree *next;
}TREE;

typedef struct {/* structure for a list of all the nodes */
     int node_tag;
     TREE *node;
}LIST_HEAD;

typedef struct {/* store the sum and number of nodes in a region */
     float sum[DIMENSION];
     int num_vec;
}STAT;

/************************** FUNCTION DEFITIONS ********************/
char *          get_memory();
float      link_weight();
float new_weight();
STAT add_struct();
STAT get_pix_value();
LINK *  lightest();
LINK * jump();
LINK *  compress();
TREE *  front_of_tree();

/************************** GLOBAL VARIABLES ********************/
u_char  image[DIMENSION][MAX_YSIZE][MAX_XSIZE];
int  dimension,/* dimension of the input multspectral image */
     top_x, top_y,/* top left coordinates of the window in the data file
                  */
     win_xsize, win_ysize;/* the size of the window to be processed */
struct rasterfile header;/* the header of data file, SUN rasterfile */
LINK *edges, head;  /* edges for array, head for list */

main()
{
     int s,
         total_link,/* number of links in the image graph */
```

```
        comp(),/* function for quick sort comparsion */
        cpu_time;
    LIST_HEAD *vertex;/* list of all nodes in the image */
    UNIQUE_LINK *unique_link, **heavy;/* to store the link in spanning
                                         tree */

    printf("GENERATION OF RECURSIVE SPANNING TREE AND STORE DATA ON DISK\n");

    clock();/* count cpu time */
    get_data();/* get input images */
    total_link = number_link();/* number of links in the image graph */
    /* allocate array of structure for linked list of tree */
    vertex = (LIST_HEAD *)get_memory(win_xsize*win_ysize, sizeof(LIST_HEAD));
    /* memory to store all links in the image graph */
    edges = (LINK *)get_memory(4*win_xsize*win_ysize-4, sizeof(LINK));
    /* allocate array of structure for unique link (link in spanning
       tree */
    unique_link = (UNIQUE_LINK *)get_memory(win_xsize*win_ysize-1,
                                            sizeof(UNIQUE_LINK));

    /* compute all weight of links */
    printf("Calculating the link weight.\n");
    get_link();

    printf("Doing the recursive merging...\n");
    recru_tree(unique_link, vertex);/* main function to do CEST */

    /* free the edges memory */
    free((char *)edges);
    /* allocate array of pointer for sorting array of structure */
    /* sort the link in spanning tree in descending order of weight */
    heavy = (UNIQUE_LINK **)get_memory(win_xsize*win_ysize-1,
                                       sizeof(UNIQUE_LINK *));
    /* initalize array of pointer point to unique link */
    for (s = 0; s < win_xsize*win_ysize-1; s++) heavy[s] = &unique_link[s];
    /* sort the unique link */
    qsort((char *)heavy, win_xsize*win_ysize-1, sizeof(UNIQUE_LINK *), comp);
    cpu_time = clock();/* count cpu time */
    printf("Run time was %.2f sec.\n", cpu_time / 1.0e6);

    rst_file(vertex, heavy);/* write the spanning to a file */

}
/***************************** END OF MAIN ******************************/

get_data()/* get input image */
{
    char  filename[DIMENSION][50];
    int  i, j;
    FILE  *f[DIMENSION];
    struct  rasterfile  head[DIMENSION];

    printf("Enter the number of channel to be used < 4, ");
    scanf("%d", &dimension);/* specify number of bands in image */

    for (i = 0; i < dimension; i++)
    {/* input filename */
        printf("Enter the channel filename %d, ", i);
        scanf("%s", (char *)&filename[i][0]);
        f[i] = fopen((char *)&filename[i][0], "r");
        demand(f[i], Cannot open file);
    }

    /* store header in global area, for later use */
    fread((char *)&header, sizeof(struct rasterfile), 1, f[0]);
    rewind(f[0]);
    for (i = 0; i < dimension; i++)
    {/* read data */
        fread((char *)&head[i], sizeof(struct rasterfile), 1, f[i]);
        demand(header.ras_length == head[i].ras_length,
               Make sure the images has the same size and coordinates);
    }

    printf("The images size is %d\n", header.ras_width);
    printf("Enter the window width (xsize), ");
    scanf("%d", &win_xsize);/* process window xsize */
    printf("Enter the window height (ysize), ");
    scanf("%d", &win_ysize);/* process window ysize */

    /* in this study all image file are 512 x 512, and top left
       coordinates are (330, 60) of in a B format METEOSAT image */
    printf("Enter the top left window coordinate
            of the image to be processed,\n");
    printf("\tx: "); scanf("%d", &top_x);
    printf("\ty: "); scanf("%d", &top_y);


    demand(head[0].ras_width >= win_xsize+top_x,
           The window is out of range\, please reduce size);

    for (i = 0; i < dimension; i++)
```

```
        fseek(f[i], (long)(top_y*head[i].ras_width+top_x), 1);
    for (i = 0; i < dimension; i++)
        for (j = 0; j < win_ysize; j++)
        {
            fread((char *)&image[i][j][0], sizeof(char), win_xsize, f[i]);
            fseek(f[i], (long)(head[i].ras_width-win_xsize), 1);
        }

    for (i = 0; i < dimension; i++) fclose(f[i]);/* close files */
}

char *get_memory(items, size)
unsigned items, size;
{
     char *buffer;

     buffer = (char *)calloc(items, size);
     demand(buffer, no memory);

     return(buffer);
}

int number_link()
{/* computer total number of links, in order to allocate memory */
   int s, total_link;

   total_link = 0;
   for (s = 1; s < win_xsize-1; s++)
      total_link += s;
   total_link *= 4;
   total_link += 2*(win_xsize*win_ysize-1);
   printf("\nTotal number of links in original graph is %d.\n",total_link);

   return (total_link);
}

get_link()/* compute all links in the image graph */
{
    register int i, node;
    /* the coordinate of two dimensional array is transformed to
    linear coordinate and store in the edges array, so a mapping
    has to be used to relate the two coordinate systems */

    head.next = edges;
    edges[0].last = &head;
    for (i = node = 0; node < win_xsize*win_ysize-1; node++)
    {   /* for every vertex, except the last one */
        if (node >= win_xsize*win_ysize-win_xsize)  /* last row */
        {
           if (node == win_xsize*win_ysize-2)
           {  /* last vertice */
              edges[i].node1 = node;
              edges[i].node2 = node+1;
              edges[i].weight = link_weight(node, node+1);
              edges[i].next = &head;
              head.last = &edges[i];
           }
           else   fill_edges(node, node+1, i, 4);
           edges[++i].weight = -1.0;  /* fill the gap */
           edges[++i].weight = -1.0;
           edges[++i].weight = -1.0;
           ++i;
        }
        else
        {
           if ((node % win_xsize) == 0)  /* row head */
           {
              fill_edges(node, node+1, i, 1);
              fill_edges(node, node+win_xsize+1, ++i, 1);
              fill_edges(node, node+win_xsize, ++i, 2);
              edges[++i].weight = -1.0;
              ++i;
           }
           else if ((node % win_xsize) == win_xsize-1)  /* row tail */
           {
              fill_edges(node, node+win_xsize, i, 1);
              fill_edges(node, node+win_xsize-1, ++i, 3);
              edges[++i].weight = -1.0;
              edges[++i].weight = -1.0;
              ++i;
           }
           else
           {
              fill_edges(node, node+1, i, 1);
              fill_edges(node, node+win_xsize+1, ++i, 1);
              fill_edges(node, node+win_xsize, ++i, 1);
              fill_edges(node, node+win_xsize-1, ++i, 1);
              ++i;
           }
        }
    }
```

```
}

fill_edges(node1, node2, index, offset)
register int node1, node2, index, offset;
{/* store data of a link */
     edges[index].node1 = node1;
     edges[index].node2 = node2;
     edges[index].weight = link_weight(node1, node2);
     edges[index].next = &edges[index + offset];
     edges[index + offset].last = &edges[index];
}

float link_weight(node1, node2)/* compute link weight, squared Euclidean
                                  distance */
int node1, node2;
{
     int i1, j1, i2, j2,
         k;
     float  weight;

     get_coordinates(node1, &i1, &j1);
     get_coordinates(node2, &i2, &j2);
     weight = 0.0;
     for (k = 0; k < dimension; k++)
         weight += SQUARE((float)image[k][i1][j1]-(float)image[k][i2][j2]);

     return(weight);
}

rst_file(vertex, heavy)
LIST_HEAD *vertex;
UNIQUE_LINK **heavy;
{
    char  *buf1 = "/home/image/output/rst";/* default output file */
    FILE *fp;
    int t, numwritten;
    TREE *temp1_tree;

    /* open file in binary mode for read write the tree data */
    fp = fopen(buf1,"w");
    demand(fp, cannot open file for rsst);

    fwrite((char *)&dimension, sizeof(int), 1, fp);
    fwrite((char *)&top_x, sizeof(int), 1, fp);
    fwrite((char *)&top_y, sizeof(int), 1, fp);
    fwrite((char *)&win_xsize, sizeof(int), 1, fp);
    fwrite((char *)&win_ysize, sizeof(int), 1, fp);
    /* write the unique link to file */
    numwritten = 0;
    for (t = 0; t < win_xsize*win_ysize-1; t++)
       numwritten += fwrite((char *)heavy[t], sizeof(int), 2, fp);
    for (t = 0; t < win_xsize*win_ysize; t++)
    {
       temp1_tree = vertex[t].node;
       while (temp1_tree != NULL)
       {  /* write tree to file */
            numwritten += fwrite((char *)temp1_tree, sizeof(TREE), 1, fp);
            temp1_tree = temp1_tree->next;
       }
    }

    demand(numwritten == 4*win_xsize*win_ysize-4, Write error);
    printf("Data write to file %s.\n",buf1);
    fclose(fp);
}

TREE * front_of_tree(new, list)
TREE *new, *list;
{/* store structure in the link list */
     new->next = list;
     list = new;

     return(list);
}

/* main function to do the CEST using Kruskals algorithm */
recru_tree(unique_link, vertex)
LIST_HEAD *vertex;
UNIQUE_LINK *unique_link;
{
    int count, label, biggest, smallest;
    TREE *node;
    STAT *region_sum;
    LINK *lightest_ptr;

    node = (TREE *)get_memory(2 *(win_xsize*win_ysize-1), sizeof(TREE));
    region_sum = (STAT *)get_memory(win_xsize*win_ysize, sizeof(STAT));
    for(count = 0, label = 1; label < win_xsize*win_ysize; label++)
    {
       /* pick the lightest link */
       lightest_ptr = lightest();
```

```
        unique_link[label-1].node1 = lightest_ptr->node1;
        unique_link[label-1].node2 = lightest_ptr->node2;
        unique_link[label-1].weight = lightest_ptr->weight;
        /* put link into the tree */
        node[count].node_pos = lightest_ptr->node2;
        vertex[lightest_ptr->node1].node =
      front_of_tree(&node[count++],vertex[lightest_ptr->node1].node);
        node[count].node_pos = lightest_ptr->node1;
        vertex[lightest_ptr->node2].node =
      front_of_tree(&node[count++],vertex[lightest_ptr->node2].node);
        if (label < win_xsize*win_ysize-1)
        {
       sum_label_nodes(region_sum,vertex,lightest_ptr,label,
                       &biggest,&smallest);
       /* delete all duplicate link and recalculate link weight */
       del_duplicate_recal(vertex,region_sum,label,biggest,smallest);
        }
      }
     free((char *)region_sum);
}

LINK * lightest()
{    /* find the lightest link */
     register LINK *min_ptr, *this;

     min_ptr = head.next;
     this = min_ptr->next;
     while (this != &head)
     {
         if (min_ptr->weight > this->weight)
            min_ptr = this;
         this = this->next;
     }
     /* delete lightest link */
     min_ptr->last->next = min_ptr->next;
     min_ptr->next->last = min_ptr->last;
     min_ptr->next = NULL;  /* status out */

     return(min_ptr);
}

/* compute sum of a region, update label of nodes */
sum_label_nodes(region_sum, vertex, light, tag, biggest, smallest)
STAT *region_sum;
LINK *light;
int tag;
register LIST_HEAD *vertex;
register int *biggest, *smallest;
{
     register int i, done = 0;
     int old_tag[2];

     if (vertex[light->node1].node_tag == 0 &&
         vertex[light->node2].node_tag == 0)
     {     /* both are new node, use a new label */
         vertex[light->node1].node_tag = tag;
         vertex[light->node2].node_tag = tag;
         region_sum[tag] = add_struct(get_pix_value(light->node1),
                                      get_pix_value(light->node2));
         *smallest = light->node1;
         *biggest = light->node2;
     }
     else if (vertex[light->node1].node_tag == 0 &&
              vertex[light->node2].node_tag != 0)
     {
           region_sum[tag] = add_struct(get_pix_value(light->node1),
                                region_sum[vertex[light->node2].node_tag]);
           vertex[light->node1].node_tag = tag;
           old_tag[0] = vertex[light->node2].node_tag;
           for (i = 0; i < win_xsize*win_ysize; i++)
           {     /* update label and count */
               if (vertex[i].node_tag == old_tag[0])
               {
                   vertex[i].node_tag = tag;
                   if (!done)
                   {
                       *smallest = i;
                       ++done;
                   }
                   *biggest = i;
               }
           }
           if (light->node1 < *smallest)   *smallest = light->node1;
     }
     else if (vertex[light->node1].node_tag != 0 &&
              vertex[light->node2].node_tag == 0)
     {
           region_sum[tag] = add_struct(get_pix_value(light->node2),
 region_sum[vertex[light->node1].node_tag]);
           vertex[light->node2].node_tag = tag;
           old_tag[0] = vertex[light->node1].node_tag;
```

```
            for (i = 0; i < win_xsize*win_ysize; i++)
            {     /* update label and count */
                if (vertex[i].node_tag == old_tag[0])
                {
                    vertex[i].node_tag = tag;
                    if (!done)
                    {
                        *smallest = i;
                        ++done;
                    }
                    *biggest = i;
                }
            }
            if (light->node2 > *biggest)   *biggest = light->node2;
        }
        else
        {  /* both node tag != 0 */
            old_tag[0] = vertex[light->node1].node_tag;
            old_tag[1] = vertex[light->node2].node_tag;
            region_sum[tag] =
                add_struct(region_sum[vertex[light->node1].node_tag],
                           region_sum[vertex[light->node2].node_tag]);
            for (i = 0; i < win_xsize*win_ysize; i++)
            {
                if (vertex[i].node_tag == old_tag[0] ||
                    vertex[i].node_tag == old_tag[1])
                {
                    vertex[i].node_tag = tag;
                    if (!done)
                    {
                        *smallest = i;
                        ++done;
                    }
                    *biggest = i;
                }
            }
        }
}

/* delete duplicated links and recalculate sum of a region */
del_duplicate_recal(vertex, region_sum, region, biggest, smallest)
register LIST_HEAD *vertex;
STAT *region_sum;
int region, biggest, smallest;
{
        /* delete all duplicate link connected to external vertices */
        register LINK *find,*this,*that;
        LINK dummy;
        int temp_node, temp_re;

        /* goto starting point */
        this = jump(smallest);
        /* connect the related link in a shorter list */
        this = compress(this, vertex, region, biggest);
        while (this != NULL)
        {  /* first recalculate then delete */
            temp_re = 0; /* initialize for delete */
            if (vertex[this->node1].node_tag == region)
            {  /* find node1 in region */
                temp_node = this->node2;
                if (vertex[temp_node].node_tag != 0) /* node2 is in a region */
                {
                    this->weight = new_weight(region_sum[region],
                                    region_sum[vertex[temp_node].node_tag]);
                    temp_re = vertex[temp_node].node_tag;
                }
                else  /* node2 is a single node */
                    this->weight = new_weight(region_sum[region],
                                              get_pix_value(temp_node));
            }
            else
            {  /* find node2 in region */
                temp_node = this->node1;
                if (vertex[temp_node].node_tag != 0) /* node1 is in a region */
                {
                    this->weight = new_weight(region_sum[region],
             region_sum[vertex[temp_node].node_tag]);
                    temp_re = vertex[temp_node].node_tag;
                }
                else  /* node1 is a single node */
                    this->weight = new_weight(region_sum[region],
                                              get_pix_value(temp_node));
            }
            /* delete duplicate links for all nodes, no link in region */
            that = &dummy;
            dummy.other = this->other;
            if (!temp_re)   /* temp_node is single */
            {
                while ((find = that->other) != NULL)
                {  /* delete duplicate link, temp_node is single */
                    if (find->node1 > temp_node)   break;
```

```
            if (find->node1 == temp_node || find->node2 == temp_node)
            {
                /* delete link form whole list */
                find->last->next = find->next;
                find->next->last = find->last;
                find->next = NULL;
                /* delete link form compressed list */
                that->other = find->other;
                break;  /* only delete one link */
            }
            that = that->other;
         }
      }
      else
      {  /* temp_node is in a region */
         while ((find = that->other) != NULL)
         {  /* delete duplicate link */
            /* only one duplicate between two regions */
            if (vertex[find->node1].node_tag == temp_re ||
                vertex[find->node2].node_tag == temp_re)
            {
                find->last->next = find->next;
                find->next->last = find->last;
                find->next = NULL;
                that->other = find->other;
                break;
            }
            that = that->other;
         }  /* end while */
      }     /* end else */
      this = dummy.other;  /* reconnect and point to next */
   }        /* end while */
}

/* this function is to compress the list of all links to increase
efficiecy */
LINK * compress(this, vertex, region, biggest)
register LINK *this;
register LIST_HEAD *vertex;
register int region, biggest;
{
     LINK *find, *start;

     while (1)
     {   /* find the first link to start */
         if (vertex[this->node1].node_tag == region ||
             vertex[this->node2].node_tag == region)
             break;
         this = this->next;
     }
     start = find = this;
     this = this->next;
     while (this != &head)
     {   /* connect all relate link in a list */
         if (this->node1 > biggest)   break;
         if (vertex[this->node1].node_tag == region ||
             vertex[this->node2].node_tag == region)
         {
             find->other = this;
             find = this;
         }
         this = this->next;
     }
     find->other = NULL;  /* terminate */

     return(start);
}

/* skip unnecessary position */
LINK * jump(smallest)
int smallest;
{
     register int index;

     if (smallest < win_xsize+1) index = 0;
     else                        index = 4*(smallest-win_xsize-1);
     /* at least one undeleted link within range */
     while (edges[index].next == NULL)   ++index;

     /* return the first element in the list */
     return(&edges[index]);
}

float new_weight(new, old)
STAT new, old;
{
     int  k;
     float weight,
           new_mean[DIMENSION],
           old_mean[DIMENSION];
```

```
    for (k = 0; k < dimension; k++)
    {
        new_mean[k] = new.sum[k]/(float)new.num_vec;
        old_mean[k] = old.sum[k]/(float)old.num_vec;
    }
    weight = 0.0;
    for (k = 0; k < dimension; k++)
        weight += SQUARE(new_mean[k]-old_mean[k]);

    return(weight);
}

STAT add_struct(a, b)/* add two structures */
STAT a, b;
{
    int  k;

    a.num_vec += b.num_vec;
    for (k = 0; k < dimension; k++) a.sum[k] += b.sum[k];

    return(a);
}

STAT get_pix_value(node)
register int node;
{
int  i, j, k;
       STAT a;

get_coordinates(node, &i, &j);
       a.num_vec = 1;
for (k = 0; k < dimension; k++)
    a.sum[k] = (float)image[k][i][j];

       return(a);
}

int comp(a, b)/* compare function for qsort() */
UNIQUE_LINK **a, **b;
{    /* in descending order of magnitude */
     if ((*a)->weight > (*b)->weight)      return(-1);
     else if ((*a)->weight < (*b)->weight) return(1);
     else                                  return(0);
}

get_coordinates(node_pos, a, b)/* transform mapping */
register int node_pos;
register int *a, *b;
{
*a = node_pos / win_xsize;/* i */
*b = node_pos % win_xsize;/* j */
}

/*************************************************************************
 *  You are reading mm3.c                                                *
 *  This program performs minimax top-down GTHS.                         *
 *  This program read in a spanning tree, which generated by             *
 *  e.g. rsst3.c, and use minimax to partition the spanning tree.        *
 *  To obtain m segments the m-1 links will be deleted.                  *
 *  There are eight neighbours to each pixels.                           *
 *  The pattern vectors are two dimensional.                             *
 *  ****** OBTAIN SEGMENTS FROM THE INPUT SPANNING TREE FILE ******     *
 *  ****** WRITE THE MINIMAX LINKS TO FILE                   ******     *
 *  ****** THE COST FUNCTION IS THE INTRASET DISTANCE        ******     *
 *  K.S.LAU 21-1-92                                                      *
 *************************************************************************/
#include <malloc.h>
#include <stdio.h>
#include <pixrect/pixrect_hs.h>

#define  DIMENSION          3/* maximum dimension of image */
#define  MAX_XSIZE          128/* image size */
#define  MAX_YSIZE          128  /* image size */

#define  NOTVISIT           0/* label notvisit */
#define  VISIT              1/* label visit */
#define  TMPVISIT           2/* label temporary visit */

#define  SQUARE(x)          ((x)*(x))  /* macro for square */
#define  demand(fact, remark) {\
        if (!(fact)) {\
            fprintf(stderr, "demand not met: fact\n");\
            fprintf(stderr, "remark\n");\
            exit(1);\
        }\
}

#define  FAIL               0    /* minimax link test */
#define  OK                 1

#define  DUPLICATE          1     /* find single node */
```

```
#define  UNIQUE          0

/************************ STRUCTURE DEFINITION ************************/
typedef struct {/* store links which is to be cut */
int node1;
int node2;
}UNIQUE_LINK;

typedef struct {/* store the node of the links which is to be cut */
int mm_node;
float var;
    char root;
}MINIMAX_NODE;

typedef struct tree {/* structure to store the spanning tree */
    int node_pos;
struct tree *next;
char status;
}TREE;   /* memory is allocated in order of declaration */

typedef struct {/* the list of all nodes in a spanning tree */
int node_tag;
struct tree *node;
}LIST_HEAD;

typedef struct {/* statistics of a region */
float sum[DIMENSION];
float ssq[DIMENSION];
int num_vec;
}STAT;

/************************ FUNCTION DEFITIONS ************************/
char *          get_memory();
char *          reget_memory();
STAT            get_tree_variance();
float           get_distance();
TREE *          find_alone_node();
TREE *          depth_first_next();
TREE *          depth_first_attribute();
LIST_HEAD *     depth_first_re_attribute();
LIST_HEAD *     depth_first_link();
LIST_HEAD *  depth_first_pix();
LIST_HEAD *  depth_first_screen();
LIST_HEAD *  depth_first_link_sum_square();

/********************** GLOBAL VARIABLES **********************/
u_char image[DIMENSION][MAX_YSIZE][MAX_XSIZE];

int  dimension,/* dimension of the image */
     top_x, top_y,/* top left coordinates of process window */
     win_xsize, win_ysize;/* size of process window */
int *group_pixel,/* store points in a region */
    *stack;/* stack to do depth first search */
LIST_HEAD *new, *old;/* for depth first search */
struct rasterfile header;/* header for image */

main()
{
    int **clust,/* pointer for start and end of a region */
        s, t,
        segment;
    UNIQUE_LINK *link_intree;
    TREE *tree_element;
    LIST_HEAD *vertex,
              **del_link;
    MINIMAX_NODE **variance,
                 *minimax_node;
    STAT *node_intree;
    FILE *span_tree;

    puts("TOP DOWN MINIMAX SEGMENTATION");
    /* read the spanning tree */
    get_parameters(&span_tree, &segment);
    get_data();/* read the image of the spaning tree */
    number_link();  /* get number of vertices */

    /* allocate array of structure for linked list of spanning tree */
    tree_element = (TREE *)get_memory(2*win_xsize*win_ysize-2, sizeof(TREE));
    vertex = (LIST_HEAD *)get_memory(win_xsize*win_ysize, sizeof(LIST_HEAD));
    /* memory for depth first search */
    stack = (int *)get_memory(win_xsize*win_ysize, sizeof(int));
    group_pixel = (int *)get_memory(win_xsize*win_ysize, sizeof(int));
    /* memory for minimax */
    node_intree = (STAT *)get_memory(win_xsize*win_ysize, sizeof(STAT));
    link_intree = (UNIQUE_LINK *)get_memory(win_xsize*win_ysize-1,
                                            sizeof(UNIQUE_LINK));
    minimax_node = (MINIMAX_NODE *)get_memory(2*(segment-1),
                                              sizeof(MINIMAX_NODE));
    variance = (MINIMAX_NODE **)get_memory(2*(segment-1),
                                           sizeof(MINIMAX_NODE *));
    load_data(&span_tree, link_intree, tree_element, vertex);
```

```
    /* initalize label for depth first search */
    for(s = 0; s < win_xsize*win_ysize; s++)    vertex[s].node_tag = s;
    /* initalize array of pointer to minimax node for sorting */
    for(s = 0; s < 2*(segment - 1); s++)   variance[s] = &minimax_node[s];
    /* do minimax variance */
    puts("Doing minimax ...");
    minimax_variance(vertex, link_intree, node_intree,
                     minimax_node, variance, segment);

    mm_link_file(segment, minimax_node, vertex);

}

get_data()
{
    char  filename[DIMENSION][50];
    int  i, j;
    FILE  *f[DIMENSION];
    struct  rasterfile  head[DIMENSION];

    printf("The spanning tree is on %d dimensional data\n", dimension);

    for (i = 0; i < dimension; i++)
    {
        printf("Enter the channel filename %d, ", i);
        scanf("%s", (char *)&filename[i][0]);
        f[i] = fopen((char *)&filename[i][0], "r");
        demand(f[i], Cannot open file);
    }

    /* store header in global area, for later use */
    fread((char *)&header, sizeof(struct rasterfile), 1, f[0]);
    rewind(f[0]);
    for (i = 0; i < dimension; i++)
    {
        fread((char *)&head[i], sizeof(struct rasterfile), 1, f[i]);
        demand(header.ras_length == head[i].ras_length,
               Make sure the images has the same size and coordinates);
    }

    printf("The images size is %d\n", header.ras_width);
    printf("The window width (xsize) is %d\n", win_xsize);
    printf("The window height (ysize) is %d\n", win_ysize);

    printf("The top left window coordinate of the window,\n");
    printf("\tx: %d\n", top_x);
    printf("\ty: %d\n", top_y);

    demand(head[0].ras_width >= win_xsize+top_x,
           The window is out of range\, please reduce size);

    for (i = 0; i < dimension; i++)
        fseek(f[i], (long)(top_y*head[i].ras_width+top_x), 1);
    for (i = 0; i < dimension; i++)
        for (j = 0; j < win_ysize; j++)
        {
            fread((char *)&image[i][j][0], sizeof(char), win_xsize, f[i]);
            fseek(f[i], (long)(head[i].ras_width-win_xsize), 1);
        }

    for (i = 0; i < dimension; i++) fclose(f[i]);
}

get_parameters(span_tree, segment)
FILE **span_tree;
int *segment;
{
    char buf[50];
    int s;

    printf("Input the (r)sst data file name, ");
    scanf("%s", buf);
    printf("Enter the maximum number of segments to be generated,\n");
    printf("the links are stored in a file to generate segmentation, ");
    scanf("%d", segment);
    /* open file in binary mode for reading the tree data */
    *span_tree = fopen(buf, "r");
    demand(*span_tree, Could not open file for spanning tree);
    printf("Data read from file %s \n", buf);
    fread((char *)&dimension, sizeof(int), 1, *span_tree);
    fread((char *)&top_x, sizeof(int), 1, *span_tree);
    fread((char *)&top_y, sizeof(int), 1, *span_tree);
    fread((char *)&win_xsize, sizeof(int), 1, *span_tree);
    fread((char *)&win_ysize, sizeof(int), 1, *span_tree);
}

load_data(span_tree, heavy, tree_element, vertex)
FILE **span_tree;
UNIQUE_LINK *heavy;
TREE *tree_element;
```

```
LIST_HEAD *vertex;
{
    int s, t,
        total_items,
        numread = 0;
    TREE *temp_tree;

    /* read a spanning tree */
    numread += fread((char *)heavy, sizeof(UNIQUE_LINK),
                     win_xsize*win_ysize-1, *span_tree);
    for(t = 0; t < 2 * win_xsize*win_ysize-2; t++)
        numread += fread((char *)&tree_element[t],
                         sizeof(int)+sizeof(TREE *), 1, *span_tree);
    demand(numread == 3*win_xsize*win_ysize-3, Read error);
    fclose(*span_tree);
    /* reconstruct the tree data structures just read from file */
    for(t = 0, total_items = 0; t < win_xsize*win_ysize; t++, total_items++)
    {
       vertex[t].node = &tree_element[total_items];
       temp_tree = tree_element[total_items].next;
       while(temp_tree != NULL)
       {
             s = total_items++;
             tree_element[s].next = &tree_element[total_items];
             temp_tree = tree_element[total_items].next;
       }
    }
}

/* write the minimax links to a file */
mm_link_file(segment, minimax_node, vertex)
int segment;
MINIMAX_NODE *minimax_node;
LIST_HEAD *vertex;
{
    char  *buf2 = "/home/image/output/mmlink";
    FILE *mm_link;
    TREE *temp_tree;
    int s, numread, total_link;

    mm_link = fopen(buf2, "w");
    demand(mm_link, Could not open file for minimax link);

    fwrite((char *)&dimension, sizeof(int), 1, mm_link);
    fwrite((char *)&top_x, sizeof(int), 1, mm_link);
    fwrite((char *)&top_y, sizeof(int), 1, mm_link);
    fwrite((char *)&win_xsize, sizeof(int), 1, mm_link);
    fwrite((char *)&win_ysize, sizeof(int), 1, mm_link);
    numread = 0;
    total_link = segment-1;
    numread += fwrite((char *)&total_link, sizeof(int), 1, mm_link);
    for(s = 0; s < 2*(segment-1); s++)
        numread += fwrite((char *)&minimax_node[s], sizeof(int), 1, mm_link);
    for(s = 0; s < win_xsize*win_ysize; s++)
    {
       temp_tree = vertex[s].node;
       while(temp_tree != NULL)
       {  /* write tree to file */
             numread += fwrite((char *)temp_tree,
                       sizeof(int)+sizeof(TREE *), 1, mm_link);
             temp_tree = temp_tree->next;
       }
    }
    demand(numread == 2*(win_xsize*win_ysize+segment)-3,
           Minimax link data write error);
    printf("Minimax link file is %s\n", buf2);
    fclose(mm_link);
}

char *get_memory(items, size)
unsigned items, size;
{
     char *buffer;

     buffer = (char *)calloc(items, size);
     demand(buffer, Nothing allocated for array);

     return(buffer);
}

int number_link()
{
    int s, total_link;

    total_link = 0;
    for (s = 1; s < win_xsize - 1; s++)
        total_link += s;
    total_link *= 4;
    total_link += 2 * (win_xsize*win_ysize - 1);
    printf("Total number of links in original graph is %d.\n",total_link);
    return (total_link);
```

```
}

/* label the root for depth first search */
visit_root(vertex, node)
LIST_HEAD *vertex;
register int *node;
{
int i,j;
register TREE *this;

        for (i = 0,j = 1; i < 2; i++,j--)
{
            this = vertex[node[i]].node;
            while(this != NULL)
            {
 if(this->node_pos == node[j])
 {
    this->status = VISIT;
    break;
 }
 this = this->next;
            }
}
}

/* reset label after depth first search */
reset_tree_tmp(vertex, start, numpix)
LIST_HEAD *vertex;
int numpix, *start;
{
register int i, index;
register TREE *this_node;

        index = start - group_pixel;
        for (i = 0; i < numpix; i++, index++)
{
            this_node = vertex[group_pixel[index]].node;
            while(this_node != NULL)
            {
                 if (this_node->status == TMPVISIT)
                     this_node->status = NOTVISIT;
 this_node = this_node->next;
            }
}
}

/* index mapping */
get_coordinates(node_pos, a, b)
register int node_pos;
register int *a, *b;
{
*a = node_pos / win_xsize;
*b = node_pos % win_xsize;
}

int comp(a, b)/* for qsort() compare */
MINIMAX_NODE **a, **b;
{
     if ((*a)->var > (*b)->var)             return(-1);
     else if ((*a)->var < (*b)->var)        return(1);
     else             return(0);
}

/* compute the statistics of a subtree, sum and sum square */
int get_link_sum_square(vertex, root_node, link_intree, node_intree)
register LIST_HEAD *vertex;
UNIQUE_LINK *link_intree;
STAT *node_intree;
int root_node;
{
    int total_link = 0, stack_pos = 0, num_vec = 0;
    TREE *root;
    register LIST_HEAD *temp_list;

    /* write the link in tree to link_intree tmpfile */
    if ((root = find_alone_node(&vertex[root_node])) == NULL)
       return(1);  /* subtree is a single node */
    start_depth_first(&stack_pos,&num_vec,vertex[root_node].node_tag,root);
    link_intree[total_link].node1 = vertex[root_node].node_tag;
    link_intree[total_link++].node2 = root->node_pos;
    new = &vertex[root->node_pos];
    old = &vertex[root_node];
    do
    {  /* depth first next */
       temp_list = depth_first_link_sum_square(vertex,link_intree,
                        node_intree,&num_vec,&total_link,&stack_pos);
       old = new;
       new = temp_list;
    } while(stack_pos != 1);  /* stack != NULL */
    reset_tree_tmp(vertex,group_pixel,num_vec);
    return(num_vec);
```

```
}

LIST_HEAD * depth_first_link_sum_square(vertex, link_intree, node_intree,
                                        num_vec, num_link, stack_pos)
LIST_HEAD *vertex;
UNIQUE_LINK *link_intree;
STAT *node_intree;
register int *num_link, *stack_pos, *num_vec;
{    /*********  write all link in tree to file  *********/
    TREE *root;

    if((root = depth_first_attribute(vertex,node_intree,stack_pos)) == NULL)
        return(NULL);
    stack[(*stack_pos)++] = root->node_pos; /* put node to stack */
    group_pixel[(*num_vec)++] = root->node_pos;  /* put node for reset */
    root->status = TMPVISIT;
    /* record link_intree */
    link_intree[*num_link].node1 = new->node_tag;
    link_intree[(*num_link)++].node2 = root->node_pos;
    return(&vertex[root->node_pos]);
}

TREE * depth_first_attribute(vertex, node_intree, stack_pos)
register LIST_HEAD *vertex;
register int *stack_pos;
STAT *node_intree;
{
    TREE *root;
    LIST_HEAD *last;

    /* delete the duplicate node in the next tree list */
    delete_duplicate();
    if ((root = find_alone_node(new)) == NULL)
    {  /* one of the end */
       last = new;  /* take down the last location */
       self_attribute(new, node_intree);  /* the end node */
       while (root == NULL)
       {     /* delete stack */
             *stack_pos -= 2;
             new = &vertex[stack[(*stack_pos)++]];
             root = find_alone_node(new);
             /* no branch */
             if (root == NULL)   full_attribute(new, last, node_intree);
             /* has at least one branch */
             else                half_attribute(new, last, node_intree);
             if (*stack_pos == 1 && root == NULL)    return(NULL);
             last = new;
       }
    }
    return(root);
}

/* compute statistics of a subtree */
half_attribute(this, last, node_intree)
LIST_HEAD *this, *last;
STAT *node_intree;
{
     int  k;

     for (k = 0; k < dimension; k++)
     {
         node_intree[this->node_tag].sum[k]
                        += node_intree[last->node_tag].sum[k];
         node_intree[this->node_tag].ssq[k]
                        += node_intree[last->node_tag].ssq[k];
     }
     node_intree[this->node_tag].num_vec
                        += node_intree[last->node_tag].num_vec;
}

/* compute statistics of a subtree */
full_attribute(this, last, node_intree)
LIST_HEAD *this, *last;
STAT *node_intree;
{
     self_attribute(this, node_intree);
     half_attribute(this, last, node_intree);
}

/* compute statistics of a subtree */
self_attribute(this, node_intree)
LIST_HEAD * this;
STAT *node_intree;
{
     int  i, j, k;

     get_coordinates(this->node_tag, &i, &j);
     for (k = 0; k < dimension; k++)
     {
         node_intree[this->node_tag].sum[k] += (float)image[k][i][j];
         node_intree[this->node_tag].ssq[k] += SQUARE((float)image[k][i][j]);
```

```
    }
    node_intree[this->node_tag].num_vec++;
}

/* reset label after depth first search */
reset_subtree(node_intree, minimax_node, node, num_vec, start)
STAT *node_intree;
MINIMAX_NODE *minimax_node;
int node, num_vec, *start[];
{    /* node is the index of mm_node just obtained */
     /* tree root is the root of this subtree */
     register int i;

     /* for all pixel in the whole subtree */
     for (i = 0; i < num_vec; i++)
     {   /* linear search */
 if (group_pixel[i] == minimax_node[node].mm_node)
 {  /* find pixel in min subtree */
    start[0] = &group_pixel[i];
    start[1] = &group_pixel[i + node_intree[group_pixel[i]].num_vec];
    break;
 }
     }
     reset_node_intree(node_intree, group_pixel,
                       (int)(start[0] -group_pixel));
     reset_node_intree(node_intree, start[1],
                       num_vec -(int)(start[1] -group_pixel));
}

/* reset buffer after depth first search */
reset_node_intree(node_intree, start, items)
STAT *node_intree;
int *start, items;
{
     register int i, k, index;

     index = start - group_pixel;
     for (i = 0; i < items; i++, index++)
     {
         node_intree[group_pixel[index]].num_vec = 0;
         for (k = 0; k < dimension; k++)
         {
     node_intree[group_pixel[index]].sum[k] = 0.0;
     node_intree[group_pixel[index]].ssq[k] = 0.0;
 }
     }
}

re_attribute(vertex, root_node, node_intree, start, num_vec)
register LIST_HEAD *vertex;
STAT *node_intree;
int num_vec, root_node, *start[];
{
    int stack_pos = 0;
    TREE *root;
    register LIST_HEAD *temp_list;

    /* write the link in tree to link_intree tmpfile */
    if ((root = find_alone_node(&vertex[root_node])) == NULL)
    {
   self_attribute(&vertex[root_node], node_intree);
   return;  /* subtree is a single node */
    }
    stack[stack_pos++] = root_node;       /* put root to stack */
    stack[stack_pos++] = root->node_pos;  /* put node to stack */
    root->status = TMPVISIT;              /* visit node */
    new = &vertex[root->node_pos];
    old = &vertex[root_node];
    do
    {  /* depth first next */
       temp_list = depth_first_re_attribute(vertex,node_intree,&stack_pos);
       old = new;
       new = temp_list;
    } while(stack_pos != 1);  /* stack != NULL */
    reset_tree_tmp(vertex,group_pixel,(int)(start[0] -group_pixel));
    reset_tree_tmp(vertex,start[1],num_vec -(int)(start[1] -group_pixel));
}

LIST_HEAD * depth_first_re_attribute(vertex, node_intree, stack_pos)
LIST_HEAD *vertex;
STAT *node_intree;
register int *stack_pos;
{    /*********  write all link in tree to file  *********/
TREE *root;

    if((root = depth_first_attribute(vertex,node_intree,stack_pos)) == NULL)
        return(NULL);
stack[(*stack_pos)++] = root->node_pos; /* put node to stack */
    root->status = TMPVISIT;
return(&vertex[root->node_pos]);
}
```

```
int get_link(vertex, root_node, link_intree)
register LIST_HEAD *vertex;
UNIQUE_LINK *link_intree;
int root_node;
{
    int total_link = 0, stack_pos = 0, num_vec = 0;
    TREE *root;
    register LIST_HEAD *temp_list;

    if ((root = find_alone_node(&vertex[root_node])) == NULL)
       return(1);  /* subtree is a single node */
    start_depth_first(&stack_pos,&num_vec,vertex[root_node].node_tag,root);
    link_intree[total_link].node1 = vertex[root_node].node_tag;
    link_intree[total_link++].node2 = root->node_pos;
    new = &vertex[root->node_pos];
    old = &vertex[root_node];
    do
    {  /* depth first next */
       temp_list = depth_first_link(vertex,link_intree,&num_vec,
                                &total_link,&stack_pos);
       old = new;
       new = temp_list;
    } while(stack_pos != 1);  /* stack != NULL */
    reset_tree_tmp(vertex,group_pixel,num_vec);
    return(num_vec);
}

LIST_HEAD * depth_first_link(vertex, link_intree, num_vec,
                             num_link, stack_pos)
LIST_HEAD *vertex;
UNIQUE_LINK *link_intree;
register int *num_link, *stack_pos, *num_vec;
{    /*********  write all link in tree to file  *********/
    TREE *root;

    if ((root = depth_first_next(vertex,stack_pos)) == NULL)
       return(NULL);
    stack[(*stack_pos)++] = root->node_pos; /* put node to stack */
    group_pixel[(*num_vec)++] = root->node_pos;  /* put node for reset */
    root->status = TMPVISIT;
    /* record link_intree */
    link_intree[*num_link].node1 = new->node_tag;
    link_intree[(*num_link)++].node2 = root->node_pos;
    return(&vertex[root->node_pos]);
}

/* main function to do minimax */
minimax_variance(vertex, link_intree, node_intree, minimax_node,
                 variance, segment)
LIST_HEAD *vertex;
STAT *node_intree;
MINIMAX_NODE *minimax_node, **variance;
UNIQUE_LINK *link_intree;
int segment;
{
    int s, t, num_vec, swap;
    static int *start[2];

    /* choose node 0 arbitrarily as the root for the diagraph */
    num_vec = get_link_sum_square(vertex, 0, link_intree, node_intree);
    swap = minimaxon_tree(vertex, link_intree, node_intree,
                          minimax_node, num_vec, 0, 0);
    /* finally all mm_node are the tree_root */
    if (swap)
    {  /* always reset and reattribute the upper subtree */
       reset_subtree(node_intree, minimax_node, 0, num_vec, start);
       re_attribute(vertex, minimax_node[1].mm_node, node_intree,
                    start, num_vec);
    }
    else
    {
       reset_subtree(node_intree, minimax_node, 1, num_vec, start);
       re_attribute(vertex, minimax_node[0].mm_node, node_intree,
                    start, num_vec);
    }
    for (t = 2; t < 2*(segment - 1); t += 2)  /* number of minimax link */
    {  /* number of tree equal number of minimax link + 1 = t/2 +1 */
       /* choose the link which minimize the variance of the max var tree */
       /* sort variance */
       qsort((char *)variance, t, sizeof(MINIMAX_NODE *), comp);
       /* take the true root with highest variance, root label = 1 */
       for (s = 0; ; s++)   if (variance[s]->root)   break;
   num_vec = get_link(vertex, variance[s]->mm_node, link_intree);
       if (num_vec == 1)
       {  /* the max var subtree is a single node */
          printf("All %d subtrees are homogeneous.\n", t/2 +1);
          return;   /* return to write mm_file */
       }
       swap = minimaxon_tree(vertex, link_intree, node_intree, minimax_node,
                             num_vec, variance[s]->mm_node, t);
```

```
        if (swap)
        {
           reset_subtree(node_intree, minimax_node, t, num_vec, start);
           re_attribute(vertex, minimax_node[t+1].mm_node, node_intree,
                        start, num_vec);
        }
        else
        {
           reset_subtree(node_intree, minimax_node, t+1, num_vec, start);
           re_attribute(vertex, minimax_node[t].mm_node, node_intree,
                        start, num_vec);
        }
        readjust_tree_var(minimax_node, t, start, num_vec, swap);
    }
}

/* this is the important bit */
int minimaxon_tree(vertex, link_intree, node_intree,
                   minimax_node, num_vec, tree_root, index)
LIST_HEAD *vertex;
UNIQUE_LINK *link_intree;
MINIMAX_NODE *minimax_node;
STAT *node_intree;
int index, tree_root, num_vec;
{
    register int i;
    int j, swap, exchange;
    UNIQUE_LINK dummy_link;
    MINIMAX_NODE link1[2],link2[2];
    MINIMAX_NODE *dummy_ptr[2],*minimax_ptr[2],*temp_ptr[2];
    STAT dummy;

    dummy_ptr[0] = &link1[0]; dummy_ptr[1] = &link1[1];
    minimax_ptr[0] = &link2[0]; minimax_ptr[1] = &link2[1];
    for (i = j = 0; i < num_vec - 1; i++)
    {   /* number of link in subtree */
        dummy = get_tree_variance(link_intree, node_intree, tree_root, i);
        if (dummy.sum[0] > dummy.ssq[0])  /* dummy.v_sum belongs to node2 */
        {
           /* intraset distance of subtree 1 */
           dummy_ptr[1]->var = dummy.ssq[0];
           /* intraset distance of subtree 2 */
           dummy_ptr[0]->var = dummy.sum[0];
           dummy_ptr[1]->mm_node = link_intree[i].node1;
           dummy_ptr[0]->mm_node = link_intree[i].node2;
           dummy_ptr[1]->root = 1;
           dummy_ptr[0]->root = 1;
           /* link_intree node1 is the upper subtree, node2 is the lower */
           swap = 1;   /* node1 and node2 exchange */
        }
        else
        {
           dummy_ptr[0]->var = dummy.ssq[0];
           dummy_ptr[1]->var = dummy.sum[0];
           dummy_ptr[0]->mm_node = link_intree[i].node1;
           dummy_ptr[1]->mm_node = link_intree[i].node2;
           dummy_ptr[0]->root = 1;
           dummy_ptr[1]->root = 1;
           swap = 0;
        }

        if (j == 0)
        {    /* initialize comparison */
            *minimax_ptr[0] = *dummy_ptr[0];
            *minimax_ptr[1] = *dummy_ptr[1];
            if (swap)    exchange = 1;
            else         exchange = 0;
            j++;
        }
        if (dummy_ptr[0]->var < minimax_ptr[0]->var)
        /* find minimax var */
        {  /* find minimum, all var1 are maximum */
            if (swap)    exchange = 1;
            else         exchange = 0;
            temp_ptr[0] = minimax_ptr[0];      /* exchange pointer */
            temp_ptr[1] = minimax_ptr[1];
            minimax_ptr[0] = dummy_ptr[0];
            minimax_ptr[1] = dummy_ptr[1];
            dummy_ptr[0] = temp_ptr[0];
            dummy_ptr[1] = temp_ptr[1];
        }
    }
    dummy_link.node1 = minimax_ptr[0]->mm_node;
    dummy_link.node2 = minimax_ptr[1]->mm_node;
    /* write to minimax_node for sorting */
    minimax_node[index] = *minimax_ptr[0];
    minimax_node[index+1] = *minimax_ptr[1];
    /* label the true root */
    adjust_mm_node(minimax_node, num_vec, index);
    /* visit minimax root for depth first screen */
    visit_root(vertex,(int *)&dummy_link);
```

```
    return(exchange);
}

adjust_mm_node(minimax_node, num_vec, num_mm)
MINIMAX_NODE *minimax_node;
int num_vec, num_mm;
{
     register i, j;

     for (i = 0; i < num_mm; i++)
     {   /* exclude the latest pair of mm_node */
         for (j = 0; j < num_vec; j++)
         if (group_pixel[j] == minimax_node[i].mm_node)
         {  /* only one mm_node in a subtree can be the root */
            minimax_node[i].root = 0;
            break;
         }
     }
}

STAT get_tree_variance(link_intree, node_intree, tree_root, i)
UNIQUE_LINK *link_intree;
STAT *node_intree;
int tree_root, i;
{   /**** calculate the variance of subtrees ****/
    int  k;
    STAT sum[2], result;

    if (node_intree[tree_root].num_vec == 2)
    {   /* only one link in tree */
        for (k = 0; k < dimension; k++)
            result.sum[k] = result.ssq[k] = 0.0;
        return(result);
    }
    /* node2 is the lower subtree */
    sum[0] = node_intree[link_intree[i].node2];  /* must be node2 */
    /* get variance of the other tree */
    sum[1].num_vec = node_intree[tree_root].num_vec-sum[0].num_vec;
    for (k = 0; k < dimension; k++)
    {
        sum[1].sum[k] = node_intree[tree_root].sum[k]-sum[0].sum[k];
        sum[1].ssq[k] = node_intree[tree_root].ssq[k]-sum[0].ssq[k];
    }

    result.ssq[0] = get_distance(sum[1]);
    result.sum[0] = get_distance(sum[0]);

    return(result);  /* return the intraset distance of subtrees */
}

float get_distance(sub)
STAT sub;
{
     int  k;
     float  var = 0.0;

     if (sub.num_vec == 1) return(0.0);/* if only one node variance is zero */

     /* calculate variances of the tree, on both dimension */
     for (k = 0; k < dimension; k++)
         var += (((float)sub.num_vec * sub.ssq[k])
               -SQUARE(sub.sum[k]))/((float)sub.num_vec
               *((float)sub.num_vec-1));

     /* squared intraset distance */
     return(2*var);
}

readjust_tree_var(minimax_node, t, start, num_vec, swap)
MINIMAX_NODE *minimax_node;
int t, swap, num_vec, *start[];
{    /**** update the minimax links variance ****/
     if (swap)
     {  /* first two call the upper subtree, last call the lower one */
        renew_variance(minimax_node, group_pixel,
                       (int)(start[0] -group_pixel), t+1, t);
        renew_variance(minimax_node, start[1],
                       num_vec -(int)(start[1] -group_pixel), t+1, t);
        renew_variance(minimax_node, start[0],
                       (int)(start[1] -start[0]), t, t);
     }
     else
     {
        renew_variance(minimax_node, group_pixel,
                       (int)(start[0] -group_pixel), t, t);
        renew_variance(minimax_node, start[1],
                       num_vec -(int)(start[1] -group_pixel), t, t);
        renew_variance(minimax_node, start[0], (int)(start[1] -start[0]),
                       t+1, t);
     }
```

```
}

renew_variance(minimax, start, numpix, node, t)
register MINIMAX_NODE *minimax;
int numpix, t, *start;
register int node;
{
    register int k, index;
    int j;

    for (j = 0; j < t; j++)
    {  /* for all minimax_node */
        index = start - group_pixel;
        for (k = 0; k < numpix; k++, index++)
        {   /* for all node in the present subtree */
            /* check the present of other minimax node  */
            if (minimax[j].mm_node == group_pixel[index]) /* node1 */
            {  /* node is node1 or node2 */
               minimax[j].var = minimax[node].var;
               break;
            }
        }
    }
}

TREE * find_alone_node(list)
LIST_HEAD *list;
{
     register TREE *this;

     this = list->node;
     while(this != NULL)
     {
   if (this->status == NOTVISIT)
      return(this);
   this = this->next;
     }
     return(NULL);
}

delete_duplicate()
{
register TREE *this;

this = new->node;
while(this != NULL)
{  /* mark the duplicate node */
   if(this->node_pos == old->node_tag)
   {
            this->status = TMPVISIT;
              break;
   }
   this = this->next;
}
}

start_depth_first(stack_pos, numpix, root, node)
register int *stack_pos, *numpix;
int root;
TREE *node;
{
     stack[(*stack_pos)++] = root;    /* put root to stack */
     group_pixel[(*numpix)++] = root;  /* put root to cluster */
     stack[(*stack_pos)++] = node->node_pos;  /* put node to stack */
     group_pixel[(*numpix)++] = node->node_pos;  /* put node to cluster */
     node->status = TMPVISIT;  /* visit node */
}

TREE * depth_first_next(vertex, stack_pos)
register LIST_HEAD *vertex;
register int *stack_pos;
{
    TREE *root;

    delete_duplicate();
    /* delete the duplicate node in the next tree list */
    if((root = find_alone_node(new)) == NULL)
    {
       while(root == NULL)
       {
           /* delete stack */
           *stack_pos -= 2;
           new = &vertex[stack[(*stack_pos)++]];
           root = find_alone_node(new);
           if(*stack_pos == 1 && root == NULL)    return(NULL);
       }
    }
    return(root);
}

/*************************************************************************
```

```
 *  You are reading tdseg3.c                                              *
 *  This program read a spanning tree and its links generated by          *
 *  e.g. rsst3.c or mm3.c then generated segments and write segments      *
 *  to a file for clustering.                                             *
 *  To obtain m segments the heaviest m-1 links will be deleted.          *
 *  There are eight neighbours to each pixels.                            *
 *  ****** OBTAIN SEGMENTS FROM THE INPUT DATA FILE      *****            *
 *  ****** USING (R)SST OR MINIMAX_LINK                  *****            *
 *  K.S.LAU 21-1-92                                                       *
 ************************************************************************/

#include <malloc.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <pixrect/pixrect_hs.h>

#define  DIMENSION          3/* maximum demension of image */
#define  MAX_XSIZE        128   /* maximum xsize of image */
#define  MAX_YSIZE        128   /* maximum ysize of image */

#define  TMPVISIT           2     /* depth first search */
#define  VISIT              1
#define  NOTVISIT           0

#define  DUPLICATE          1     /* find single node */
#define  UNIQUE             0
#define  demand(fact, remark) {\
        if (!(fact)) {\
            fprintf(stderr, "demand not met: fact\n");\
            fprintf(stderr, "remark\n");\
            exit(1);\
        }\
}

/*************** STRUCTURE DECLARATION ***************/
typedef struct tree {/* structure for tree elements */
int node_pos;
struct tree *next;
char status;
} TREE;

typedef struct {/* the first element in a link list */
int node_tag;
TREE *node;
} LIST_HEAD;

typedef struct {/* store a link */
int node1;
int node2;
} UNIQUE_LINK;

/*************** FUNCTIONS DEFINITIONS ***************/
char *          get_memory();
TREE *          find_alone_node();
LIST_HEAD *  depth_first_segment();
LIST_HEAD *  depth_first_inform();
TREE *          depth_first_next();

/****************** GOLBAL VARIABLES ******************/
u_char image[DIMENSION][MAX_YSIZE][MAX_XSIZE];
u_char seg[DIMENSION][MAX_YSIZE][MAX_XSIZE];
int nos_vertices,/* number of nodes in a region */
    dimension,/* dimension of image */
    top_x, top_y,/* top left coordinate of window */
    win_xsize, win_ysize;/* size of window */
float mean[DIMENSION];/* mean of a region */
LIST_HEAD *new_ptr, *old_ptr;/* for depth first search */
struct rasterfile header;/* header for file */

main()
{
    int **clust, *group_pixel,/* storage of segments */
        *stack;/* for depth first search */
    int s, numlink, segment;
    UNIQUE_LINK *heavy;/* links to be cut */
    LIST_HEAD *vertex, **heaviest;
    TREE *tree_element;
    FILE *span_tree;

    puts("GENERATES SEGMENTS FOR TOP DOWN APPROACHES");

    get_parameter(&span_tree, &segment, &numlink);
    get_data();
    number_link();

    /* allocate array of structure for linked list of tree */
    vertex = (LIST_HEAD *)get_memory(win_xsize*win_ysize, sizeof(LIST_HEAD));
    heavy = (UNIQUE_LINK *)get_memory(win_xsize*win_ysize-1,
                                      sizeof(UNIQUE_LINK));
    tree_element = (TREE *)get_memory(2*win_xsize*win_ysize-2, sizeof(TREE));
```

```
    stack = (int *)get_memory(win_xsize*win_ysize, sizeof(int));
    group_pixel = (int *)get_memory(win_xsize*win_ysize+1, sizeof(int));
    /* allocate array of pointer to the segment lists */
    clust = (int **)get_memory(segment+1, sizeof(int *));
    clust[segment] = &group_pixel[win_xsize*win_ysize]; /* assign the end */

    load_data(&span_tree, heavy, tree_element, vertex, numlink);

    /* initalize label for depth first search */
    for(s = 0; s < win_xsize*win_ysize; s++)    vertex[s].node_tag = s;
    /* allocate array of pointer to the heaviest roots */
    heaviest = (LIST_HEAD **)get_memory(2*(segment-1), sizeof(LIST_HEAD *));

    get_root(heaviest, vertex, heavy, segment);
    get_segment(heaviest, vertex, stack, group_pixel, clust, segment);
    segment_file(clust, segment);
    image_file();  /* write output image to file */

}
/************************** END OF MAIN **************************/

get_data()
{
    char  filename[DIMENSION][50];
    int  i, j;
    FILE  *f[DIMENSION];
    struct  rasterfile  head[DIMENSION];

    printf("The spanning tree is on %d dimensional data\n", dimension);

    for (i = 0; i < dimension; i++)
    {
        printf("Enter the channel filename %d, ", i);
        scanf("%s", (char *)&filename[i][0]);
        f[i] = fopen((char *)&filename[i][0], "r");
        demand(f[i], Cannot open file);
    }

    /* store header in global area, for later use */
    fread((char *)&header, sizeof(struct rasterfile), 1, f[0]);
    rewind(f[0]);
    for (i = 0; i < dimension; i++)
    {
        fread((char *)&head[i], sizeof(struct rasterfile), 1, f[i]);
        demand(header.ras_length == head[i].ras_length,
               Make sure the images has the same size and coordinates);
    }

    printf("The images size is %d\n", header.ras_width);
    printf("The window width (xsize) is %d\n", win_xsize);
    printf("The window height (ysize) is %d\n", win_ysize);

    printf("The top left window coordinate of the window,\n");
    printf("\tx: %d\n", top_x);
    printf("\ty: %d\n", top_y);

    demand(head[0].ras_width >= win_xsize+top_x,
           The window is out of range\, please reduce size);

    for (i = 0; i < dimension; i++)
        fseek(f[i], (long)(top_y*head[i].ras_width+top_x), 1);
    for (i = 0; i < dimension; i++)
        for (j = 0; j < win_ysize; j++)
        {
            fread((char *)&image[i][j][0], sizeof(char), win_xsize, f[i]);
            fseek(f[i], (long)(head[i].ras_width-win_xsize), 1);
        }

    for (i = 0; i < dimension; i++) fclose(f[i]);
}

/* store the segments in a file */
segment_file(clust, segment)
int  **clust, segment;
{
     char *buf = "/home/image/output/td.seg";
     int  s, num_pix, item;
     FILE *fp;

     fp = fopen(buf, "w");
     demand(fp, Cannot open file for segment file);

     fwrite((char *)&dimension, sizeof(int), 1, fp);
     fwrite((char *)&top_x, sizeof(int), 1, fp);
     fwrite((char *)&top_y, sizeof(int), 1, fp);
     fwrite((char *)&win_xsize, sizeof(int), 1, fp);
     fwrite((char *)&win_ysize, sizeof(int), 1, fp);
     item = fwrite((char *)&segment, sizeof(int), 1, fp);
     for (s = 0; s < segment; s++)
     {   /* write the segment to file, format is num_seg, seg.... */
         num_pix = clust[s+1] - clust[s];
```

```
            item += fwrite((char *)&num_pix, sizeof(int), 1, fp);
            item += fwrite((char *)clust[s], sizeof(int), num_pix, fp);
        }
        demand(item == win_xsize*win_ysize+segment+1, Write segment error);
        printf("Segments are stored in file %s\n", buf);
        fclose(fp);
}

char *get_memory(items, size)
int items, size;
{
        char *buffer;

        buffer = (char *)calloc((u_int)items, (u_int)size);
        demand(buffer, Nothing allocated for array);

        return(buffer);
}

/* store the segmented image */
image_file()
{   /* only write the visible image */
        FILE *fp;
        int item, i, a;
        char *path = "/home/image/output/seg",
             c[5],
             buf[50];

        header.ras_height = win_ysize;
        header.ras_width = win_xsize;
        header.ras_length = win_xsize*win_ysize;
        for (a = 0; a < dimension; a++)
        {
            strcpy(buf, path);
            strcpy(c, l64a((long)(a+2)));
            strcat(buf, c);
            strcat(buf, ".ras");
            fp = fopen(buf, "w");
            demand(fp, Cannot open file for output image);

            item = fwrite((char *)&header, sizeof(struct rasterfile), 1, fp);
            for (i = 0; i < win_ysize; i++)
                item += fwrite((char *)&seg[a][i][0],
                               sizeof(char), win_xsize, fp);
            demand(item == win_xsize*win_ysize+1, Write output file error);
            fclose(fp);
            printf("Segment image is stored in %s\n", buf);
        }
}

initialize()
{
        int  k;

        nos_vertices = 0;
        for (k = 0; k < dimension; k++) mean[k] = 0.0;
}

calculate_sum(a)
int a;
{
int i, j, k;

get_coordinates(a,&i,&j);
for (k = 0; k < dimension; k++)
    mean[k] += (float)image[k][i][j];
}

get_coordinates(node_pos, a, b)
int node_pos;
int *a, *b;
{
*a = node_pos / win_xsize;
*b = node_pos % win_xsize;
}

get_parameter(span_tree, segment, numlink)
FILE **span_tree;
int *segment, *numlink;
{
    char buf[50];
    int  pos,
         tot,
         length;

    printf("Enter the spanning tree filename, ");
    scanf("%s", buf);
    *span_tree = fopen(buf, "r");
    demand(*span_tree, cannot open file);
    fseek(*span_tree, 0L, 2);
    pos = ftell(*span_tree);
```

```
    printf("spanning tree filelength %d bytes\n", pos);
    rewind(*span_tree);
    fread((char *)&dimension, sizeof(int), 1, *span_tree);
    fread((char *)&top_x, sizeof(int), 1, *span_tree);
    fread((char *)&top_y, sizeof(int), 1, *span_tree);
    fread((char *)&win_xsize, sizeof(int), 1, *span_tree);
    fread((char *)&win_ysize, sizeof(int), 1, *span_tree);

    tot = win_xsize*win_ysize;
    length = (tot-1)*sizeof(UNIQUE_LINK)
           +(2*tot-2)*(sizeof(int)+sizeof(TREE *))
           +5*sizeof(int);
    if (pos == length)
    {/* use only spanning tree */
       *numlink = 0;
       printf("Enter the number of segments to be generated, ");
    }
    else
    {/* use minimax */
        fread((char *)numlink, sizeof(int), 1, *span_tree);
        printf("Input the number of segments, must be <= %d, ", *numlink + 1);
    }
    scanf("%d", segment);
}

int number_link()
{
int s, total_link;

total_link = 0;
for (s = 1; s < win_xsize-1; s++)
total_link += s;
total_link *= 4;
total_link += 2*(win_xsize*win_ysize-1);
printf("Total number of links in original graph is %u\n",total_link);
return (total_link);
}

load_data(span_tree, heavy, tree_element, vertex, numlink)
FILE **span_tree;
UNIQUE_LINK *heavy;
TREE *tree_element;
LIST_HEAD *vertex;
int numlink;
{
    int s, t, total_items, numread = 0;
    TREE *temp_tree;

    if (numlink)  /* this is a minimax file */
       numread += fread((char *)heavy, sizeof(UNIQUE_LINK),
                        numlink, *span_tree);
    else
       numread += fread((char *)heavy, sizeof(UNIQUE_LINK),
                        win_xsize*win_ysize-1, *span_tree);
    for (t = 0; t < 2*win_xsize*win_ysize-2; t++)
        numread += fread((char *)&tree_element[t],
                         sizeof(int)+sizeof(TREE *), 1, *span_tree);
    demand(numlink ? numread == 2*win_xsize*win_ysize+numlink-2
          : numread == 3*win_xsize*win_ysize-3, Read error);
    fclose(*span_tree);

    /* reconstruct the tree data structures just read from file */
    for(t = 0, total_items = 0; t < win_xsize*win_ysize; t++, total_items++)
    {
       vertex[t].node = &tree_element[total_items];
       temp_tree = tree_element[total_items].next;
       while (temp_tree != NULL)
       {
           s = total_items++;
           tree_element[s].next = &tree_element[total_items];
           temp_tree = tree_element[total_items].next;
       }
    }
}

/* prepare for depth first search */
get_root(heaviest, vertex, heavy, segment)
LIST_HEAD **heaviest, *vertex;
UNIQUE_LINK *heavy;
int segment;
{/* locate all the root node in the list head */
register int s, t;

for(s = 0,t = 0; s < segment - 1; s++)
{  /* take down the required 2*(segment-1) heaviest link(root) */
   heaviest[t++] = &vertex[heavy[s].node1];
   /* do the duplicate link */
   heaviest[t++] = &vertex[heavy[s].node2];
   /* label the root as visit */
   visit_root(heaviest,t-2);
}
```

```
}

/* cut spanning tree and obtain segments */
get_segment(heaviest, vertex, stack, group_pixel, clust, segment)
LIST_HEAD *vertex, **heaviest;
int *stack, *group_pixel, **clust;
int segment;
{
    LIST_HEAD *temp_list;
    TREE *root;
    int k, t, stack_pos, root_pos, numpixel;

    numpixel = 0;   /* all root already visited */
    t = get_single_node(heaviest, 2*(segment-1), group_pixel,
                    clust, &numpixel);
    if (t > 0)
    {
        printf("There are %d single point segments\n", t);
        output_single_node(group_pixel, clust, t);
    }
    for (root_pos = 0; t < segment; root_pos++, t++)
    {
       /* delete all single nodes from the heaviest list */
       initialize();
       stack_pos = 0;
       clust[t] = &group_pixel[numpixel]; /* point to start of new cluster */
       while ((root = find_alone_node(heaviest[root_pos])) == NULL)
          root_pos++;
       start_depth_first(group_pixel, stack, &stack_pos,
                         &numpixel, heaviest[root_pos]->node_tag, root);
       calculate_sum(heaviest[root_pos]->node_tag);
       calculate_sum(root->node_pos);
       nos_vertices += 2;
       new_ptr = &vertex[root->node_pos];
       old_ptr = heaviest[root_pos];
       do
       {  /* depth first next */
          temp_list = depth_first_segment(vertex, &stack_pos,
                                          &numpixel, group_pixel, stack);
          old_ptr = new_ptr;
          new_ptr = temp_list;
       } while(stack_pos != 1);  /* stack != NULL */
       for (k = 0; k < dimension; k++)
           mean[k] /= nos_vertices;
       output_image(clust[t], group_pixel);
    }
    clust[t] = &group_pixel[numpixel];  /* point to end */
}

output_single_node(group_pixel, clust, num_node)
int *group_pixel, **clust, num_node;
{
     register int i;

     for (i = 0; i < num_node; i++)
     {
         initialize();
         calculate_sum(group_pixel[i]);
         nos_vertices++;
         output_image(clust[i], group_pixel);
     }
}

start_depth_first(group_pixel, stack, stack_pos, numpix, root, node)
int *stack_pos, *numpix, *group_pixel, *stack, root;
TREE *node;
{
     stack[(*stack_pos)++] = root;        /* put root to stack */
     group_pixel[(*numpix)++] = root;     /* put root to cluster */
     stack[(*stack_pos)++] = node->node_pos;
     group_pixel[(*numpix)++] = node->node_pos;
     node->status = TMPVISIT;
}

LIST_HEAD * depth_first_segment(node_head, stack_pos, numpixel,
                                group_pixel, stack)
LIST_HEAD *node_head;
int *group_pixel, *stack;
int *stack_pos, *numpixel;
{
     TREE *root;

     if ((root = depth_first_next(node_head, stack_pos, numpixel,
                                  group_pixel, stack)) == NULL)
         return(NULL);
     calculate_sum(root->node_pos);
     nos_vertices++;

     return(&node_head[root->node_pos]);
}
```

```
visit_root(heaviest, root_pos)
LIST_HEAD **heaviest;
int root_pos;
{
    int i,j;
    TREE *this;

    for (i = 0,j = 1; i < 2; i++,j--)
    {  /* visit a pair of root */
       this = heaviest[root_pos+i]->node;
       while (this != NULL)
       {
          if (this->node_pos == heaviest[root_pos+j]->node_tag)
          {
              this->status = VISIT;
              break;
          }
          this = this->next;
       }
    }
}

int get_single_node(heaviest, num_root, all_node, clust, numpixel)
LIST_HEAD **heaviest;
int num_root, all_node[], *clust[], *numpixel;
{
    TREE *this_tree;
    int i, j, k, num_seg;
    char test;

    num_seg = 0;
    for (k = 0; k < num_root; k++)
    {   /* go over all root */
        i = 0;
        this_tree = heaviest[k]->node;
        while(this_tree != NULL)
        {  /* count number of node */
           if (this_tree->status == NOTVISIT)
              i++;  /* count any Notvisit node */
           this_tree = this_tree->next;
        }
        if (i == 0)
        {  /* node eligible to be single node */
           test = UNIQUE;
           if (num_seg > 0)
           {  /* more than 1 segment */
               for (j = 0; j < num_seg; j++)
               {
                   if (all_node[j] == heaviest[k]->node_tag)
                   {  /* check each cluster */
                      test = DUPLICATE;
                      break;
                   }
               }
           }
           if (test == UNIQUE)
           {  /* put unique single node to the cluster list */
              clust[num_seg++] = &all_node[*numpixel];
              all_node[(*numpixel)++] = heaviest[k]->node_tag;
           }
        }
   }
   return(num_seg);  /* return number of single node */
}

TREE * find_alone_node(list)
LIST_HEAD *list;
{
    TREE *this;

    this = list->node;
    while(this != NULL)
    {
       if (this->status == NOTVISIT)
          return(this);
       this = this->next;
    }
    return(NULL);
}

delete_duplicate()
{
    TREE *this;

    this = new_ptr->node;
    while (this != NULL)
    {  /* mark the duplicate node */
       if (this->node_pos == old_ptr->node_tag)
       {
          this->status = TMPVISIT;
          break;
```

```
        }
        this = this->next;
    }
}

TREE * depth_first_next(vertex, stack_pos, numpix, group_pixel, stack)
LIST_HEAD *vertex;
int *stack_pos, *numpix;
int *group_pixel, *stack;
{
    TREE *root;

    delete_duplicate();    /* delete duplicate node in the next tree list */
    if ((root = find_alone_node(new_ptr)) == NULL)
    {
       while (root == NULL)
       {
          /* delete stack */
          *stack_pos -= 2;
          new_ptr = &vertex[stack[*stack_pos]];
          root = find_alone_node(&vertex[stack[(*stack_pos)++]]);
          if(*stack_pos == 1 && root == NULL)    return(NULL);
       }
    }
    /* put node to stack */
    stack[(*stack_pos)++] = root->node_pos;
    /* put node to notepad for reset */
    group_pixel[(*numpix)++] = root->node_pos;
    root->status = TMPVISIT;
    return(root);
}

output_image(clust_start, group_pixel)
int *clust_start, group_pixel[];
{
     /* write to screen with precalculated average value */
     int i, k, a, b, index;

     index = clust_start-group_pixel;  /* get the index of group pixel */
     for(i = 0; i < nos_vertices; i++ ,index++) /* number of pixel */
     {
        get_coordinates(group_pixel[index], &a, &b);
        for (k = 0; k < dimension; k++) seg[k][a][b] = mean[k];
     }
}

/*************************************************************************
 *  You are reading bhc3.c                                               *
 *  This program read a segments file generated by tdseg3.c, and         *
 *  cluster the segments.                                                *
 *  Segment clustering based on graph theoretic approach.                *
 *  To obtain m segments the m-1 links will be deleted.                  *
 *  Use Hotelling test as distance between two segments.                 *
 *  Bottom up clustering.                                                *
 *  K.S.LAU 24-1-92                                                      *
 *************************************************************************/
#include "cluster.h"

typedef struct {
float sum[DIMENSION];
float ssq[DIMENSION];
float cosum[DIMENSION*(DIMENSION-1)/2];
int   num_vec;
}NSTAT;/* statistic required for a gaussian model */

float  hotelling_distance();
float  mahalanobis_distance();
float  get_hotelling_distance();
NSTAT  add_NSTAT();
NSTAT  segment_all_stat();

/*************************** GLOBAL VARIABLES ********************/
u_char  image[DIMENSION][MAX_YSIZE][MAX_XSIZE];
char    label[MAX_YSIZE][MAX_XSIZE];
int *group_pixel,
    *stack,
    *class_segment,/* store segments */
    num_seg,/* number of segments */
    dimension,
    top_x, top_y,
    win_xsize, win_ysize;
LIST head;
LIST_HEAD *new, *old;
struct rasterfile header;

main()
{
    int num_class,/* number of clusters */
        numread,
        **seg, **class_ptr,
        r, s, t;
```

```
LIST_HEAD *vertex, **root;
STAT  stat[MAX_CLUS];
FILE *seg_file;

    puts("SEGMENT CLUSTERING USING RST(INTRSET DISTANCE)");
    numread = 0;
    get_parameters(&seg_file, &numread, &num_class, &num_seg);
    get_data();
    stack = (int *)get_memory(num_seg, sizeof(int));
    group_pixel = (int *)get_memory(win_xsize*win_ysize+1, sizeof(int));
    vertex = (LIST_HEAD *)get_memory(num_seg, sizeof(LIST_HEAD));
    seg = (int **)get_memory(num_seg+1, sizeof(int *));
    /* point to end */
    seg[num_seg] = &group_pixel[win_xsize*win_ysize];
    /* allocate array of structure for linked list of tree */
    load_data(&seg_file, &numread, seg);

    puts("Grouping segments...");
    /* this is the essentially the same as the CEST algorithm */
    rst_clustering(vertex, seg, num_class);

    /* allocate array of pointer to the heaviest roots */
    class_segment = (int *)get_memory(num_seg+1, sizeof(int));
    class_ptr = (int **)get_memory(num_class+1, sizeof(int *));
    root = (LIST_HEAD **)get_memory(num_class, sizeof(LIST_HEAD *));
    get_root(vertex, root, num_class);
    group_segment(vertex, root, class_ptr, num_class);
    output_clusters(num_class, seg, class_ptr, stat);
    show_cluster_parameters(num_class, stat);
    image_file(num_class, stat);

}

get_root(vertex, root, num_class)
LIST_HEAD *vertex, **root;
int  num_class;
{
    int  i, j,
         comptag();
    LIST_HEAD  **ptr;

    ptr = (LIST_HEAD **)get_memory(num_seg, sizeof(LIST_HEAD *));

    for (i = 0; i < num_seg; i++) ptr[i] = &vertex[i];
    qsort((char *)ptr, num_seg, sizeof(LIST_HEAD *), comptag);

    /* get roots */
    for (i = j = 0; i < num_seg; i++, j++)
        if (ptr[i]->node_tag == 0) root[j] = ptr[i];
        else                       break;

    root[j++] = ptr[i++];
    for ( ; i < num_seg; i++)
        if (ptr[i]->node_tag != ptr[i-1]->node_tag)
            root[j++] = ptr[i];

    demand(num_class == j, find inconsistent number of segments);
    free((char *)ptr);

    /* initalize label for depth first search */
    for(i = 0; i < num_seg; i++)    vertex[i].node_tag = i;

}

int comptag(a, b)
LIST_HEAD **a, **b;
{    /* in ascending order of magnitude */
     return((*a)->node_tag - (*b)->node_tag);
}

group_segment(vertex, del_link, class_ptr, num_class)
LIST_HEAD *vertex, **del_link;
int **class_ptr, num_class;
{
    int i,stack_pos,num_seg,root_pos;
    TREE *root;
    LIST_HEAD *temp_list;

    num_seg = 0;
    i = find_single_segment(del_link, num_class, &num_seg, class_ptr);
    for (root_pos = 0; i < num_class; root_pos++, i++)
    {
        stack_pos = 0;
        while((root = find_alone_node(del_link[root_pos])) == NULL)
              root_pos++;
        class_ptr[i] = &class_segment[num_seg];
        start_depth_first(&stack_pos, &num_seg,
                          del_link[root_pos]->node_tag, root);
        new = &vertex[root->node_pos];
        old = &vertex[del_link[root_pos]->node_tag];
        do
```

```
        {
            temp_list = depth_first_class(vertex, &stack_pos, &num_seg);
            old = new;
            new = temp_list;
        } while(stack_pos != 1);
    }
    class_ptr[i] = &class_segment[num_seg];/* point to end */
}

make_link(total, vertex, link, region_sum, seg_stat)
int total;
LIST_HEAD *vertex;
ESTAT *region_sum, *seg_stat;
LIST *link;
{
     int i, j, k;

     /* fix the head */
     head.next = &link[0];
     head.last = &link[total-1];
     link[0].last  = link[total -1].next = &head;
     for (i = 0, j = total-1; i < total-1; i++, j--)
     {   /* connect the link */
         link[i].next = &link[i +1];
         link[j].last = &link[j -1];
     }
     for (i = k = 0; i < num_seg-1; i++)
     {
         for (j = i+1; j < num_seg; j++)
         {   /* calculate link weight */
            link[k].node.node1 = i;
            link[k].node.node2 = j;
            link[k++].weight =
                 get_hotelling_distance(i, j, vertex, region_sum,
                                        0, seg_stat);
         }
     }
}

float get_hotelling_distance(node1, node2, vertex,
                             region_sum, region, seg_stat)
int node1, node2, region;
LIST_HEAD *vertex;
ESTAT *region_sum, *seg_stat;
{
     if (vertex[node1].node_tag == 0 && vertex[node2].node_tag == 0)
      return(single_single(node1, node2, seg_stat));
     else if (vertex[node1].node_tag == 0 && vertex[node2].node_tag != 0)
            return(single_group(node1, region_sum, region, seg_stat));
     else if (vertex[node1].node_tag != 0 && vertex[node2].node_tag == 0)
          return(single_group(node2, region_sum, region, seg_stat));
     else    return(group_group(region_sum, vertex[node1].node_tag,
                                vertex[node2].node_tag));
}

float single_single(node1, node2, seg_stat)
int node1, node2;
ESTAT *seg_stat;
{
     float dist;

     dist = hotelling_distance(seg_stat[node1], seg_stat[node2]);

     return(dist);
}

float single_group(node, region_sum, region, seg_stat)
int node, region;
ESTAT *region_sum, *seg_stat;
{
     float  dist;

     dist = hotelling_distance(seg_stat[node], region_sum[region]);

     return(dist);
}

float group_group(region_sum, region1, region2)
int region1, region2;
ESTAT *region_sum;
{
     float  dist;

     dist = hotelling_distance(region_sum[region1], region_sum[region2]);

     return(dist);
}

float  hotelling_distance(a, b)
ESTAT a, b;
{
```

```
    int  i, j, k;
    float  prod, add, dist,
           mean1[DIMENSION],
           mean2[DIMENSION],
           covar1[DIMENSION][DIMENSION],
           covar2[DIMENSION][DIMENSION],
           addcovar[DIMENSION][DIMENSION];

    add = (float)(a.num_vec)+(float)(b.num_vec);
    prod = (float)(a.num_vec)*(float)(b.num_vec);
    for (k = 0; k < dimension; k++)
    {
        mean1[k] = a.sum[k]/(float)(a.num_vec);
        mean2[k] = b.sum[k]/(float)(b.num_vec);
    }
    if (add < DIMENSION)
    {/* if cannot compute covariance matrix return Euclidean distance */
         dist = 0.0;
         for (k = 0; k < dimension; k++)
             dist += SQUARE(mean1[k]-mean2[k]);
         return(dist);
    }
    else if (a.num_vec < DIMENSION || b.num_vec < DIMENSION)
    {/* if any one group is too small */
        if (a.num_vec < DIMENSION)
        {
            get_covar(b, covar2);
            inverse_matrix(covar2);
            dist = mahalanobis_distance(mean1, mean2, covar2);
        }
        else
        {
            get_covar(a, covar1);
            inverse_matrix(covar1);
            dist = mahalanobis_distance(mean1, mean2, covar1);
        }
        dist = dist*prod/add;
    }
    else
    {/* both covariance matrix can be computed */
        get_within_group_ssq(a, covar1);
        get_within_group_ssq(b, covar2);
        for (i = 0; i < dimension; i++)
            for (j = 0; j < dimension; j++)
                addcovar[i][j] += covar1[i][j]+covar2[i][j];
        inverse_matrix(addcovar);
        for (i = 0; i < dimension; i++)
            for (j = 0; j < dimension; j++)
                addcovar[i][j] *= add-2.0;
        dist = mahalanobis_distance(mean1, mean2, addcovar);
        dist = dist*prod/add;
    }

    return(dist);
}

get_covar(c, covar)
NSTAT  c;
float  covar[][DIMENSION];
{
    register int  i, j, k, total;

    total = c.num_vec;

    for (k = 0; k < dimension; k++)
        covar[k][k] = ((float)total*c.ssq[k]-SQUARE(c.sum[k]))
                      /((float)total*(float)(total-1));/* unbaised */
    for (i = k = 0; i < dimension; i++)
        for (j = i+1; j < dimension; j++, k++)
        {
            covar[i][j] = ((float)total*c.cosum[k]-(c.sum[i]*c.sum[j]))
                          /((float)total*(float)(total-1));
            covar[j][i] = covar[i][j];
        }
}

get_within_group_ssq(c, covar)
NSTAT  c;
float  covar[][DIMENSION];
{
    register int  i, j, k, total;

    total = c.num_vec;

    for (k = 0; k < dimension; k++)
        covar[k][k] = ((float)total*c.ssq[k]-SQUARE(c.sum[k]))
                      /(float)total;
    for (i = k = 0; i < dimension; i++)
        for (j = i+1; j < dimension; j++, k++)
        {
            covar[i][j] = ((float)total*c.cosum[k]-(c.sum[i]*c.sum[j]))
```

```
                        /(float)total;
            covar[j][i] = covar[i][j];
        }
}

inverse_matrix(wcovar)
float  wcovar[DIMENSION][DIMENSION];
{
     register int  i, j, k;

     /* compute elements of reduced matrix */
     for (k = 0; k < dimension; k++)
     {   /* new elements of pivot row */
         for (j = 0;  j < dimension; j++)
             if (j != k) wcovar[k][j] /= wcovar[k][k];
         /* element replacing pivot element */
         wcovar[k][k] = 1.0/wcovar[k][k];
         /* compute new elements not in pivot row or pivot column */
         for (i = 0; i < dimension; i++)
             if (i != k)
                 for (j = 0; j < dimension; j++)
                     if (j != k)
                         wcovar[i][j] = wcovar[i][j]
                                        -wcovar[k][j]*wcovar[i][k];
         /* compute replacement elements for
                pivot column-except pivot element */
         for (i = 0; i < dimension; i++)
             if (i != k)
                 wcovar[i][k] *= -wcovar[k][k];
     }
}

float  mahalanobis_distance(mean1, mean2, wcovar)
float  *mean1, *mean2,
       wcovar[][DIMENSION];
{/* generalised Mahalanobis distance */
register int  k, a, b;
float  vector[DIMENSION],
       result[DIMENSION],
       distance = 0.0;

for (k = 0; k < dimension; k++) result[k] = 0.0;

for (k = 0; k < dimension; k++)
    vector[k] = mean1[k]-mean2[k];

for (a = 0; a < dimension; a++)
    for (b = 0; b < dimension; b++)
        result[a] += vector[b]*wcovar[b][a];

for (k = 0; k < dimension; k++)
    distance += result[k]*vector[k];

return(distance);
}

calculate_cosum_square(node, sum)
int node;
NSTAT *sum;
{
     int i, j, k, a, b;

     get_coordinates(node, &i, &j);
     for (k = 0; k < dimension; k++)
     {
         sum->sum[k] += (float)image[k][i][j];
         sum->ssq[k] += SQUARE((float)image[k][i][j]);
     }
     for (a = k = 0; a < dimension; a++)
         for (b = a+1; b < dimension; b++, k++)
             sum->cosum[k] += (float)image[a][i][j]*(float)image[b][i][j];
}

NSTAT add_NSTAT(a, b)
NSTAT a, b;
{
     int  k;

     for (k = 0; k < dimension; k++)
     {
         a.sum[k] += b.sum[k];
         a.ssq[k] += b.ssq[k];
     }
     for (k = 0; k < dimension*(dimension-1)/2; k++)
         a.cosum[k] += b.cosum[k];
     a.num_vec += b.num_vec;

     return(a);
}

NSTAT segment_all_stat(node, seg)
```

```
int node, **seg;
{
     register int index, i;
     NSTAT sum;

     initialize_NSTAT(&sum);
     index = seg[node] - group_pixel;
     sum.num_vec = seg[node+1] - seg[node];
     for (i = 0; i < sum.num_vec; i++, index++)
         calculate_cosum_square(group_pixel[index], &sum);
     return(sum);  /* intraset distance */
}

initialize_NSTAT(a)
NSTAT *a;
{
     int  k;

     a->num_vec = 0;
     for (k = 0; k < dimension; k++)
         a->sum[k] = a->ssq[k] = 0.0;
     for (k = 0; k < dimension*(dimension-1)/2; k++)
         a->cosum[k] = 0.0;
}

rst_clustering(vertex, seg, num_class)
LIST_HEAD *vertex;
int **seg, num_class;
{
    int region, total, region1, region2, node1, node2;
    LIST *link, *min;
    NSTAT *region_sum, *seg_stat;
    TREE *temp1_tree,*temp2_tree;

    total = number_list();
    link = (LIST *)get_memory(total, sizeof(LIST));
    region_sum = (NSTAT *)get_memory(num_seg, sizeof(NSTAT));
    seg_stat = (NSTAT *)get_memory(num_seg, sizeof(NSTAT));
    for (region = 0; region < num_seg; region++)
        seg_stat[region] = segment_all_stat(region, seg);
    make_link(total, vertex, link, region_sum, seg_stat);
    /* take down the region of each segment */
    for(region = 1; region < num_seg; region++)
    {
       /* pick the smallest link */
       /* record to unique link for sorting later */
       min = lightest_link();
       region1 = vertex[min->node.node1].node_tag;
       region2 = vertex[min->node.node2].node_tag;
       node1 = min->node.node1;
       node2 = min->node.node2;
       /* save the smallest link */
       temp1_tree = get_node();    /* put into linked list */
       temp2_tree = get_node();
       temp1_tree->node_pos = node2;
       temp2_tree->node_pos = node1;
       /* put link into the tree */
       vertex[node1].node = front_of_tree(temp1_tree, vertex[node1].node);
       vertex[node2].node = front_of_tree(temp2_tree, vertex[node2].node);
       if (region < num_seg-1)
       {
           sum_label_vertex(seg_stat, region_sum, vertex, min,
                            region, region1, region2);
           delete_recal(seg_stat, min, link, vertex, region_sum,
                            region, region1, region2);
       }
       if (num_seg-num_class == region) break;
    }
    free((char *)region_sum);
    free((char *)seg_stat);
    free((char *)link);
}

LIST *lightest_link()
{
     LIST *min, *this;

     min = this = head.next;
     while (this != &head)
     {
         if (min->weight > this->weight)   min = this;
         this = this->next;
     }
     min->last->next = min->next;
     min->next->last = min->last;
     min->next = NULL;
     return(min);
}

sum_label_vertex(seg_stat, region_sum, vertex, min, region, region1, region2)
int region, region1, region2;
```

```
LIST *min;
LIST_HEAD *vertex;
NSTAT *region_sum, *seg_stat;
{
     int i;

     if (region1 == 0 && region2 == 0)
     {
        vertex[min->node.node1].node_tag = region;
        vertex[min->node.node2].node_tag = region;
        region_sum[region] = add_NSTAT(seg_stat[min->node.node1],
                                       seg_stat[min->node.node2]);
     }
     else if (region1 == 0 && region2 != 0)
     {
         vertex[min->node.node1].node_tag = region;
         for (i = 0; i < num_seg; i++)
         {
             if (vertex[i].node_tag == region2)
                 vertex[i].node_tag = region;
         }
         region_sum[region] = add_NSTAT(seg_stat[min->node.node1],
                                        region_sum[region2]);
     }
     else if (region1 != 0 && region2 == 0)
     {
         vertex[min->node.node2].node_tag = region;
         for (i = 0; i < num_seg; i++)
         {
             if (vertex[i].node_tag == region1)
                vertex[i].node_tag = region;
         }
         region_sum[region] = add_NSTAT(seg_stat[min->node.node2],
                                        region_sum[region1]);
     }
     else
     {
         for (i = 0; i < num_seg; i++)
         {
             if (vertex[i].node_tag == region1 ||
                 vertex[i].node_tag == region2)
                 vertex[i].node_tag = region;
         }
         region_sum[region] = add_NSTAT(region_sum[region1],
                                        region_sum[region2]);
     }
}

int jump(node1)
int node1;
{       /* return the start position */
return((num_seg*node1)-(node1*(node1+1)/2));
}

delete_recal(seg_stat, min, link, vertex, region_sum,
             region, region1, region2)
LIST *link, *min;
LIST_HEAD *vertex;
NSTAT *region_sum, *seg_stat;
int region, region1, region2;
{
    register int i, j, k;
    LIST *this, *that;
    int min1, min2, region_node, new_node, top;

    min1 = min->node.node1;
    min2 = min->node.node2;
    top = jump(min1+1);
    if (region1 == 0 && region2 == 0)
    {   /* del_recal single to single */
        this = head.next;
        while (this != &head)
        {
            if (this - link > top)   break;
            if (this->node.node1 == min1
             || this->node.node2 == min1)
            {
                if (this->node.node1 == min1)
                    new_node = this->node.node2;
                else
                    new_node = this->node.node1;
                /* recalculate first */
                this->weight =
                      get_hotelling_distance(this->node.node1,
                                             this->node.node2, vertex,
                                             region_sum, region, seg_stat);
                if (new_node == this->node.node1 ||
                    new_node == this->node.node2)
                    j = this -link +1;
         else if (new_node > min2)   j = jump(min2);
                else                 j = jump(new_node);
```

```
                that = &link[j];
                while (that->next == NULL)   that = &link[++j];
                while (that != &head)
                {   /* delete */
                    if ((that->node.node1 == new_node &&
                        that->node.node2 == min2) ||
                        (that->node.node2 == new_node &&
                        that->node.node1 == min2))
                    {   /* del one link for every node */
                        that->last->next = that->next;
                        that->next->last = that->last;
                        that->next = NULL;
                        break;
                    }
                    that = that->next;
                }
            }
            this = this->next;
    }
}
else
{   /* del_recal for single to group and group to group */
    for (i = 0; i < num_seg; i++)
    {   /* for all node in region */
            if (vertex[i].node_tag == region)
            {
                region_node = i;
                this = head.next;
                while (this != &head)
                {   /* for every other node to this region node */
                    if (this - link > top)   break;
                    if (this->node.node1 == region_node ||
                        this->node.node2 == region_node)
                    {
                        if (this->node.node1 == region_node)
                           new_node = this->node.node2;
                        else
                           new_node = this->node.node1;
                        this->weight =
                             get_hotelling_distance(this->node.node1,
                                                    this->node.node2, vertex,
                                                    region_sum, region, seg_stat);
                        if (new_node == this->node.node1 ||
                            new_node == this->node.node2)
                            k = this -link +1;
                        else if (new_node > min2)   k = jump(min2);
                        else                        k = jump(new_node);
                        that = &link[k];
                        while (that->next == NULL)   that = &link[++k];
                        while (that != &head)
                        {
                             if ((that->node.node1 == new_node &&
                                 vertex[that->node.node2].node_tag == region)
                                 || (that->node.node2 == new_node &&
                                 vertex[that->node.node1].node_tag == region))
                             {
                                 that->last->next = that->next;
                                 that->next->last = that->last;
                                 that->next = NULL;
                                 break;
                             }
                             that = that->next;
                        }
                    }
                    this = this->next;
                }
            }
        }
    }
}

/*************************************************************************
 *   You are reading bmirs1.c                                            *
 *   Bottom up segmentation using CEST.                                  *
 *   This is basically the same as rsst3.c                               *
 *   Compute information loss to guide segmentation.                     *
 *   Use a gaussian model for each segment.                              *
 *   K.S.LAU 24-1-92                                                     *
 *************************************************************************/

#include "bu.h"

/* when to start computing entropy */
#define  START_ENTROPY  0.2/* percentage of number of pixel i.e. segments */
#define  ENTROPY_STEP   30/* compute entropy after ENTROPY_STEP of segments */
#define  TRIVIAL_SEG    5/* no entropy for segment smaller than this */
/************************** FUNCTION DEFITIONS ************************/
char *      get_memory();
float       link_weight();
float       get_entropy();
float       get_whole_entropy();
```

```
float       matrix_determinant();
float       new_weight();
LINK *      lightest();
LINK *      jump();
LINK *      compress();
TREE *      front_of_tree();
TREE *      find_alone_node();
TREE *      depth_first_next();
STAT        add_struct();
STAT        get_pix_value();
LIST_HEAD * depth_first_seg();

/***************************** GLOBAL VARIABLES ************************/
extern  LIST_HEAD  *new_ptr, *old_ptr;
u_char  image[DIMENSION][MAX_YSIZE][MAX_XSIZE];/* store image */
u_char  seg[DIMENSION][MAX_YSIZE][MAX_XSIZE];/* store segmented image */
int  num_seg,/* user required number of segments */
     entropy_step,/* how often to compute entropy loss */
     start_entropy,/* when to start compute entropy loss */
     dimension,/* dimension of the image */
     top_x, top_y,/* top left coordinates of the process window */
     win_xsize, win_ysize;/* size of process window */
struct rasterfile header;/* header of the image file */
LINK *edges, head;  /* edges for array, head for list */

main()
{
     char answer[5];
     int s,
         total_link,
         **clus, *group_pixel,/* to store a region */
         *stack,/* for depth first search */
         cpu_time;
     char  *rtag;/* root label */
     LIST_HEAD *vertex;

     puts("BOTTOM UP SEGMENATION USING RST");
     get_data();/* get input data */
     start_entropy = (int)(win_xsize*win_ysize*START_ENTROPY);
     entropy_step = ENTROPY_STEP;
     printf("Start computing entropy when number of segments are %d\n",
            start_entropy);
     printf("The sampling frequency of entropy is every %d segments\n",
            entropy_step);
     printf("Do you want to change the above parameters? y/n ");
     scanf("%s", answer);
     if (strchr(answer, 'y') || strchr(answer, 'Y'))
     {
         printf("Enter the required starting point, ");
         scanf("%d", &start_entropy);
         printf("Enter the sampling frequency, ");
         scanf("%d", &entropy_step);
     }
     printf("Enter the number of segments to be generated, ");
     scanf("%d", &num_seg);
     total_link = number_link();
     /* allocate array of structure for linked list of tree */
     vertex = (LIST_HEAD *)get_memory(win_xsize*win_ysize, sizeof(LIST_HEAD));
     edges = (LINK *)get_memory(4*win_xsize*win_ysize-4, sizeof(LINK));
     rtag = (char *)get_memory(win_xsize*win_ysize, sizeof(char));
     for (s = 0; s < win_xsize*win_ysize; s++) rtag[s] = ROOT;
     stack = (int *)get_memory(win_xsize*win_ysize, sizeof(int));
     group_pixel = (int *)get_memory(win_xsize*win_ysize, sizeof(int));
     if (num_seg > start_entropy)
         clus = (int **) get_memory(num_seg+1, sizeof(int *));
     else
         clus = (int **) get_memory(start_entropy+1, sizeof(int *));

     clock();
     /* compute all weight of links */
     printf("Calculating link weight.\n");
     get_link();

     printf("Merging...\n");
     recru_tree(vertex, rtag, group_pixel, clus, stack);
     /* free the edges memory */
     free((char *)edges);

     cpu_time = clock();
     printf("Run time was %.2f sec.\n", cpu_time / 1.0e6);

}
/***************************** END OF MAIN ****************************/

get_data()
{
char  filename[DIMENSION][50];
int  i, j;
FILE  *f[DIMENSION];
struct  rasterfile  head[DIMENSION];
```

```
printf("Enter the number of channel to be used < 4, ");
scanf("%d", &dimension);

for (i = 0; i < dimension; i++)
{
    printf("Enter the channel filename %d, ", i);
    scanf("%s", (char *)&filename[i][0]);
    f[i] = fopen((char *)&filename[i][0], "r");
    demand(f[i], Cannot open file);
}

        /* store header in global area, for later use */
        fread((char *)&header, sizeof(struct rasterfile), 1, f[0]);
        rewind(f[0]);
        for (i = 0; i < dimension; i++)
        {
            fread((char *)&head[i], sizeof(struct rasterfile), 1, f[i]);
    demand(header.ras_length == head[i].ras_length,
          Make sure the images has the same size and coordinates);
}

        printf("The images size is %d\n", header.ras_width);
        printf("Enter the window width (xsize), ");
        scanf("%d", &win_xsize);
        printf("Enter the window height (ysize), ");
        scanf("%d", &win_ysize);

        printf("Enter the top left window coordinate
                of the image to be processed,\n");
        printf("\tx: "); scanf("%d", &top_x);
        printf("\ty: "); scanf("%d", &top_y);


demand(head[0].ras_width >= win_xsize+top_x,
       The window is out of range\, please reduce size);

for (i = 0; i < dimension; i++)
            fseek(f[i], (long)(top_y*head[i].ras_width+top_x), 1);
for (i = 0; i < dimension; i++)
    for (j = 0; j < win_ysize; j++)
    {
        fread((char *)&image[i][j][0], sizeof(char), win_xsize, f[i]);
        fseek(f[i], (long)(head[i].ras_width-win_xsize), 1);
    }

for (i = 0; i < dimension; i++) fclose(f[i]);
}

char *get_memory(items, size)
int items, size;
{
     char *buffer;

     buffer = (char *)calloc((unsigned int)items, (unsigned int)size);
     demand(buffer, no memory);

     return(buffer);
}

int number_link()
{
int s, total_link;

total_link = 0;
for (s = 1; s < win_xsize-1; s++)
    total_link += s;
total_link *= 4;
total_link += 2*(win_xsize*win_ysize-1);
        printf("\nTotal number of links in original graph is %d.\n",
               total_link);

return (total_link);
}

get_link()
{
register int i, node;

        head.next = edges;
        edges[0].last = &head;
for (i = node = 0; node < win_xsize*win_ysize-1; node++)
{   /* for every vertex, except the last one */
    if (node >= win_xsize*win_ysize-win_xsize)  /* last row */
    {
       if (node == win_xsize*win_ysize-2)
       {  /* last vertice */
  edges[i].node1 = node;
  edges[i].node2 = node+1;
  edges[i].weight = link_weight(node, node+1);
                edges[i].next = &head;
  head.last = &edges[i];
```

```
        }
        else   fill_edges(node, node+1, i, 4);
                edges[++i].weight = -1.0;  /* fill the gap */
                edges[++i].weight = -1.0;
        edges[++i].weight = -1.0;
        ++i;
      }
      else
            {
        if ((node % win_xsize) == 0)  /* row head */
              {
         fill_edges(node, node+1, i, 1);
    fill_edges(node, node+win_xsize+1, ++i, 1);
    fill_edges(node, node+win_xsize, ++i, 2);
    edges[++i].weight = -1.0;
    ++i;
          }
        else if ((node % win_xsize) == win_xsize-1)  /* row tail */
            {
    fill_edges(node, node+win_xsize, i, 1);
    fill_edges(node, node+win_xsize-1, ++i, 3);
    edges[++i].weight = -1.0;
    edges[++i].weight = -1.0;
    ++i;
        }
        else
        {
    fill_edges(node, node+1, i, 1);
    fill_edges(node, node+win_xsize+1, ++i, 1);
    fill_edges(node, node+win_xsize, ++i, 1);
    fill_edges(node, node+win_xsize-1, ++i, 1);
    ++i;
        }
            }
}
}

fill_edges(node1, node2, index, offset)
register int node1, node2, index, offset;
{
     edges[index].node1 = node1;
     edges[index].node2 = node2;
     edges[index].weight = link_weight(node1, node2);
     edges[index].next = &edges[index + offset];
     edges[index + offset].last = &edges[index];
}

float link_weight(node1, node2)
int node1, node2;
{
     int i1, j1, i2, j2,
         k;
     float  weight;

     get_coordinates(node1, &i1, &j1);
     get_coordinates(node2, &i2, &j2);
     weight = 0.0;
     for (k = 0; k < dimension; k++)
         weight += SQUARE((float)image[k][i1][j1]-(float)image[k][i2][j2]);

     return(weight);
}

TREE * front_of_tree(new, list)
TREE *new, *list;
{
     new->next = list;
     list = new;

     return(list);
}

/* construct CEST using Kruskals algorithm */
recru_tree(vertex, rtag, group_pixel, clus, stack)
LIST_HEAD *vertex;
char  *rtag;
int  *group_pixel, **clus, *stack;
{
    char  *file = "/home/image/output/bu.mi";
    int step, count, label, biggest, smallest;
    float  entropy;
    TREE *node;
    LIST_HEAD **root;
    STAT *region_sum;
    LINK *lightest_ptr;
    FILE  *fp;

    node = (TREE *)get_memory(2*(win_xsize*win_ysize-1), sizeof(TREE));
    region_sum = (STAT *)get_memory(win_xsize*win_ysize, sizeof(STAT));
    root = (LIST_HEAD **)get_memory(start_entropy, sizeof(LIST_HEAD *));
    fp = fopen(file, "w");
```

```
    demand(fp, Cannot open entropy file);
    fprintf(fp, "%d\n", ((start_entropy-num_seg)/entropy_step)+2);
    step = 0;
    for(count = 0, label = 1; label < win_xsize*win_ysize; label++)
    {
       /* pick the lightest link */
       lightest_ptr = lightest();
       /* put link into the tree */
       node[count].node_pos = lightest_ptr->node2;
       vertex[lightest_ptr->node1].node =
           front_of_tree(&node[count++],vertex[lightest_ptr->node1].node);
       node[count].node_pos = lightest_ptr->node1;
       vertex[lightest_ptr->node2].node =
           front_of_tree(&node[count++],vertex[lightest_ptr->node2].node);
       if (label < win_xsize*win_ysize-1)
       {
          sum_label_nodes(region_sum, vertex, lightest_ptr, label,
                          &biggest,&smallest, rtag, group_pixel,
                          clus, stack);
          /* delete all duplicate link and recalculate link weight */
          del_duplicate_recal(vertex, region_sum, label, biggest, smallest);
       }

       if (win_xsize*win_ysize-label <= start_entropy)
       {
           if (step % entropy_step == 0)
           {
               entropy = get_entropy(root, vertex, group_pixel,
                                     clus, stack, rtag, label);
               fprintf(fp, "%d  %f\n", win_xsize*win_ysize-label, entropy);
           }
           step ++;/* compute entropy every number of step */
       }
       if (win_xsize*win_ysize-num_seg == label) break;
    }
    free((char *)region_sum);
    free((char *)root);
    entropy = get_whole_entropy();
    fprintf(fp, "%d  %f\n", 1, entropy);
    fclose(fp);
    printf("The entropy output is stored in %s\n", file);
}

float get_whole_entropy()
{
     int  i, j, k, m,
          nodeinseg;
     float  entropy,
            sum[DIMENSION], ssq[DIMENSION],
            covar[DIMENSION][DIMENSION],
            cosum[DIMENSION][DIMENSION];

     nodeinseg = win_xsize*win_ysize;
     for (k = 0; k < dimension; k++)
         sum[k] = ssq[k] = 0.0;
     for (m = 0; m < dimension; m++)
         for (k = 0; k < dimension; k++)
             cosum[m][k] = 0.0;
     for (i = 0; i < win_ysize; i++)
         for (j = 0; j < win_xsize; j++)
         {
             for (k = 0; k < dimension; k++)
             {
                 sum[k] += (float)image[k][i][j];
                 ssq[k] += SQUARE((float)image[k][i][j]);
             }
             for (m = 0; m < dimension; m++)
                 for (k = m+1; k < dimension; k++)
                     cosum[m][k] += (float)image[k][i][j]
                                    *(float)image[m][i][j];
         }
     for (m = 0; m < dimension; m++)/* comput covariance matrix */
         for (k = m; k < dimension; k++)
             if (k != m)/* covariance */
             {
                 covar[m][k] = ((float)nodeinseg*cosum[m][k]
                 -sum[m]*sum[k])/((float)nodeinseg*(float)(nodeinseg-1));
                 covar[k][m] = covar[m][k];
             }
             else/* variance */
             {
                 covar[m][k] = ((float)nodeinseg*ssq[k]-SQUARE(sum[k]))
                 /((float)nodeinseg*(float)(nodeinseg-1));
             }
     entropy = 17.0795*matrix_determinant(covar);
     entropy = (float)dimension*log(entropy)/2.0;
     printf("Entropy loss for whole image is %f\n", entropy);

     return(entropy);
}
```

```
float get_entropy(root, vertex, group_pixel, clus, stack, rtag, label)
LIST_HEAD  *vertex, **root;
int  *group_pixel, **clus, *stack, label;
char  *rtag;
{
     int  i, j, k, m, a, b,
          num_node,
          num_segment,
          start,
          nodeinseg;
     float  sigma, entropy,
            sum[DIMENSION], ssq[DIMENSION],
            covar[DIMENSION][DIMENSION],
            cosum[DIMENSION][DIMENSION];

     num_node = win_xsize*win_ysize;
     num_segment = get_root(vertex, root, rtag);
     get_all_segment(root, vertex, group_pixel, clus, num_segment, stack);
     reset_whole_tree(vertex);

     sigma = 0.0;
     /* compute entropy */
     for (i = 0; i < num_segment; i++)
     {/* entropy for each segment */
         nodeinseg = clus[i+1]-clus[i];
         if (nodeinseg < TRIVIAL_SEG) continue;/* trivial segment */
         for (k = 0; k < dimension; k++)
             sum[k] = ssq[k] = 0.0;
         for (m = 0; m < dimension; m++)
             for (k = 0; k < dimension; k++)
                 cosum[m][k] = 0.0;
         start = clus[i]-clus[0];
         for (j = 0; j < nodeinseg; j++)
         {
             get_coordinates(group_pixel[start+j], &a, &b);
             for (k = 0; k < dimension; k++)
             {/* for variance */
                 sum[k] += (float)image[k][a][b];
                 ssq[k] += SQUARE((float)image[k][a][b]);
             }
             for (m = 0; m < dimension; m++)
                 for (k = m+1; k < dimension; k++)
                     cosum[m][k] += (float)image[k][a][b]
                                    *(float)image[m][a][b];
         }
         for (m = 0; m < dimension; m++)/* comput covariance matrix */
             for (k = m; k < dimension; k++)
                 if (k != m)/* covariance */
                 {
                     covar[m][k] = ((float)nodeinseg*cosum[m][k]
                     -sum[m]*sum[k])/((float)nodeinseg*(float)(nodeinseg-1));
                     covar[k][m] = covar[m][k];
                 }
                 else/* variance */
                 {
                     covar[m][k] = ((float)nodeinseg*ssq[k]-SQUARE(sum[k]))
                     /((float)nodeinseg*(float)(nodeinseg-1));
                 }
         entropy = matrix_determinant(covar);
         if (iszero(entropy)) continue;
         entropy = (float)dimension*log(17.0795*entropy)/2.0;
         entropy = entropy*(float)nodeinseg/(float)num_node;
         sigma += entropy;
     }

     return (sigma);/* sum entropy x segment proabilities */
}

float matrix_determinant(w)
float  w[DIMENSION][DIMENSION];
{
int  iflag,
       ipivot[DIMENSION],
     istar,
     i, j, k;
float  awikod,
       colmax,
       ratio,
       rowmax,
       temp,
       det,
       d[DIMENSION];

iflag = 1;
/* initialise ipivot, d */
for (i = 0; i < dimension; i++)
{
    ipivot[i] = i;
    rowmax = 0.0;
    for (j = 0; j < dimension; j++)
        if (rowmax < fabs(w[i][j])) rowmax = fabs(w[i][j]);
```

```
    if (rowmax == 0.0)
    {
        iflag = 0;
        rowmax = 1.0;
    }
    d[i] = rowmax;
}
/* factorisation */
for (k = 0; k < dimension-1; k++)
{
    /* determine pivot row, the row istar */
    colmax = fabs(w[k][k])/d[k];
    istar = k;
    for (i = k+1; i < dimension; i++)
    {
        awikod = fabs(w[i][k])/d[i];
        if (awikod > colmax)
        {
            colmax = awikod;
            istar = i;
        }
    }
    if (colmax == 0.0) iflag = 0;
    else
    {
        if (istar > k)
        {
            /* make k the pivot row by interchangeing
               it with the chosen row istar */
            iflag = -iflag;
            i = ipivot[istar];
            ipivot[istar] = ipivot[k];
            ipivot[k] = i;
            temp = d[istar];
            d[istar] = d[k];
            d[k] = temp;
            for (j = 0; j < dimension; j++)
            {
                temp = w[istar][j];
                w[istar][j] = w[k][j];
                w[k][j] = temp;
            }
        }
        /* eliminate  x[k] from rows k+1...n */
        for (i = k+1; i < dimension; i++)
        {
            w[i][k] /= w[k][k];
            ratio = w[i][k];
                    for (j = k+1; j < dimension; j++)
                        w[i][j] -= ratio*w[k][j];
        }
    }
}
if (w[dimension-1][dimension-1] == 0.0) iflag = 0;

det = (float)iflag;
for (i = 0; i < dimension; i++) det *= w[i][i];

return(det);
}

get_all_segment(rnode, vertex, group_pixel, clust, segment, stack)
LIST_HEAD *vertex, **rnode;
int *group_pixel, **clust, *stack;
int segment;
{/* put all segments into group_pixel */
    LIST_HEAD *temp_list;
    TREE *root;
    int t, stack_pos, root_pos, numpixel;

    numpixel = 0;   /* all root already visited */
    t = get_single_node(vertex, rnode, segment, group_pixel,
                        clust, &numpixel);
    for(root_pos = 0; t < segment; root_pos++, t++)
    {
       /* delete all single nodes from the rnode list */
       initialize();
       stack_pos = 0;
       /* point to start of new cluster */
       clust[t] = &group_pixel[numpixel];
       while ((root = find_alone_node(rnode[root_pos])) == NULL)
              root_pos++;
       start_depth_first(group_pixel, stack, &stack_pos,
                         &numpixel, rnode[root_pos]->vertex, root);
       new_ptr = &vertex[root->node_pos];
       old_ptr = rnode[root_pos];
       do
       {  /* depth first next */
             temp_list = depth_first_seg(vertex, &stack_pos,
                                         &numpixel, group_pixel, stack);
             old_ptr = new_ptr;
```

```
            new_ptr = temp_list;
        }while(stack_pos != 1);  /* stack != NULL */
    }
    clust[t] = &group_pixel[numpixel];  /* point to end */
}

LIST_HEAD * depth_first_seg(vertex, stack_pos, numpixel,
                            group_pixel, stack)
LIST_HEAD *vertex;
int *group_pixel, *stack;
int *stack_pos, *numpixel;
{
    TREE *root;

    if ((root = depth_first_next(vertex, stack_pos, numpixel,
                                 group_pixel, stack)) == NULL)
        return(NULL);

    return(&vertex[root->node_pos]);
}

reset_whole_tree(vertex)
LIST_HEAD *vertex;
{
register int i;
TREE *this_node;

for (i = 0; i < win_xsize*win_ysize; i++)
{
            this_node = vertex[i].node;
            while (this_node != NULL)
            {
                  if (this_node->status == TMPVISIT)
                     this_node->status = NOTVISIT;
                  this_node = this_node->next;
            }
}
}

LINK * lightest()
{   /* find the lightest link */
    register LINK *min_ptr, *this;

    min_ptr = head.next;
    this = min_ptr->next;
    while (this != &head)
    {
        if (min_ptr->weight > this->weight)
            min_ptr = this;
        this = this->next;
    }
    /* delete lightest link */
    min_ptr->last->next = min_ptr->next;
    min_ptr->next->last = min_ptr->last;
    min_ptr->next = NULL;  /* status out */

    return(min_ptr);
}

sum_label_nodes(region_sum, vertex, light, tag, biggest, smallest, rtag,
                group_pixel, clus, stack)
STAT *region_sum;
LINK *light;
int tag;
register LIST_HEAD *vertex;
register int *biggest, *smallest;
char *rtag;
int  *group_pixel, **clus, *stack;
{
    register int i, done = 0;
    int old_tag[2];

    if (vertex[light->node1].node_tag == 0 &&
        vertex[light->node2].node_tag == 0)
    {     /* both are new node, use a new label */
        vertex[light->node1].node_tag = tag;
        vertex[light->node2].node_tag = tag;
        region_sum[tag] = add_struct(get_pix_value(light->node1),
                                     get_pix_value(light->node2));
        rtag[light->node2] = NROOT;/* remove from root list */
        *smallest = light->node1;
        *biggest = light->node2;
    }
    else if (vertex[light->node1].node_tag == 0 &&
             vertex[light->node2].node_tag != 0)
    {
          region_sum[tag] = add_struct(get_pix_value(light->node1),
                                       region_sum[vertex[light->node2].node_tag]);
          vertex[light->node1].node_tag = tag;
          old_tag[0] = vertex[light->node2].node_tag;
          rtag[light->node1] = NROOT;
```

```
            for (i = 0; i < win_xsize*win_ysize; i++)
            {     /* update label and count */
              if (vertex[i].node_tag == old_tag[0])
              {
                  vertex[i].node_tag = tag;
                  if (!done)
                  {
                     *smallest = i;
                     ++done;
                  }
                  *biggest = i;
              }
            }
            if (light->node1 < *smallest)   *smallest = light->node1;
      }
      else if (vertex[light->node1].node_tag != 0 &&
               vertex[light->node2].node_tag == 0)
      {
            region_sum[tag] = add_struct(get_pix_value(light->node2),
 region_sum[vertex[light->node1].node_tag]);
            vertex[light->node2].node_tag = tag;
            old_tag[0] = vertex[light->node1].node_tag;
            rtag[light->node2] = ROOT;
            for (i = 0; i < win_xsize*win_ysize; i++)
            {     /* update label and count */
                 if (vertex[i].node_tag == old_tag[0])
                 {
                     vertex[i].node_tag = tag;
                     if (!done)
                     {
                         *smallest = i;
                         ++done;
                     }
                     *biggest = i;
                 }
            }
            if (light->node2 > *biggest)   *biggest = light->node2;
      }
      else
      {  /* both node tag != 0 */
         old_tag[0] = vertex[light->node1].node_tag;
         old_tag[1] = vertex[light->node2].node_tag;
         region_sum[tag] =
                 add_struct(region_sum[vertex[light->node1].node_tag],
         region_sum[vertex[light->node2].node_tag]);
         remove_root(light->node2, vertex, group_pixel, clus, stack, rtag);
         reset_tree_tmp(vertex, group_pixel, clus[1]-clus[0]);
         for (i = 0; i < win_xsize*win_ysize; i++)
         {
             if (vertex[i].node_tag == old_tag[0] ||
                 vertex[i].node_tag == old_tag[1])
             {
                 vertex[i].node_tag = tag;
                 if (!done)
                 {
                     *smallest = i;
                     ++done;
                 }
                 *biggest = i;
             }
         }
      }
}

del_duplicate_recal(vertex, region_sum, region, biggest, smallest)
register LIST_HEAD *vertex;
STAT *region_sum;
int region, biggest, smallest;
{
      /* delete all duplicate link connected to external vertices */
      register LINK *find,*this,*that;
      LINK dummy;
      int temp_node, temp_re;

      /* goto starting point */
      this = jump(smallest);
      /* connect the related link in a shorter list */
      this = compress(this, vertex, region, biggest);
      while (this != NULL)
      {  /* first recalculate then delete */
         temp_re = 0; /* initialize for delete */
         if (vertex[this->node1].node_tag == region)
         {  /* find node1 in region */
            temp_node = this->node2;
            if (vertex[temp_node].node_tag != 0) /* node2 is in a region */
            {
                this->weight = new_weight(region_sum[region],
                                  region_sum[vertex[temp_node].node_tag]);
                temp_re = vertex[temp_node].node_tag;
            }
            else  /* node2 is a single node */
```

```
            this->weight = new_weight(region_sum[region],
                                      get_pix_value(temp_node));
        }
        else
        {  /* find node2 in region */
           temp_node = this->node1;
           if (vertex[temp_node].node_tag != 0) /* node1 is in a region */
           {
               this->weight = new_weight(region_sum[region],
                         region_sum[vertex[temp_node].node_tag]);
               temp_re = vertex[temp_node].node_tag;
           }
           else  /* node1 is a single node */
              this->weight = new_weight(region_sum[region],
                                      get_pix_value(temp_node));
        }
        /* delete duplicate links for all nodes, no link in region */
        that = &dummy;
        dummy.other = this->other;
        if (!temp_re)   /* temp_node is single */
        {
            while ((find = that->other) != NULL)
            {  /* delete duplicate link, temp_node is single */
                if (find->node1 > temp_node)   break;
                if (find->node1 == temp_node || find->node2 == temp_node)
                {
                    /* delete link from whole list */
                    find->last->next = find->next;
                    find->next->last = find->last;
                    find->next = NULL;
                    /* delete link from compressed list */
                    that->other = find->other;
                    break;  /* only delete one link */
                }
                that = that->other;
            }
        }
        else
        {  /* temp_node is in a region */
            while ((find = that->other) != NULL)
            {  /* delete duplicate link */
               /* only one duplicate between two regions */
               if (vertex[find->node1].node_tag == temp_re ||
                   vertex[find->node2].node_tag == temp_re)
               {
                    find->last->next = find->next;
                    find->next->last = find->last;
                    find->next = NULL;
                    that->other = find->other;
                    break;
               }
               that = that->other;
            }  /* end while */
        }     /* end else */
        this = dummy.other;  /* reconnect and point to next */
    }       /* end while */
}

LINK * compress(this, vertex, region, biggest)
register LINK *this;
register LIST_HEAD *vertex;
register int region, biggest;
{
    LINK *find, *start;

    while (1)
    {   /* find the first link to start */
        if (vertex[this->node1].node_tag == region ||
            vertex[this->node2].node_tag == region)
            break;
        this = this->next;
    }
    start = find = this;
    this = this->next;
    while (this != &head)
    {   /* connect all relate link in a list */
        if (this->node1 > biggest)   break;
        if (vertex[this->node1].node_tag == region ||
            vertex[this->node2].node_tag == region)
        {
            find->other = this;
            find = this;
        }
        this = this->next;
    }
    find->other = NULL;  /* terminate */

    return(start);
}

LINK * jump(smallest)
```

```
int smallest;
{
     register int index;

     if (smallest < win_xsize+1) index = 0;
     else                        index = 4*(smallest-win_xsize-1);
     /* at least one undeleted link within range */
     while (edges[index].next == NULL)   ++index;

     /* return the first element in the list */
     return(&edges[index]);
}

float new_weight(new, old)
STAT new, old;
{
     int  k;
     float weight,
           new_mean[DIMENSION],
           old_mean[DIMENSION];

     for (k = 0; k < dimension; k++)
     {
         new_mean[k] = new.sum[k]/(float)new.num_vec;
         old_mean[k] = old.sum[k]/(float)old.num_vec;
     }
     weight = 0.0;
     for (k = 0; k < dimension; k++)
         weight += SQUARE(new_mean[k]-old_mean[k]);

     return(weight);
}

STAT add_struct(a, b)
STAT a, b;
{
     int  k;

     a.num_vec += b.num_vec;
     for (k = 0; k < dimension; k++) a.sum[k] += b.sum[k];

     return(a);
}

STAT get_pix_value(node)
register int node;
{
int  i, j, k;
        STAT a;

get_coordinates(node, &i, &j);
        a.num_vec = 1;
for (k = 0; k < dimension; k++)
    a.sum[k] = (float)image[k][i][j];

        return(a);
}

get_coordinates(node_pos, a, b)
register int node_pos;
register int *a, *b;
{
*a = node_pos / win_xsize;/* i */
*b = node_pos % win_xsize;/* j */
}

/*************************************************************************
 *  You are reading buseg2.c                                             *
 *  This program contains functions of bmirs1.c                          *
 *  This file contains function for bottom up approaches of segmention.*
 *  This functions is to generate segment from a forest, and output the*
 *  segmented images.                                                    *
 *  K S LAU 22-1-92                                                      *
 *************************************************************************/
#include "bu.h"

extern u_char image[DIMENSION][MAX_YSIZE][MAX_XSIZE];
extern u_char seg[DIMENSION][MAX_YSIZE][MAX_XSIZE];
extern  int  num_seg,
             dimension,
             top_x, top_y,
             win_xsize, win_ysize;
extern struct rasterfile  header;

TREE *          find_alone_node();
TREE *          depth_first_next();
LIST_HEAD *  depth_first_segment();
LIST_HEAD *     depth_first_tree();
LIST_HEAD *     depth_first_seg();

static int  nos_vertices;
```

```
static float mean[DIMENSION];
LIST_HEAD *new_ptr, *old_ptr;

image_file()
{
     FILE *fp;
     int item, i, a;
     char *path = "/home/image/output/seg",
          c[5],
          buf[50];

     header.ras_height = win_ysize;
     header.ras_width = win_xsize;
     header.ras_length = win_xsize*win_ysize;
     for (a = 0; a < dimension; a++)
     {
         strcpy(buf, path);
         strcpy(c, l64a((long)(a+2)));
         strcat(buf, c);
         strcat(buf, ".ras");
         fp = fopen(buf, "w");
         demand(fp, Cannot open file for output image);

         item = fwrite((char *)&header, sizeof(struct rasterfile), 1, fp);
         for (i = 0; i < win_ysize; i++)
             item += fwrite((char *)&seg[a][i][0], sizeof(char),
                            win_xsize, fp);
         demand(item == win_xsize*win_ysize+1, Write output file error);
         fclose(fp);
         printf("Segment image is stored in %s\n", buf);
     }
}

reset_tree_tmp(vertex, group_pixel, numpix)
LIST_HEAD *vertex;
int numpix, *group_pixel;
{
register int i;
TREE *this_node;

for (i = 0; i < numpix; i++)
{
   this_node = vertex[group_pixel[i]].node;
   while (this_node != NULL)
   {
          if (this_node->status == TMPVISIT)
           this_node->status = NOTVISIT;
          this_node = this_node->next;
   }
}
}

remove_root(rnode, vertex, group_pixel, clust, stack, rtag)
LIST_HEAD *vertex;
int rnode, *group_pixel, **clust, *stack;
char  *rtag;
{/* traverse a tree */
    LIST_HEAD *temp_list;
    TREE *next;
    int stack_pos, numpixel, done = 0;

    numpixel = 0;
    stack_pos = 0;
    clust[0] = &group_pixel[0]; /* point to start of new cluster */
    /* this tree is not a single node tree */
    next = find_alone_node(&vertex[rnode]);
    start_depth_first(group_pixel, stack, &stack_pos,
                      &numpixel, &vertex[rnode]-vertex, next);
    if (rtag[rnode] == ROOT)
    { rtag[rnode] = NROOT; done = 1; }
    else if (rtag[next->node_pos] == ROOT)
    { rtag[next->node_pos] = NROOT; done = 1; }
    if (done) { clust[1] = &group_pixel[numpixel]; return; }
    new_ptr = &vertex[next->node_pos];
    old_ptr = &vertex[rnode];
    do
    {  /* depth first next */
       temp_list = depth_first_tree(vertex, &stack_pos, &numpixel,
                                    group_pixel, stack, rtag, &done);
       old_ptr = new_ptr;
       new_ptr = temp_list;
       if (done) break;
    } while (stack_pos != 1);  /* stack != NULL */
    clust[1] = &group_pixel[numpixel];  /* point to end */
}

LIST_HEAD * depth_first_tree(vertex, stack_pos, numpixel,
                             group_pixel, stack, rtag, done)
LIST_HEAD *vertex;
int *group_pixel, *stack;
int *stack_pos, *numpixel, *done;
```

```
char *rtag;
{
    TREE *root;

    if ((root = depth_first_next(vertex, stack_pos, numpixel,
                                 group_pixel, stack)) == NULL)
        return(NULL);

    if (rtag[root->node_pos] == ROOT)/* there is two root, del one only */
        { rtag[root->node_pos] = NROOT; *done = 1; }

    return(&vertex[root->node_pos]);
}

int get_root(vertex, root, rtag)
LIST_HEAD *vertex, **root;
char  *rtag;
{
    int  i, j,
         num_node;

    num_node = win_xsize*win_ysize;
    for (i = j = 0; i < num_node; i++)
        if (rtag[i] == ROOT) root[j++] = &vertex[i];
    /* no need to initialise node_tag, use pointer arithmatic */

    return(j);/* number of segments */
}

get_segment(rnode, vertex, group_pixel, clust, segment, stack)
LIST_HEAD *vertex, **rnode;
int *group_pixel, **clust, *stack;
int segment;
{
    LIST_HEAD *temp_list;
    TREE *root;
    int k, t, stack_pos, root_pos, numpixel;

    numpixel = 0;   /* all root already visited */
    t = get_single_node(vertex, rnode, segment, group_pixel,
                        clust, &numpixel);
    if (t > 0)
    {
        printf("There are %d single point segments\n", t);
        output_single_node(group_pixel, clust, t);
    }
    for (root_pos = 0; t < segment; root_pos++, t++)
    {
       /* delete all single nodes from the rnode list */
       initialize();
       stack_pos = 0;
       clust[t] = &group_pixel[numpixel]; /* point to start of new cluster */
       while ((root = find_alone_node(rnode[root_pos])) == NULL)
              root_pos++;
       start_depth_first(group_pixel, stack, &stack_pos,
                         &numpixel, rnode[root_pos]->vertex, root);
       calculate_sum(rnode[root_pos]->vertex);
       calculate_sum(root->node_pos);
       nos_vertices += 2;
       new_ptr = &vertex[root->node_pos];
       old_ptr = rnode[root_pos];
       do
       {  /* depth first next */
           temp_list = depth_first_segment(vertex, &stack_pos,
                                           &numpixel, group_pixel, stack);
           old_ptr = new_ptr;
           new_ptr = temp_list;
       } while(stack_pos != 1);  /* stack != NULL */
       for (k = 0; k < dimension; k++)
            mean[k] /= nos_vertices;
       output_image(clust[t], group_pixel);
    }
    clust[t] = &group_pixel[numpixel];  /* point to end */
}

int get_single_node(vertex, rnode, num_root, group_pixel, clust, numpixel)
LIST_HEAD *vertex, **rnode;
int num_root, *group_pixel, **clust, *numpixel;
{
    TREE *this_tree;
    int i, j, k, num_seg;
    char test;

    num_seg = 0;
    for (k = 0; k < num_root; k++)
    {   /* go over all root */
        i = 0;
        this_tree = rnode[k]->node;
        while(this_tree != NULL)
        {  /* count number of node */
           if (this_tree->status == NOTVISIT)
```

```
            i++;  /* count any notvisit node */
         this_tree = this_tree->next;
      }
      if (i == 0)
      {  /* node eligible to be single node */
         test = UNIQUE;
         if (num_seg > 0)
         {  /* more than 1 segment */
            for (j = 0; j < num_seg; j++)
            {
                if (group_pixel[j] == rnode[k]-vertex)
                {  /* check each cluster */
                   test = DUPLICATE;puts("DUPLICATE");
                   break;
                }
            }
         }
         if (test == UNIQUE)
         {  /* put unique single node to the cluster list */
            clust[num_seg++] = &group_pixel[*numpixel];
            group_pixel[(*numpixel)++] = rnode[k]-vertex;
         }
      }
   }
   return(num_seg);  /* return number of single node */
}

initialize()
{
        int  k;

        nos_vertices = 0;
        for (k = 0; k < dimension; k++) mean[k] = 0.0;
}

calculate_sum(a)
int a;
{
int i, j, k;

get_coordinates(a,&i,&j);
for (k = 0; k < dimension; k++)
    mean[k] += (float)image[k][i][j];
}

output_single_node(group_pixel, clust, num_node)
int *group_pixel, **clust, num_node;
{
     register int i;

     for (i = 0; i < num_node; i++)
     {
         initialize();
         calculate_sum(group_pixel[i]);
         nos_vertices++;
         output_image(clust[i], group_pixel);
     }
}

start_depth_first(group_pixel, stack, stack_pos, numpix, root, node)
int *stack_pos, *numpix, *group_pixel, *stack, root;
TREE *node;
{
     stack[(*stack_pos)++] = root;        /* put root to stack */
     group_pixel[(*numpix)++] = root;     /* put root to cluster */
     stack[(*stack_pos)++] = node->node_pos;
     group_pixel[(*numpix)++] = node->node_pos;
     node->status = TMPVISIT;
}

LIST_HEAD * depth_first_segment(node_head, stack_pos, numpixel,
                                group_pixel, stack)
LIST_HEAD *node_head;
int *group_pixel, *stack;
int *stack_pos, *numpixel;
{
     TREE *root;

     if ((root = depth_first_next(node_head, stack_pos, numpixel,
                                  group_pixel, stack)) == NULL)
         return(NULL);
     calculate_sum(root->node_pos);
     nos_vertices++;

     return(&node_head[root->node_pos]);
}

TREE * find_alone_node(list)
LIST_HEAD *list;
{
    TREE *this;
```

```
    this = list->node;
    while(this != NULL)
    {
       if (this->status == NOTVISIT)
           return(this);
       this = this->next;
    }
    return(NULL);
}

delete_duplicate(vertex)
LIST_HEAD *vertex;
{
TREE *this;

      this = new_ptr->node;
      while(this != NULL)
      {  /* mark the duplicate node */
   if(this->node_pos == old_ptr-vertex)
   {
 this->status = TMPVISIT;
 break;
   }
          this = this->next;
      }
}

TREE * depth_first_next(vertex, stack_pos, numpix, group_pixel, stack)
LIST_HEAD *vertex;
int *stack_pos, *numpix;
int *group_pixel, *stack;
{
     TREE *root;

     /* delete duplicate node in the next tree list */
     delete_duplicate(vertex);
     if((root = find_alone_node(new_ptr)) == NULL)
     {
        while(root == NULL)
        {
              /* delete stack */
              *stack_pos -= 2;
              new_ptr = &vertex[stack[*stack_pos]];
              root = find_alone_node(&vertex[stack[(*stack_pos)++]]);
              if(*stack_pos == 1 && root == NULL)    return(NULL);
        }
     }
     /* put node to stack */
     stack[(*stack_pos)++] = root->node_pos;
     /* put node to notepad for reset */
     group_pixel[(*numpix)++] = root->node_pos;
     root->status = TMPVISIT;
     return(root);
}

output_image(clust_start, group_pixel)
int *clust_start, group_pixel[];
{
     /* write to screen with precalculated average value */
     int i, k, a, b, index;

     index = clust_start-group_pixel;  /* get the index of group pixel */
     for(i = 0; i < nos_vertices; i++ ,index++) /* number of pixel */
     {
        get_coordinates(group_pixel[index], &a, &b);
        for (k = 0; k < dimension; k++) seg[k][a][b] = mean[k];
     }
}
```

Appendix J

Programs of the Automatic Cloud Wind Scheme

```
/*************************************************************************
 * The automatic cloud motion wind vectors generation scheme includes:   *
 * 1) cmv12.c                                                            *
 * 2) io.c                                                               *
 * 3) geo.c                                                              *
 * 4) draw_vector3.c                                                     *
 * 5) rad_temp.c                                                         *
 * 6) regression.h                                                       *
 *************************************************************************/

/*************************************************************************
 *  You are reading regression.h                                         *
 *************************************************************************/

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include <pixrect/pixrect_hs.h>

#define    MAX_GCP      100/* number of ground control points */

#ifdef     FIRST
#define    NOS_EQN 3/* number of equations */
#endif

#ifdef     SECOND
#define    NOS_EQN      6
#endif

#ifdef     THIRD
#define    NOS_EQN      10
#endif

#define  NAUT_TO_KM   1.8532
/* 1 n mile in km, 1 n mile = 1' of great circle arc */

#define  TRUE     1
#define  FALSE    !TRUE
#define  SIN(x)   (sin(M_PI*(x)/180.0))
#define  COS(x)   (cos(M_PI*(x)/180.0))
#define  TAN(x)   (tan(M_PI*(x)/180.0))

#define    NOS_COL (NOS_EQN+1)
#define    X 0/* offset for coeff_of_multiple_determination */
#define    Y 1/* offset for coeff_of_multiple_determination */
#define    strsave(s)   (strcpy(malloc(strlen(s)+1),s))

extern char *optarg;
extern int   optind, opterr;

typedef struct {
double  x;
double  y;
}Coord;/* struct for coordinates */

typedef struct {
double  mean;
double  sd;
}Stat;/* struct for standardizing */

/*************************************************************************
 * You are reading cmv.h                                                 *
 *************************************************************************/

#include <stdio.h>
```

```
#include <math.h>
#include <string.h>
#include <malloc.h>
#include <cgipw.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
#include <pixrect/pixrect_hs.h>

/* for cmv12 cmv14 cmv15 to read cluster map */
#define    DIMENSION           2
#define    NO_CENTER           10/* number of cluster for tracking */

#define    VEC_LENGTH          18    /* maximum length of display vector */
/* use 1.0 for cmv7 cmv8 cmv9 cmv10 cmv11, 2.0 for other */
#ifndef    SCALE
#define    SCALE               1.0   /* scale for display */
#endif
#define    X_CORREL_THRESHOLD 80    /* number of pixel in target win */

/* constant for data array */
#define    IMAGE_SIZE         256   /* maximum image size for correlation */
#define    TARGET_ARRAY       32    /* max size for the target array */
#define    SEARCH_ARRAY       72    /* max size for the search array */

/* distance between target window and search window */
/* target window is in the centre of search window */
#define    WIN_OFFSET        20 /* default, can be changed */
#define    TARGET_SIZE       4 /* default, can be changed */
#define    SEARCH_SIZE       50 /* default, can be changed */
#define    SURF_SIZE         (SEARCH_SIZE-TARGET_SIZE+1)
#define    X_RESOLUTION      2.67  /* spatial resolution n mile, trivial */
#define    Y_RESOLUTION      2.67  /* spatial resolution n mile, trivial */

/* constant for display window, only for cmv7 cmv8 */
#define    DEFAULT_WIN_XSIZE   850
#define    DEFAULT_WIN_YSIZE   1000
#define    MIN_WIN_XSIZE       200
#define    MIN_WIN_YSIZE       200

#define    MAX(a,b)             (((a)>(b))?(a):(b))
#define    MIN(a,b)             (((a)<(b))?(a):(b))
#define    strsave(s)           (strcpy(malloc(strlen(s)+1),s))
#ifndef     SQUARE(x)
#define    SQUARE(x)      ((x)*(x))
#endif
#define    PI             3.141592654
#define  demand(fact, remark) {\
if (!(fact)) {\
    fprintf(stderr, "demand not met: fact\n");\
    fprintf(stderr, "remark\n");\
    exit(1);\
}\
}

typedef struct cloud_win_vector {
          float   speed;
          float   direction;
          float   lat;
          float   longt;
          float   temp;/* temperature */
          float   mean;/* mean of ir pixel, for cloud height */
          float   sd;/* standard devation of ir pixel */
          int     count;/* number of nonzero pixel in this template */
}CMV;

/* the followings are for regression and true distance calculation */
#define   NOS_EQN   10
#define   MAX_GCP   70

typedef struct {
        double  x;
        double  y;
}Coord;/* struct for coordinates */

typedef struct {
        double  mean;
        double  sd;
}Stat;/* struct for standardizing */

/*************************************************************************
 * You are reading file icp                                              *
 * These are the data use for geometrical rectificaiton.                 *
 *************************************************************************/
/*
 longitude   latitude    pixel   line
*/
  5.750000   58.966667   362.0   18.0
 10.750000   59.916667   416.0   11.0
 10.600000   57.733333   422.0   32.0
 10.883333   56.416667   432.0   44.0
```

```
 10.683333   54.750033   436.0   61.0
 14.183333   57.783333   465.0   33.0
 12.616667   56.033333   454.0   48.0
 16.416667   56.233333   501.0   48.0
 14.783333   55.283333   486.0   57.0

 -3.083333   58.633333   257.0   22.0
 -1.816667   57.500000   272.0   32.0
 -1.233333   54.583333   280.0   60.0
 -3.633333   54.533333   247.0   62.0
 -6.500000   55.683333   212.0   50.0
 -7.316667   57.150000   204.0   36.0
 -5.600000   54.500000   221.0   62.0
 -8.800000   54.700000   179.0   61.0

-10.216667   53.400000   156.0   74.0
-10.450000   52.100000   148.0   90.0
 -9.816667   51.450000   154.0   97.0
 -6.366667   52.166667   204.0   88.0
 -6.250000   53.333333   208.0   75.0
 -6.416667   54.016667   207.0   67.0
 -3.233333   54.116667   251.0   65.0
 -4.633333   53.316667   231.0   75.0
 -5.316667   51.916667   219.0   91.0
 -4.516667   51.033333   229.0  102.0
 -5.733333   50.050000   210.0  114.0
 -5.216667   49.933333   216.0  114.0
 -3.633333   50.216667   239.0  111.0
 -2.066667   50.566667   265.0  107.0
  1.400000   51.400000   316.0   97.0
  0.116667   53.566667   296.0   71.0
  1.583333   50.866667   319.0  103.0
 -1.250000   49.666667   276.0  118.0
 -1.850000   49.666667   267.0  117.0
 -4.733333   48.033333   222.0  139.0

 11.216667   54.500000   443.0   64.0
  8.700000   53.866667   412.0   70.0
  5.283333   52.700000   367.0   82.0


 13.633333   54.516667   475.0   64.0

 -1.516667   46.200000   270.0  164.0
 -1.166667   44.400000   275.0  190.0

 -5.850000   43.650000   195.0  201.0
 -9.000000   42.550000   138.0  219.0
 -9.383333   39.350000   123.0  269.0
 -8.866667   37.950000   129.0  293.0
 -8.933333   37.000000   125.0  309.0
 -5.600000   36.016667   187.0  325.0
 -2.200000   36.716667   253.0  313.0

  0.233333   38.733333   298.0  278.0
  2.316667   39.583333   338.0  264.0
  3.833333   40.033333   365.0  257.0
  3.233333   41.950000   351.0  227.0

  9.416667   42.950000   456.0  212.0
 10.533333   42.916667   476.0  213.0
 12.366667   44.933333   498.0  185.0

  9.516667   39.100000   470.0  274.0
```

```
/*****************************************************************************
 *  You are reading cmv12.c                                                  *
 *  This is the automatic cloud wind scheme use by the author, this          *
 *  program only use cross correlation, ssda and 2d search are not           *
 *  included.                                                                *
 *                                                                           *
 *  The target is either a visible or infared clustered or raw cloud image.
 *  The search area is raw image.                                            *
 *  The clustered image is a window in a 512x512 raw image.                  *
 *  The raw image is a window (330, 60) in a B format METEOSAT image.        *
 *  Use the cluster map generated by the Global-Local clutering algorithm,*
 *  and compute cmv for all clusters.                                        *
 *                                                                           *
 *  The whole image is divided into small nonoverlap target windows.         *
 *  If the number of pixels within the window is more than a thershold,      *
 *  the program will search for the motion, otherwise that window has no     *
 *  wind vector calculated.                                                  *
 *  display the wind field using pixwin with image on background.            *
 *  The wind field is calculated with geometric rectification.               *
 *                                                                           *
 *  The rectification is only valid to the following image.                  *
 *  The image which is extraction from a 512x512 b format image.            *
 *  The top left coordinates of the 512x512 image is (330,60)                *
 *  K.S.LAU 25-2-91                                                          *
 *****************************************************************************/
```

```
#include "cmv.h"

#define  UPPER_TARGET_SD_THRESHOLD  10.0
#define  LOWER_TARGET_SD_THRESHOLD  0.0
#define  SPEED_DIFF                 0.5/* differenc of two vector speed */
#define  DIRECT_DIFF                30.0/* difference of two vectors */
#define  TARGET_THRESHOLD   0.3/* mini percentage of pixel in target win */
#define  PASS               1
#define  FAIL               !PASS
#define  ONBOUNDARY         2/* peak on boundary */
#define  MAX_ROOT           25/* max number of cluster on surface */
#define  CLUSTERKEEPLIMIT   25/* cluster > this will be keep */
#define  PEAKSAMECLUSTER    1.05/* ratio of two peak if > ok */
#define  PEAKDIFFCLUSTER    1.5/* ratio of two peak if > ok */

char  title[50], progname[50], filename[50],
      ti_file[50], s2i_file[50],/* infrared target and search filename */
      tv_file[50], s2v_file[50],/* visible t and s filename */
      s1v_file[50], s1i_file[50],/* first search image */
      tvo_file[50],/* the visible original */
      tio_file[50],/* the original filename */
      clusmap[IMAGE_SIZE][IMAGE_SIZE];/* cluster map */
struct rasterfile t_header, s_header,/* for clustered image */
                  to_header;/* for original header */
unsigned char s1i[IMAGE_SIZE][IMAGE_SIZE],/* first infrared search */
              s1v[IMAGE_SIZE][IMAGE_SIZE],/* first visible search */
              s2i[IMAGE_SIZE][IMAGE_SIZE],/* infrared search window */
              ti[IMAGE_SIZE][IMAGE_SIZE],/* infrared target template */
              s2v[IMAGE_SIZE][IMAGE_SIZE],/* visible search window */
              tv[IMAGE_SIZE][IMAGE_SIZE],/* visible target template */
              tvo[IMAGE_SIZE][IMAGE_SIZE],/* original target(visible) */
              tio[IMAGE_SIZE][IMAGE_SIZE],/* original target(infrared) */
              ss1v[IMAGE_SIZE][IMAGE_SIZE],/* contrast stretch of original */
              ss2v[IMAGE_SIZE][IMAGE_SIZE],/* contrast stretch of original */
              stv[IMAGE_SIZE][IMAGE_SIZE];/* contrast stretch of original */
int target_size, search_size,
    win_offset,/* target window wrt search window */
    winxsize, winysize,/* for pixrect use, not for user */
    win_xsize, win_ysize,/* size of clustered area */
    nos_center,/* number of cluster */
    dimension,/* dimension of clusters */
    count[NO_CENTER],/* number of pixels in cluster */
    wholexsize, wholeysize,/* size of whole search area */
    xs, ys,/* top left of search window in search file */
    xso, yso,/* top left of search window w.r.t. original */
    size_difference;/* the number of cross correlation shift */
float  mean[DIMENSION][NO_CENTER],
       deviation[DIMENSION][NO_CENTER],
       get_standard_deviation(),
       mean_gradient();
CMV cloud_motion(),
    /* to store the output vectors */
    cmv[NO_CENTER][IMAGE_SIZE/TARGET_SIZE][IMAGE_SIZE/TARGET_SIZE];

main(argc, argv)
int argc; char **argv;
{
     char  answer[5], print[5],/* print wind field */
           track_type[5],/* choose v or i image for tracking */
           fill_average[5],/* fill zero pixel with average */
           display[5],/* how to display */
           show[5],/* draw matching surface */
           image_type[5],/* cluster image or raw */
           stretch[5];/* choose stretched image for tracking */
     int  i, clus;
     float  ir_threshold;/* threshold for ir clusters */
     Pixrect *pr, *mem;
     colormap_t  colormap;
     FILE  *fp;

     puts("CROSS CORRELATION TRACKING");
     load_data(print, show, track_type, fill_average, stretch, image_type);
     strcpy(progname, argv[0]);
     strcpy(filename, ti_file);
     get_regression_coeff();/* get formula to calculate true distance */
     if (strchr(image_type, 'y'))
     {/* clustered tracking */
         print_cluster_stat();
         printf("Enter the infrared pixel count threshold,\n");
         printf("clusters with ir mean lower than this will be ignored, ");
         scanf("%f", &ir_threshold);
         for (i = 0; i < nos_center; i++)
         {
             if (mean[1][i] < ir_threshold) continue;/* infrared threshold */
             extract_cluster(i);
             wind_vector(show, track_type, fill_average, stretch, i);
             rad_temp(i);/* convert infrared greylevel to temperature */
             if (strchr(print, 'y')) print_result(progname, ti_file, i);
         }
     }
     else
```

```
    {/* raw tracking */
        wind_vector(show, track_type, fill_average, stretch, 0);
        rad_temp(0);
        if (strchr(print, 'y')) print_result(progname, ti_file, 0);
    }
    mem = mem_create(t_header.ras_width, t_header.ras_height,
                     t_header.ras_depth);
    fp = fopen(tio_file, "r"); demand(fp, cannot open file for pr);
    pr = pr_load(fp, &colormap);
    pr_rop(mem, 0, 0, mem->pr_size.x, mem->pr_size.y, PIX_SRC, pr, 0, 0);
    pr_rop(pr, 0, 0, pr->pr_size.x, pr->pr_size.y, 0, (Pixrect *)NULL, 0, 0);
    if (strchr(image_type,'y'))
    {
        composite_wind_field();
        puts("Composite result");
        if (strchr(print, 'y')) print_result(progname, ti_file, nos_center);
        print_cluster_stat();
    }
    print_to_file();
    for (;;)
    {/* display wind field */
        printf("Do you want to display wind field? "); scanf("%s", answer);
        if (strchr(answer, 'n')) break;
        printf("Enter o (overlay), w (field only), m (on map), ");
        scanf("%s", display);
        printf("Enter the cluster you want to look, %d
                for composite field ",
               (nos_center > 1) ? nos_center : nos_center-1);
        scanf("%d", &clus);
        if (strchr(display, 'w')) display_wind_field_only(clus);
        if (strchr(display, 'o')) display_wind_field_onimage(clus, pr, mem);
        if (strchr(display, 'm')) display_wind_field_onmap(clus);
    }
}

wind_vector(show, track_type, fill_average, stretch, clus)
char  show[],/* show match surface, not used */
      track_type[],/* use infrared or visible */
      fill_average[],/* fill zero pixel with mean, not used */
      stretch[];/* preprocess visible, not used */
int   clus;/* which cluster */
{
    unsigned char mean,
                  target[SEARCH_ARRAY][SEARCH_ARRAY],
                  s1[SEARCH_ARRAY][SEARCH_ARRAY],
                  s2[SEARCH_ARRAY][SEARCH_ARRAY],
                  temp[IMAGE_SIZE][IMAGE_SIZE];/* for stretch */
    int count,
        numxwin, numywin,/* number of window in a row or column */
        t_x, t_y, s_x, s_y, /* coordinates of target and search win */
        i, j, s, t, /* for loop counter */
        pix_count; /* pixel count in a target window */
    float  v_sd, i_sd,/* standard deviation of visible and infrared target */
           sum, mean_speed, mean_direct;
    CMV  v1, v2;

    numxwin = numywin = (wholexsize-2*win_offset)/target_size;
    for (i = 0; i < numywin; i++)
       for (j = 0; j < numxwin; j++)
       {   /* get target coordinates */
           t_y = win_offset + i * target_size;
           t_x = win_offset + j * target_size;
           /* get search window coordinate */
           s_y = t_y - win_offset;
           s_x = t_x - win_offset;
           pix_count = 0;
           for (s = 0; s < target_size; s++)
              for (t = 0; t < target_size; t++) /* load target window */
                  if (ti[t_y+s][t_x+t] != 0)   pix_count++;
           if (pix_count < TARGET_THRESHOLD*SQUARE(target_size)) continue;

           if (strchr(track_type, 'v'))
           {
               if (strchr(stretch, 'y'))
               {/* use stretched visible */
                   for (s = 0; s < target_size; s++)
                       for (t = 0; t < target_size; t++)
                           target[s][t] = stv[t_y+s][t_x+t];
                   for (s = 0; s < search_size; s++)
                       for (t = 0; t < search_size; t++)
                       {
                           s1[s][t] = ss1v[s_y+s][s_x+t];
                           s2[s][t] = ss2v[s_y+s][s_x+t];
                       }
               }
               else
               {/* use unstretched visible */
                   for (s = 0; s < target_size; s++)
                       for (t = 0; t < target_size; t++)
                           target[s][t] = tv[t_y+s][t_x+t];
                   for (s = 0; s < search_size; s++)
```

```
                    for (t = 0; t < search_size; t++)
                    {
                        s1[s][t] = s1v[s_y+s][s_x+t];
                        s2[s][t] = s2v[s_y+s][s_x+t];
                    }
                }
            }
            else/* use infrared */
            {
                for (s = 0; s < target_size; s++)
                    for (t = 0; t < target_size; t++)
                        target[s][t] = ti[t_y+s][t_x+t];
                for (s = 0; s < search_size; s++)
                    for (t = 0; t < search_size; t++)
                    {
                        s1[s][t] = s1i[s_y+s][s_x+t];
                        s2[s][t] = s2i[s_y+s][s_x+t];
                    }
            }
            if (strchr(fill_average, 'y'))
            {/* if use clustered tracking, fill the other pixel with mean */
                sum = 0.; count = 0;
                for (s = 0; s < target_size; s++)
                    for (t = 0; t < target_size; t++)
                    {
                    if (target[s][t] == 0) continue;
                    sum += target[s][t];
                    count++;
                    }
                mean = sum/count;
                for (s = 0; s < target_size; s++)
                    for (t = 0; t < target_size; t++)
                        if (target[s][t] == 0) target[s][t] = mean;
            }
            v1 = cloud_motion(show, target, s1, i, j);
            if (v1.speed > 0.0)
                v2 = cloud_motion(show, target, s2, i, j);
            else v2.speed = 0.0;
            if (v1.direction > 180.0) v1.direction -= 180.0;
            else                      v1.direction += 180.0;
            mean_speed = (v1.speed+v2.speed)/2.0;
            mean_direct = (v1.direction+v2.direction)/2.0;
            /* check symmetric of the two vectors */
            if (fabs(v1.speed-v2.speed) > SPEED_DIFF*v1.speed ||
                fabs(v1.speed-v2.speed) > SPEED_DIFF*v2.speed ||
                fabs(v1.direction-v2.direction) > DIRECT_DIFF ||
                v1.speed == 0.0 || v2.speed == 0.0)
                ;
            else
            {
                cmv[clus][i][j].speed = mean_speed;
                cmv[clus][i][j].direction = mean_direct;
                cmv[clus][i][j].lat = v2.lat;
                cmv[clus][i][j].longt = v2.longt;
                cmv[clus][i][j].temp = v2.temp;
                cmv[clus][i][j].mean = v2.mean;
                cmv[clus][i][j].sd = v2.sd;
                cmv[clus][i][j].count = v2.count;
            }
        }
}

composite_wind_field()
{
    int i, j, k, numxwin, numywin,
        m_k, maxi;

    numxwin = numywin = (wholexsize-2*win_offset)/target_size;
    for (i = 0; i < numywin; i++)
        for (j = 0; j < numxwin; j++)
        {
            maxi = 0;
            for (k = 0; k < nos_center; k++)
            {
                if (cmv[k][i][j].speed > 0.0
                    && cmv[k][i][j].sd < UPPER_TARGET_SD_THRESHOLD
                    && cmv[k][i][j].sd > LOWER_TARGET_SD_THRESHOLD)
                {
                    if (cmv[k][i][j].count > maxi)
                    {
                        maxi = cmv[k][i][j].count;
                        m_k = k;
                    }
                }
            }
            cmv[nos_center][i][j].speed = cmv[m_k][i][j].speed;
            cmv[nos_center][i][j].direction = cmv[m_k][i][j].direction;
            cmv[nos_center][i][j].lat = cmv[m_k][i][j].lat;
            cmv[nos_center][i][j].longt = cmv[m_k][i][j].longt;
            cmv[nos_center][i][j].temp = cmv[m_k][i][j].temp;
            cmv[nos_center][i][j].mean = cmv[m_k][i][j].mean;
```

```
            cmv[nos_center][i][j].sd = cmv[m_k][i][j].sd;
            cmv[nos_center][i][j].count = cmv[m_k][i][j].count;
        }
}

CMV cloud_motion(show, target, search, shift_i, shift_j)
char  show[];
unsigned char target[][SEARCH_ARRAY],
              search[][SEARCH_ARRAY];
int shift_i,
    shift_j;/* determine which subwindow, in order to obtain coord. offset */
{
     register i, j, s, t;
     char  surfmap[SURF_SIZE][SURF_SIZE];
     int num_area,/* num of cluster on the surface */
         q,/* quality flag */
         max_i, max_j,/* index of max coeff. */
         smax1_i, smax1_j, smax2_i, smax2_j,/* peak on smoothed surface */
         xi, yi, x2, y2,/* coord. of the peak position */
         cxi, cyi,/* center coord. of target template */
         number[MAX_ROOT],/* num of element in earch cluster */
         count;/* count no. of non zero pixel */
     float sum, ssq, sd_t, sd_s, sd_st,
           mean_t, mean_s,/* statistic for xcorrelation */
           coefficient[SURF_SIZE][SURF_SIZE];
     double  lat1, long1, lat2, long2,/* start/finish lat. long. */
             departure, course;/* distance travel and course */
     CMV cmv;/* put result here */

     /* cross correlation */
     /* compute target sd */
     sum = ssq = 0.0; count = 0;
     for (i = 0; i < target_size; i++)
         for (j = 0; j < target_size; j++)
         {
             if (target[i][j] == 0) continue;
             sum += target[i][j];
             ssq += SQUARE((float)target[i][j]);
             count++;
         }
     mean_t = sum/(float)count;
     sd_t = ((float)count*ssq-SQUARE(sum))/SQUARE((float)count);
     sd_t = sqrt(sd_t);
     /* check temperature variation, if < 10C do not check */
     if (sd_t > UPPER_TARGET_SD_THRESHOLD || sd_t < LOWER_TARGET_SD_THRESHOLD)
     {   /* no motion bypass calculation */
         cmv.speed = cmv.direction = cmv.lat = cmv.longt = cmv.temp = 0.0;
         cmv.sd = sd_t; cmv.mean = mean_t; cmv.count = count;
         return(cmv);
     }
     cmv.mean = mean_t; cmv.sd = sd_t; cmv.count = count;

     for (i = 0; i < size_difference; i++)
         for (j = 0; j < size_difference; j++)
         {/* compute search template sd */
             sum = ssq = 0.; count = 0;
             for (s = 0; s < target_size; s++)
                 for (t = 0; t < target_size; t++)
                 {
                     if (target[s][t] == 0) continue;
                     sum += search[i+s][j+t];
                     ssq += SQUARE((float)search[i+s][j+t]);
                     count++;
                 }
             mean_s = sum/(float)count;
             sd_s = ((float)count*ssq-SQUARE(sum))/SQUARE((float)count);
             sd_s = sqrt(sd_s);
             /* compute covariance */
             sd_st = 0.;
             for (s = 0; s < target_size; s++)
                 for (t = 0; t < target_size; t++)
                 {
                     if (target[s][t] == 0) continue;
                     sd_st += ((float)search[i+s][j+t]-mean_s)*
                              ((float)target[s][t]-mean_t);
                 }
             sd_st /= (float)count;
             coefficient[i][j] = sd_st/(sd_s*sd_t);
         }

     x2 = max_j; y2 = max_i;
     x1 = xs+win_offset+shift_j*target_size;/* start */
     y1 = ys+win_offset+shift_i*target_size;
     if (q == FAIL || q == ONBOUNDARY)
     {/* if match point on boundary do not accept */
         cmv.speed = cmv.direction = cmv.lat = cmv.longt = cmv.temp = 0.0;
         return(cmv);
     }
     else if (max_i == win_offset && max_j == win_offset)
     {   /* no motion bypass lat and longt calculation */
         cmv.speed = 0.0;
```

```
        cmv.direction = 0.0;
    }
    else
    {/* compute true distance */
        x2 += xs+shift_j*target_size;/* finish */
        y2 += ys+shift_i*target_size;
        pixel_to_map((double)x1, (double)y1, (double)x2, (double)y2,
                     &lat1, &long1, &lat2, &long2);
        get_distance(lat1, long1, lat2, long2, &departure, &course);
        cmv.speed = departure/0.5;/* 30 minutes */
        cmv.direction = course;
    }
    cx1 = x1+target_size/2;/* center of target window */
    cy1 = y1+target_size/2;
    pixel_to_map((double)cx1, (double)cy1, (double)x2, (double)y2,
                 &lat1, &long1, &lat2, &long2);
    cmv.lat = lat1; cmv.longt = long1;

    return(cmv);
}

/*********************************************************************
 *  You are reading io.c                                             *
 *  Functions to display result and read write data,                  *
 *  for computaion of cloud motion winds.                            *
 *  K.S.LAU  5-3-91                                                  *
 *********************************************************************/
#include  "cmv.h"

load_data(print, show, track_type, fill_average, stretch, image_type)
char  show[],
      print[],
      track_type[],
      fill_average[],
      image_type[],
      stretch[];
{
char  clustermap[50];
int  i, j;
FILE  *fp;

puts("All input file should be infrared.");
puts("The visible file will follow.");
printf("Enter the type of image you use for tracking, v or i, ");
scanf("%s", track_type);
if (strchr(track_type, 'v'))
{
   printf("Do you want to stretch visible image? ");
   scanf("%s", stretch);
} else  strcpy(stretch, "n");
printf("Do you want to fill zero pixel? ");
scanf("%s", fill_average);
printf("Do you want to display all matching surface? ");
scanf("%s", show);
printf("Do you want to print wind field result? ");
scanf("%s", print);
printf("Enter the size of window in which ");
printf("tracking window is define,\n");
printf("xsize ysize: "); scanf("%d%d", &wholexsize, &wholeysize);
demand(wholexsize <= IMAGE_SIZE,  tracking window too large);
printf("Enter 'y' if you want to use cluster map,\n");
printf("'n' if you want to use clustered image directly, ");
scanf("%s", image_type);
if (strchr(image_type, 'y'))
{
    printf("Enter the cluster map filename, ");
    scanf("%s", clustermap);
}
printf("Enter the target filename: ");
if (strchr(image_type, 'y')) printf("original required ");
scanf("%s", ti_file);
if (strchr(image_type, 'y')) strcpy(tio_file, ti_file);
else
{
    printf("Enter the original filename ");
           printf("corresponding to the target area: ");
           scanf("%s", tio_file);
}
printf("Enter the 1st search filename: "); scanf("%s", s1i_file);
printf("Enter the 2nd search filename: "); scanf("%s", s2i_file);
printf("Enter the target window size: "); scanf("%d", &target_size);
printf("Enter the search window size: "); scanf("%d", &search_size);
demand(search_size >= target_size+2, window size contradiction);
printf("Enter the x and y coord. of whole ");
printf("search area in clustered area, \nwrt the cluster window: ");
scanf("%d%d", &xs, &ys);
if (strchr(image_type, 'y'))
{/* read cluster map */
    fp = fopen(clustermap, "r"); demand(fp, Cannot open clusmap);
    fread((char *)&nos_center, sizeof(int), 1, fp);
    fread((char *)&dimension, sizeof(int), 1, fp);
```

```
    fread((char *)&xso, sizeof(int), 1, fp);
    fread((char *)&yso, sizeof(int), 1, fp);
    fread((char *)&win_xsize, sizeof(int), 1, fp);
    fread((char *)&win_ysize, sizeof(int), 1, fp);
    demand(win_xsize*win_ysize<=IMAGE_SIZE*IMAGE_SIZE,
           IMAGE_SIZE too small);
    fread((char *)count, sizeof(int), nos_center, fp);
    for (i = 0; i < nos_center; i++)
        for (j = 0; j < dimension; j++)
            fread((char *)&mean[j][i], sizeof(float), 1, fp);
    for (i = 0; i < nos_center; i++)
        for (j = 0; j < dimension; j++)
            fread((char *)&deviation[j][i], sizeof(float), 1, fp);
    for (i = 0; i < win_ysize; i++)
        fread((char *)&clusmap[i][0], sizeof(char), win_xsize, fp);
    fclose(fp);
    printf("Coords. of cluster area x %d y %d\n", xso, yso);
}
else
{
    printf("Enter the x and y coord. of ");
           printf("cluster window wrt raw image: ");
           scanf("%d%d", &xso, &yso);
    puts("NB: Clustered image is either the same as the raw image");
    puts("    or smaller size entirely of clustered area.");
    nos_center = 1;/* use raw image for matching */
}
xs += xso; ys += yso;
size_difference = search_size-target_size+1;
win_offset = (search_size-target_size)/2;

read_file(ti_file, s2i_file, tio_file, ti, s2i, tio);
read_file(ti_file, s1i_file, tio_file, ti, s1i, tio);

change_fname(ti_file, tv_file);
change_fname(s2i_file, s2v_file);
change_fname(s1i_file, s1v_file);
change_fname(tio_file, tvo_file);
read_file(tv_file, s2v_file, tvo_file, tv, s2v, tvo);
read_file(tv_file, s1v_file, tvo_file, tv, s1v, tvo);
}

read_file(tf, sf, tof, t, s, t_o)
char  tf[], sf[], tof[];
u_char  t[][IMAGE_SIZE], s[][IMAGE_SIZE],
        t_o[][IMAGE_SIZE];
{
     int  i, numread,
          clus_x, clus_y,
          raw_x, raw_y;
     FILE  *ft, *fs, *fto;

     /* open all files */
     ft = fopen(tf, "r"); demand(ft, Cannot open target file.);
     fs = fopen(sf, "r"); demand(fs, Cannot open search file.);
     fto = fopen(tof, "r"); demand(fto, Cannot open original file.);
     fread((char *)&s_header, sizeof(struct rasterfile), 1, fs);
     fread((char *)&t_header, sizeof(struct rasterfile), 1, ft);
     fread((char *)&to_header, sizeof(struct rasterfile), 1, fto);
     raw_x = xs; raw_y = ys;
     clus_x = xs; clus_y = ys;/* initialise for fseek */
     /* read search file */
     numread = 0;
     fseek(fs, s_header.ras_width*clus_y+clus_x, 1);  /* goto start */
     for (i = 0; i < wholeysize; i++)
     {
          numread += fread((char *)(*(s+i)), sizeof(char), wholexsize, fs);
          fseek(fs, s_header.ras_width -wholexsize, 1);
     }
     demand(numread == wholexsize*wholeysize, read error);
     /* read target file */
     numread = 0;
     fseek(ft, t_header.ras_width*clus_y+clus_x, 1);  /* goto start */
     for (i = 0; i < wholeysize; i++)
     {
         numread += fread((char *)(*(t+i)), sizeof(char), wholexsize, ft);
         fseek(ft, t_header.ras_width -wholexsize, 1);
     }
     demand(numread == wholexsize*wholeysize, read error);
     /* read target original */
     numread = 0;
     fseek(fto, to_header.ras_width*raw_y+raw_x, 1);
     for (i = 0; i < wholeysize; i++)
     {
          numread += fread((char *)(*(t_o+i)),
                           sizeof(char), wholexsize, fto);
          fseek(fto, to_header.ras_width -wholexsize, 1);
     }
     demand(numread == wholexsize*wholeysize, read error);

     fclose(ft); fclose(fs); fclose(fto);
```

```
}

change_fname(from, to)
char  from[], to[];
{
     int  i, j, change_count = 0;

     /* change infrared name to visible name */
     for (i = strlen(from); i >= 0; i--) if (from[i] == '/') break;
     for (j = 0; j < i; j++) to[j] = from[j];
     for (j = i; j <= strlen(from); j++)
     {
         if (from[j] == 'i' && change_count < 1)
         {   to[j] = 'v';change_count++;  }
         else                to[j] = from[j];
     }
}

print_cluster_stat()
{
int  k, class;

printf("\t***** CENTERS *****");
printf("\n     ");
for (class = 0; class < nos_center; class++)
    printf(" c%2d     ", class);
for (k = 0; k < dimension; k++)
{
    printf("\n"); printf("%2d: ", k);
    for (class = 0; class < nos_center; class++)
        printf("%5.1f,  ", mean[k][class]);
}printf("\n");

printf("\t**** NUMBER OF OBJECTS PER CLUSTER ****\n");
for (class = 0; class < nos_center; class++)
    printf("   c%2d  ", class);
printf("\n");
for (class = 0; class < nos_center; class++)
    printf("%6d  ", count[class]);
printf("\n");

printf("\t***** STANDARD DEVIATION *****");
printf("\n     ");
for (class = 0; class < nos_center; class++)
    printf("  c%2d   ", class);
for (k = 0; k < dimension; k++)
{
    printf("\n"); printf("%2d: ", k);
    for (class = 0; class < nos_center; class++)
        printf("%5.1f,  ", deviation[k][class]);
}printf("\n");
}

print_to_file()
{/* print cloud wind vector to a file */
     char /**outfile*/ outfile[50];
     int i, j, k, numxwin, numywin;
     float  direction;
     FILE  *fp;

     numxwin = numywin = (wholexsize-2*win_offset)/target_size;
     /*outfile = "/home/image/output/wf";*/
     do {
         printf("Enter the filename for computed wind field, ");
         scanf("%s", outfile);
         fp = fopen(outfile, "w");
         if (!fp) printf("Cannot open file, please try again!!\n");
     } while(!fp);
     fwrite((char *)&nos_center, sizeof(int), 1, fp);
     fwrite((char *)&numxwin, sizeof(int), 1, fp);
     fwrite((char *)&numywin, sizeof(int), 1, fp);
     fwrite((char *)&win_offset, sizeof(int), 1, fp);
     fwrite((char *)&target_size, sizeof(int), 1, fp);
     fwrite((char *)&search_size, sizeof(int), 1, fp);
     fwrite((char *)&xs, sizeof(int), 1, fp);
     fwrite((char *)&ys, sizeof(int), 1, fp);
     fwrite((char *)&xso, sizeof(int), 1, fp);
     fwrite((char *)&yso, sizeof(int), 1, fp);
     fwrite((char *)&win_xsize, sizeof(int), 1, fp);
     fwrite((char *)&win_ysize, sizeof(int), 1, fp);
     fwrite((char *)&wholexsize, sizeof(int), 1, fp);
     fwrite((char *)&wholeysize, sizeof(int), 1, fp);

     for (k = 0; k < nos_center; k++)
     {
         for (i = 0; i < numywin; i++)
             for (j = 0; j < numxwin; j++)
                 fwrite((char *)&cmv[k][i][j].lat, sizeof(float), 1, fp);
         for (i = 0; i < numywin; i++)
             for (j = 0; j < numxwin; j++)
                 fwrite((char *)&cmv[k][i][j].longt, sizeof(float), 1, fp);
```

```
        for (i = 0; i < numywin; i++)
            for (j = 0; j < numxwin; j++)
                fwrite((char *)&cmv[k][i][j].speed, sizeof(float), 1, fp);
        for (i = 0; i < numywin; i++)
            for (j = 0; j < numxwin; j++)
            {
                if (cmv[k][i][j].speed == 0.0)
                    direction = cmv[k][i][j].direction;
                else if (cmv[k][i][j].direction < 180.0)
                    direction = 180.0+cmv[k][i][j].direction;
        else
            direction = cmv[k][i][j].direction-180.0;
                fwrite((char *)&direction, sizeof(float), 1, fp);
            }
        for (i = 0; i < numywin; i++)
            for (j = 0; j < numxwin; j++)
                fwrite((char *)&cmv[k][i][j].temp, sizeof(float), 1, fp);
        for (i = 0; i < numywin; i++)
            for (j = 0; j < numxwin; j++)
                fwrite((char *)&cmv[k][i][j].count, sizeof(int), 1, fp);
    }
    fclose(fp);
}

print_result(prog, t_file, clus)
char *prog, *t_file;
int  clus;
{
    int i, j, numxwin, numywin;

    numxwin = numywin = (wholexsize-2*win_offset)/target_size;
    printf("\nProg: %s Target: %s", prog, t_file);
    printf("\nCluster %d", clus);
    printf("\nSpeed in knots, direction in degree");
    for (i = 0; i < numywin; i++)
    {
 printf("\n\n");
 printf("speed  ");
        for (j = 0; j < numxwin; j++)
            printf("  %5.1f", cmv[clus][i][j].speed);
        printf("\n");
        printf("direct ");
        for (j = 0; j < numxwin; j++)
        {
            if (cmv[clus][i][j].speed == 0.0)
                printf("  %5.1f", cmv[clus][i][j].direction);
            else if (cmv[clus][i][j].direction < 180.0)
        printf("  %5.1f", 180.0+cmv[clus][i][j].direction);
    else
        printf("  %5.1f", cmv[clus][i][j].direction-180.0);
 }
        printf("\n");
        printf("temp   ");
        for (j = 0; j < numxwin; j++)
    printf("  %5.1f", cmv[clus][i][j].temp);
        printf("\n");
        printf("lat    ");
        for (j = 0; j < numxwin; j++)
    printf("  %5.1f", cmv[clus][i][j].lat);
        printf("\n");
        printf("long   ");
        for (j = 0; j < numxwin; j++)
    printf("  %5.1f", cmv[clus][i][j].longt);
        printf("\n");
        printf("mean   ");
        for (j = 0; j < numxwin; j++)
    printf("  %5.1f", cmv[clus][i][j].mean);
        printf("\n");
        printf("sd     ");
        for (j = 0; j < numxwin; j++)
    printf("  %5.1f", cmv[clus][i][j].sd);
        printf("\n");
        printf("count  ");
        for (j = 0; j < numxwin; j++)
    printf("  %5d", cmv[clus][i][j].count);
    }
    printf("\n");
}

extract_cluster(clus)
int  clus;
{/* extract a cluster for tracking */
register int  i, j;

for (i = 0; i < wholeysize; i++)
    for (j = 0; j < wholexsize; j++)
        ti[i][j] = tv[i][j] = 0;
for (i = 0; i < wholeysize; i++)
    for (j = 0; j < wholexsize; j++)
        if (clusmap[i+ys-yso][j+xs-xso] == clus)
        {
```

```
            ti[i][j] = tio[i][j];
            tv[i][j] = tvo[i][j];
        }
}

/*************************************************************************
 *  You are reading geo.c                                                *
 *  Function to do geometric rectification and calculate great circle    *
 *  distance on earth. For calculation of true wind speed on METEOSAT    *
 *  images.                                                              *
 *  The rectification only apply to a window in a                        *
 *  B (line 1810-2434) format image.                                     *
 *  The coord. of the window is (330,60), size 512x512                   *
 *  K.S.LAU 16-7-90                                                      *
 *************************************************************************/
#define  THIRD/* use thrid order polynomial

#include "/home/lau/src/map/regression.h"

/* variable for regression and true distance */
static int      num_gcp; /* number of ground control points */
static
double   indepvar[MOS_EQN][MOS_EQN],/* matrix for indepdent variable */
         depvar_l[MOS_EQN],/* array for depdent variable, line */
         depvar_p[MOS_EQN],/* array for depdent variable, pixel */
         coeffx[MOS_EQN],/* the estimated coefficients for mapped image x */
         coeffy[MOS_EQN];/* the estimated coefficients for mapped image y */
static Coord    image[MAX_GCP],/* store image pixel coord */
                map[MAX_GCP];/* store map coord */
static Stat     mapx_stat,/* mean and variance of mapx */
                mapy_stat;/* mean and variance of mapy */

get_regression_coeff()/* calculate the regression formula */
{
        FILE    *stream;

        /* read image control point not ground control point,
           because we want pixel coord. to be indepdent variables */
        if ((stream = fopen("/home/lau/src/map/icp", "r")) == NULL)
            { fprintf(stderr, "Cannot open datafile.\n");  exit(1); }

        read_gcp(stream);/* read image control point */
        standardizing_variable();
        setup_matrix();
        gauss_elimination(depvar_p, coeffx);/* solve for x */
        gauss_elimination(depvar_l, coeffy);/* solve for y */
}

read_gcp(stream)
FILE *stream;
{
        int i;

        i = num_gcp = 0;
        while (fscanf(stream, "%lf %lf %lf %lf", &image[i].x, &image[i].y,
                    &map[i].x, &map[i].y) != EOF)/* read all control point */
        { i++; num_gcp++; }
        fclose(stream);
}

pixel_to_map(p1, l1, p2, l2, lat1, long1, lat2, long2)
double   p1, l1, p2, l2,/* start/finish pixel coord. */
         *lat1, *long1, *lat2, *long2;/* start/finish lat. and long. */
{/* covert METEOSAT pixel to true geometric position */
        get_lat_long(p1, l1, lat1, long1);
        get_lat_long(p2, l2, lat2, long2);
}

get_lat_long(p, l, lati, longti)/* use 3th order regression */
double  p, l,/* pixel (x) and line (y) coord. */
        *lati, *longti;
{
        p = (p-mapx_stat.mean)/mapx_stat.sd;/* standardize */
        l = (l-mapy_stat.mean)/mapy_stat.sd;
        *longti = coeffx[0]+coeffx[1]*p+coeffx[2]*l
                  +coeffx[3]*pow(p, 2.0)+coeffx[4]*p*l
                  +coeffx[5]*pow(l, 2.0)+coeffx[6]*pow(p, 3.0)
                  +coeffx[7]*pow(p, 2.0)*l
                  +coeffx[8]*p*pow(l, 2.0)+coeffx[9]*pow(l, 3.0);
        *lati = coeffy[0]+coeffy[1]*p+coeffy[2]*l
                +coeffy[3]*pow(p, 2.0)+coeffy[4]*p*l
                +coeffy[5]*pow(l, 2.0)+coeffy[6]*pow(p, 3.0)
                +coeffy[7]*pow(p, 2.0)*l
                +coeffy[8]*p*pow(l, 2.0)+coeffy[9]*pow(l, 3.0);
}

get_distance(lat1, long1, lat2, long2, departure, course)
double lat1, lat2,
       long1, long2,
       *departure, *course;
{
```

```
/* Calculate the distance & course between 2 points given their latitudes
   & longitudes, using oblique spherical trigonometry. If the latitudes
   or longitudes are the same, then simplified expressions are used.
   One minute equal to one nautical mile.
 */
        int     complement;
        double  co_lat1, co_lat2,/* start/finish co-latitude */
                d_long,/* difference of longitude */
                alpha,/* initial course */
                distance;/* departure */

        if ((long1 < 0.0 && long2 < 0.0)||(long1 > 0.0 && long2 > 0.0))
            d_long = fabs(long1-long2);
        else
            d_long = fabs(long1)+fabs(long2);
        if (d_long > 180.0)
        {
            d_long = 360.0-d_long;
            complement = TRUE;
        }
        co_lat1 = 90.0-lat1;/* north pole as C */
        co_lat2 = 90.0-lat2;
        if (lat1 == lat2)
        {
            distance = 60.0*d_long;/* convert to minutes */
            distance *= COS(lat1);
            if (long2 > long1) alpha = 90.0;/* due east */
            else               alpha = 270.0;/* due west */
        }
        else if (long1 == long2)
        {/* always on great circle */
            distance = 60.0*fabs(lat1-lat2);
            if (lat2 > lat1) alpha = 0.0;/* due north */
            else             alpha = 180.0;/* due south */
        }
        else
        {
            distance = COS(co_lat1)*COS(co_lat2)
                      +SIN(co_lat1)*SIN(co_lat2)*COS(d_long);
            distance = acos(distance);
            distance *= 60.0*180.0/M_PI;/* in minutes */

            alpha = (COS(co_lat2)-COS(co_lat1)*COS(distance/60.0))
                   /(SIN(co_lat1)*SIN(distance/60.0));
            alpha = acos(alpha);
            alpha *= 180.0/M_PI;/* convert to degree */
            if ((long1 > 0.0 && long2 < 0.0)||(long1 < 0.0 && long2 > 0.0))
            {
                if ((long2 < long1 && complement == FALSE)||
                    (long2 > long1 && complement == TRUE))
                    alpha = 360.0-alpha;
            }
            else/* both east or both west */
                if (long2 < long1) alpha = 360.0-alpha;
        }
        *departure = distance;/* in n mile */
        *course = alpha;/* in degree */
}

gauss_elimination(depvar, coeff)/* use pivot to reduce computional error */
double  depvar[], coeff[];
{
        int  i, j, k, kp1, ip1, loop, pivot;
        double temp, sum, quot, big, absolute,
               augmented[MOS_EQN][MOS_COL];/* augmented matrix */

        for (i = 0; i < MOS_EQN; i++)/* copy to augmented matrix */
            for (j = 0; j < MOS_EQN; j++)
                augmented[i][j] = indepvar[i][j];
        for (i = 0; i < MOS_EQN; i++)
            augmented[i][MOS_COL-1] = depvar[i];

        for (k = 0; k < MOS_EQN-1; k++)
        {
            pivot = k;
            big = fabs(augmented[k][k]);
            kp1 = k+1;
            /* search for largest possible pivot element */
            for (i = kp1; i < MOS_EQN; i++)
            {
                absolute = fabs(augmented[i][k]);
                if (big < absolute) { big = absolute; pivot = i; }
                else                continue;
            }
            if (pivot != k)/* decision on necessity of row interchange */
                for (j = k; j < MOS_COL; j++)/* row interchange */
                {
                    temp = augmented[pivot][j];
                    augmented[pivot][j] = augmented[k][j];
                    augmented[k][j] = temp;
                }
```

```
            for (i = kp1; i < MOS_EQN; i++)
            {   /* calculation of elements of new matrix */
                quot = augmented[i][k]/augmented[k][k];
                for (j = kp1; j < MOS_COL; j++)
                    augmented[i][j] -= quot*augmented[k][j];
            }
            for (i = kp1; i < MOS_EQN; i++)   augmented[i][k] = 0.0;
        }
        /* back substitution */
        coeff[MOS_EQN-1] = augmented[MOS_EQN-1][MOS_COL-1]
                         /augmented[MOS_EQN-1][MOS_EQN-1];/* last coeff */
        for (loop = 0; loop < MOS_EQN-1; loop++)
        {
            sum = 0.0;
            i = MOS_EQN-2-loop;
            ip1 = i+1;
            for (j = ip1; j < MOS_EQN; j++)
                sum += augmented[i][j]*coeff[j];
            coeff[i] = (augmented[i][MOS_COL-1]-sum)/augmented[i][i];
        }
}

standardizing_variable()/* to reduce computational error */
{
        register int i;
        double  sum_xsquare=0.0, square_sumx=0.0, sumx=0.0,/* for map x */
                sum_ysquare=0.0, square_sumy=0.0, sumy=0.0,/* for map y */
                count;

        count = (double)num_gcp;
        for (i = 0; i < num_gcp; i++)/* get sum */
        {
            sumx += map[i].x;
            sumy += map[i].y;
            sum_xsquare += pow(map[i].x, 2.0);
            sum_ysquare += pow(map[i].y, 2.0);
        }
        square_sumx = pow(sumx, 2.0);
        square_sumy = pow(sumy, 2.0);
        mapx_stat.mean = sumx/count;/* mean */
        mapy_stat.mean = sumy/count;
        mapx_stat.sd = (sum_xsquare-square_sumx/count)/(count-1.0);/* var */
        mapy_stat.sd = (sum_ysquare-square_sumy/count)/(count-1.0);
        mapx_stat.sd = sqrt(mapx_stat.sd);/* sd */
        mapy_stat.sd = sqrt(mapy_stat.sd);
        for (i = 0; i < num_gcp; i++)/* standardizing */
        {
            map[i].x = (map[i].x-mapx_stat.mean)/mapx_stat.sd;
            map[i].y = (map[i].y-mapy_stat.mean)/mapy_stat.sd;
        }
}

setup_matrix()
{/* THRID ORDER REGRESSION MATRIX */
        register int  i;
        double        sum;

        /* setup matrix A first */
        indepvar[0][0] = (double)num_gcp;/* diagonal elements first */
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 2.0);
        indepvar[1][1] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].y, 2.0);
        indepvar[2][2] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 4.0);
        indepvar[3][3] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].x*map[i].y, 2.0);
        indepvar[4][4] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].y, 4.0);
        indepvar[5][5] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 6.0);
        indepvar[6][6] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++)
            sum += pow(map[i].x, 4.0)*pow(map[i].y, 2.0);
        indepvar[7][7] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++)
            sum += pow(map[i].x, 2.0)*pow(map[i].y, 4.0);
        indepvar[8][8] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += pow(map[i].y, 6.0);
        indepvar[9][9] = sum;


        sum = 0.0;/* row element */
```

```
for (i = 0; i < num_gcp; i++) sum += map[i].x;
indepvar[0][1] = indepvar[1][0] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += map[i].y;
indepvar[0][2] = indepvar[2][0] = sum;
indepvar[0][3] = indepvar[3][0] = indepvar[1][1];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += map[i].x*map[i].y;
indepvar[0][4] = indepvar[4][0] = sum;
indepvar[0][5] = indepvar[5][0] = indepvar[2][2];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 3.0);
indepvar[0][6] = indepvar[6][0] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 2.0)*map[i].y;
indepvar[0][7] = indepvar[7][0] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += map[i].x*pow(map[i].y, 2.0);
indepvar[0][8] = indepvar[8][0] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].y, 3.0);
indepvar[0][9] = indepvar[9][0] = sum;


indepvar[1][2] = indepvar[2][1] = indepvar[0][4];
indepvar[1][3] = indepvar[3][1] = indepvar[0][6];
indepvar[1][4] = indepvar[4][1] = indepvar[0][7];
indepvar[1][5] = indepvar[5][1] = indepvar[0][8];
indepvar[1][6] = indepvar[6][1] = indepvar[3][3];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 3.0)*map[i].y;
indepvar[1][7] = indepvar[7][1] = sum;
indepvar[1][8] = indepvar[8][1] = indepvar[4][4];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += map[i].x*pow(map[i].y, 3.0);
indepvar[1][9] = indepvar[9][1] = sum;



indepvar[2][3] = indepvar[3][2] = indepvar[0][7];
indepvar[2][4] = indepvar[4][2] = indepvar[0][8];
indepvar[2][5] = indepvar[5][2] = indepvar[0][9];
indepvar[2][6] = indepvar[6][2] = indepvar[1][7];
indepvar[2][7] = indepvar[7][2] = indepvar[1][8];
indepvar[2][8] = indepvar[8][2] = indepvar[1][9];
indepvar[2][9] = indepvar[9][2] = indepvar[5][5];

indepvar[3][4] = indepvar[4][3] = indepvar[2][6];
indepvar[3][5] = indepvar[5][3] = indepvar[2][7];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 5.0);
indepvar[3][6] = indepvar[6][3] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 4.0)*map[i].y;
indepvar[3][7] = indepvar[7][3] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++)
    sum += pow(map[i].x, 3.0)*pow(map[i].y, 2.0);
indepvar[3][8] = indepvar[8][3] = sum;
sum = 0.0;
for (i = 0; i < num_gcp; i++)
    sum += pow(map[i].x, 2.0)*pow(map[i].y, 3.0);
indepvar[3][9] = indepvar[9][3] = sum;


indepvar[4][5] = indepvar[5][4] = indepvar[2][8];
indepvar[4][6] = indepvar[6][4] = indepvar[3][7];
indepvar[4][7] = indepvar[7][4] = indepvar[3][8];
indepvar[4][8] = indepvar[8][4] = indepvar[3][9];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].y, 4.0)*map[i].x;
indepvar[4][9] = indepvar[9][4] = sum;

indepvar[5][6] = indepvar[6][5] = indepvar[4][7];
indepvar[5][7] = indepvar[7][5] = indepvar[4][8];
indepvar[5][8] = indepvar[8][5] = indepvar[4][9];
sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].y, 5.0);
indepvar[5][9] = indepvar[9][5] = sum;

sum = 0.0;
for (i = 0; i < num_gcp; i++) sum += pow(map[i].x, 5.0)*map[i].y;
indepvar[6][7] = indepvar[7][6] = sum;
indepvar[6][8] = indepvar[8][6] = indepvar[7][7];
sum = 0.0;
for (i = 0; i < num_gcp; i++)
    sum += pow(map[i].x*map[i].y, 3.0);
indepvar[6][9] = indepvar[9][6] = sum;

indepvar[7][8] = indepvar[8][7] = indepvar[6][9];
indepvar[7][9] = indepvar[9][7] = indepvar[8][8];
```

```
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += map[i].x*pow(map[i].y, 5.0);
        indepvar[8][9] = indepvar[9][8] = sum;


        /* setup depdent variable column for pixel */
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x;
        depvar_p[0] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*map[i].x;
        depvar_p[1] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*map[i].y;
        depvar_p[2] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*pow(map[i].x, 2.0);
        depvar_p[3] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*map[i].x*map[i].y;
        depvar_p[4] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*pow(map[i].y, 2.0);
        depvar_p[5] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*pow(map[i].x, 3.0);
        depvar_p[6] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++)
            sum += image[i].x*pow(map[i].x, 2.0)*map[i].y;
        depvar_p[7] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++)
            sum += image[i].x*pow(map[i].y, 2.0)*map[i].x;
        depvar_p[8] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].x*pow(map[i].y, 3.0);
        depvar_p[9] = sum;


        /* setup depdent variable column for line */
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y;
        depvar_l[0] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*map[i].x;
        depvar_l[1] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*map[i].y;
        depvar_l[2] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*pow(map[i].x, 2.0);
        depvar_l[3] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*map[i].x*map[i].y;
        depvar_l[4] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*pow(map[i].y, 2.0);
        depvar_l[5] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*pow(map[i].x, 3.0);
        depvar_l[6] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++)
            sum += image[i].y*pow(map[i].x, 2.0)*map[i].y;
        depvar_l[7] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++)
            sum += image[i].y*pow(map[i].y, 2.0)*map[i].x;
        depvar_l[8] = sum;
        sum = 0.0;
        for (i = 0; i < num_gcp; i++) sum += image[i].y*pow(map[i].y, 3.0);
        depvar_l[9] = sum;

}

/*************************************************************************
 *  You are read draw_vector3.c                                          *
 *  source to draw_vector, use in cmv12.c                                *
 *  using the image as the background, vector overlay on it              *
 *  K.S.LAU  27-2-91                                                     *
 *************************************************************************/
#include "cmv.h"
/* for display image on map */
#define  UPPER_LAT  (60.0)
#define  LEFT_LONG  (-25.5)
#define  EARTH_RADIUS  (6378.0)
#define  PIX_SIZE   (8.0)

static colormap_t      colormap;
```

```
static Frame            frame;
static Canvas           canvas;
static Cursor           cursor;
static Icon             icon;
static Ccgiwin          vpw;
static Pixwin           *pw, *fpw;
static Pixrect          *pr, *icon_pr;
static  void            canvas_event_proc(),
                        image_canvas_event_proc();
static u_char           red[256], green[256], blue[256];
static char             cmsname[BUFSIZ];
typedef enum direct {calm, n, ne, e, se, s, sw, w, nw};
    /* no motion, 0, 45, 90, 135, 180, 225, 270, 315 */
static struct pr_pos/* arrow head */
       clist0[7] = {{0,3},{1,2},{2,1},{3,0},{4,1},{5,2},{6,3}},
       clist1[7] = {{0,0},{1,0},{2,0},{3,0},{3,1},{3,2},{3,3}},
       clist2[7] = {{0,0},{1,1},{2,2},{3,3},{2,4},{1,5},{0,6}},
       clist3[7] = {{3,0},{3,1},{3,2},{3,3},{2,3},{1,3},{0,3}},
       clist4[7] = {{0,0},{1,1},{2,2},{3,3},{4,2},{5,1},{6,0}},
       clist5[7] = {{0,0},{0,1},{0,2},{0,3},{1,3},{2,3},{3,3}},
       clist6[7] = {{3,0},{2,1},{1,2},{0,3},{1,4},{2,5},{3,6}},
       clist7[7] = {{0,3},{0,2},{0,1},{0,0},{1,0},{2,0},{3,0}};


display_wind_field_only(clus)/* only show windfield, not overlay on image */
int  clus;
{
    Cint  name;

    frame = window_create(NULL, FRAME,
FRAME_LABEL,    filename,
WIN_WIDTH,      300,
WIN_HEIGHT,     300,
WIN_ERROR_MSG,  "Cannot create frame",
0);
    cursor = cursor_create(CURSOR_OP, PIX_SRC ^ PIX_DST,
        CURSOR_CROSSHAIR_LENGTH, 20,
        CURSOR_SHOW_CROSSHAIRS, TRUE,
        0);
    canvas = window_create(frame, CANVAS,
       WIN_CURSOR,                  cursor,
          CANVAS_WIDTH,             800,
          CANVAS_HEIGHT,            800,
          CANVAS_AUTO_SHRINK,       FALSE,
          WIN_EVENT_PROC,           canvas_event_proc,
          WIN_CONSUME_PICK_EVENTS,  LOC_MOVE, MS_RIGHT, 0,
          0);
    window_set(canvas,
              WIN_VERTICAL_SCROLLBAR, scrollbar_create(0),
              WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(0),
              0);
    pw = canvas_pixwin(canvas);

    open_pw_cgi();/* open window */
    open_cgi_canvas(canvas, &vpw, &name);
    wind_field(FALSE, 2.0, 0, 0, clus);/* draw the vector */
    close_cgi_pw(&vpw);
    close_pw_cgi();
    window_main_loop(frame);
}

display_wind_field_onimage(clus, pix, mem)
Pixrect *pix, *mem;
int  clus;
{
    register  int i, j;

    sprintf(title, "%s:%s", progname, ti_file);
    if (clus == nos_center || nos_center == 1)
      pr_rop(pix, 0, 0, pix->pr_size.x, pix->pr_size.y, PIX_SRC, mem, 0, 0);
    else
    {
        for (i = yso; i < yso+win_ysize; i++)
            for (j = 0; j < mem->pr_size.x; j++)
                if (j >= xso && j < xso+win_xsize)
                {
                    if (clusmap[i-yso][j-xso] == clus)
                        pr_put(pix, j, i, pr_get(mem, j, i));
                    else pr_put(pix, j, i, 0);
                }
                /*else pr_put(pix, j, i, 0);*/
    }
    pr = pix;

    image_window_create(FALSE, clus);
}

display_wind_field_onmap(clus)
int  clus;
{
    char  *ptr, filename[50];
```

```
    FILE *fp;

    strcpy(filename, tio_file);
    ptr = strrchr(filename, '/');
    strcpy(ptr, "/map");
    fp = fopen(filename, "r");
    demand(fp, cannot open display file);
    sprintf(title, "%s:%s", progname, ti_file);
    pr = pr_load(fp, &colormap);
    demand(pr, pixrect io error);
    fclose(fp);

    image_window_create(TRUE, clus);
}

image_window_create(displayonmap, clus)
char  displayonmap;
int  clus;
{
    Cint  name;
    int   scroll_thickness;

    set_windowsizes();
    scroll_thickness = defaults_get_integer("/Scrollbar/Thickness", 14, 0);

    frame = window_create(NULL, FRAME,
          FRAME_LABEL,    filename,
          WIN_HEIGHT,     winysize+scroll_thickness+7,
          WIN_WIDTH,      winxsize+scroll_thickness+10,
          WIN_ERROR_MSG,  "Cannot create frame",
          0);

    setup_colourmap();

    cursor = cursor_create(CURSOR_OP, PIX_SRC ^ PIX_DST,
         CURSOR_CROSSHAIR_LENGTH, 20,
         CURSOR_SHOW_CROSSHAIRS, TRUE,
         0);
    canvas = window_create(frame, CANVAS,
       WIN_CURSOR,                cursor,
          CANVAS_WIDTH,               pr->pr_size.y,
          CANVAS_HEIGHT,              pr->pr_size.x,
          CANVAS_AUTO_SHRINK,         FALSE,
          WIN_EVENT_PROC,             image_canvas_event_proc,
          WIN_CONSUME_PICK_EVENTS,    LOC_MOVE, MS_RIGHT, 0,
          0);
    window_set(canvas,
       WIN_VERTICAL_SCROLLBAR, scrollbar_create(0),
       WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(0),
       0);

    pw = canvas_pixwin(canvas);
    pw_rop(pw, 0, 0, pr->pr_size.x, pr->pr_size.y, PIX_SRC, pr, 0, 0);
    icon_pr = mem_create(64, 64, 8);
    paint_icon();

    open_pw_cgi();
    open_cgi_canvas(canvas, &vpw, &name);
    wind_field(displayonmap, 1.0, xs, ys, clus);
    close_cgi_pw(&vpw);
    close_pw_cgi();
    window_main_loop(frame);
}

static void
canvas_event_proc(canvas, event, arg)
Canvas   canvas;
Event    *event;
caddr_t  arg;
{
    char  buf[BUFSIZ];

    if (event_id(event) == LOC_MOVE)
    {
        sprintf(buf, "%s - %d, %d", title, event_x(event), event_y(event));
        window_set(frame, FRAME_LABEL, buf, 0);
    }
}

static void
image_canvas_event_proc(canvas, event, arg)
Canvas   canvas;
Event    *event;
caddr_t  arg;
{
    char  buf[BUFSIZ];
    double  lat1, long1, lat2, long2;

    if (event_id(event) == MS_LEFT)
    {
        sprintf(buf, "%s", title);
```

```
        window_set(frame, FRAME_LABEL, buf, 0);
    }
    if (event_id(event) == LOC_MOVE)
    {
        pixel_to_map((double)event_x(event), (double)event_y(event), 0., 0.,
                &lat1, &long1, &lat2, &long2);
        sprintf(buf, "x %d y %d lat %.2f long %.2f = %d", event_x(event),
                event_y(event), lat1, long1,
                pr_get(pr, event_x(event), event_y(event)));
        window_set(frame, FRAME_LABEL, buf, 0);
    }
}

paint_icon()
{
subsample(pr, icon_pr);

icon = icon_create(ICON_IMAGE, icon_pr, 0);
window_set(frame, FRAME_ICON, icon, 0);
}

subsample(in, out)
Pixrect *in, *out;
{
int     cvx, cvy, lcx, lcy, hcx, hcy;
register int i, j, i1, j1;
int     total;

cvx = (100 * in->pr_size.x) / out->pr_size.x;
cvy = (100 * in->pr_size.y) / out->pr_size.y;

for (j = 0; j < out->pr_size.y; j++)
            for (i = 0; i < out->pr_size.x; i++) {
                    hcx = i * cvx / 100;
                    hcy = j * cvy / 100;
                    lcx = (i - 1) * cvx / 100 + 1;
                    lcy = (j - 1) * cvy / 100 + 1;
                    lcx = (lcx < 0) ? 0 : (lcx > hcx) ? hcx : lcx;
                    lcy = (lcy < 0) ? 0 : (lcy > hcy) ? hcy : lcy;

                    total = 0;
                    for (j1 = lcy; j1 <= hcy; j1++)
                            for (i1 = lcx; i1 <= hcx; i1++)
                                    total += pr_get(in, i1, j1);
                    pr_put(out, i, j,
                           (total / ((hcx + 1 - lcx) * (hcy + 1 - lcy))));
            }
}

set_windowsizes()
{
    if (pr->pr_size.x > DEFAULT_WIN_XSIZE)
        winxsize = DEFAULT_WIN_XSIZE;
    else
        winxsize = MAX(pr->pr_size.x, MIN_WIN_XSIZE)+3;

    if (pr->pr_size.y > DEFAULT_WIN_YSIZE)
        winysize = DEFAULT_WIN_YSIZE;
    else
        winysize = MAX(pr->pr_size.y, MIN_WIN_YSIZE)+17;
}

setup_colourmap()
{
    register int i;

    red[0] = green[0] = blue[0] = 0;
    red[255] = green[255] = blue[255] = 255;
    fpw = (Pixwin *)window_get(frame, WIN_PIXWIN);

    if (colormap.type == RMT_NONE || colormap.length == 0)
    {
        sprintf(cmsname, "greyscale%d", pr->pr_depth);
        pw_setcmsname(fpw, cmsname);
        for (i = 1; i < 255; i++)
        {   red[i] = green[i] = blue[i] = i;  }
        pw_putcolormap(fpw, 0, 256, red, green, blue);
    }
    window_set(frame, FRAME_INHERIT_COLORS, TRUE, 0);
}

wind_field(displayonmap, scale, xs, ys, clus)
char  displayonmap;/* how to display */
int  xs, ys;/* top left offset of clustered window */
float  scale;/* the scale of spacing between vectors */
int  clus;
{
    int i, j,/* array index */
        x_off, y_off,/* offset of the upper left subwindow */
        side_off,/* offset of the ul corner of the ul subwin from origin */
        x1, y1, x2, y2,/* image coordinates of vector */
```

```
        length,/* vector length in pixel */
        numxwin, numywin;/* number of target window in a row and column */
    enum direct direction;
    float  slope, max_speed;

    numxwin = numywin = (wholexsize-2*win_offset)/target_size;
    cgipw_line_color(&vpw, 255);
    cgipw_marker_color(&vpw, 255);
    cgipw_marker_size_specification_mode(&vpw, ABSOLUTE);
    cgipw_marker_type(&vpw, ASTERISK);
    cgipw_marker_size(&vpw, 2.0);
    side_off = (search_size-target_size)/2;
    x_off = y_off = target_size;
    max_speed = 0.0;/* initialise */
    for (i = 0; i < numywin; i++)/* get max speed for scaling */
        for (j = 0; j < numxwin; j++)
            if (cmv[clus][i][j].speed > max_speed)
                max_speed = cmv[clus][i][j].speed;
    for (i = 0; i < numywin; i++)
        for (j = 0; j < numxwin; j++)
        {   /* show field in row order */
            if (cmv[clus][i][j].speed > 0.0)
               length = VEC_LENGTH*log10(cmv[clus][i][j].speed)/
                                        log10(max_speed);
            else   length = 0.0;
            y1 = ((y_off*i)+side_off+(target_size/2))*scale+ys;
            x1 = ((x_off*j)+side_off+(target_size/2))*scale+xs;
            slope = tan(cmv[clus][i][j].direction*PI/180.0);
            if (cmv[clus][i][j].direction==0.0 && cmv[clus][i][j].speed==0.0)
            {
               x2 = x1;
               y2 = y1;
               direction = calm;
            }
            else if (cmv[clus][i][j].direction == 0.0 ||
                     cmv[clus][i][j].direction == 180.0)
            {
               x2 = x1;
               if (cmv[clus][i][j].direction == 0.0)
               { y2 = y1+length; direction = n; }
               else
               { y2 = y1-length; direction = s; }
            }
            else if (cmv[clus][i][j].direction == 90.0 ||
                     cmv[clus][i][j].direction == 270.0)
            {
               y2 = y1;
               if (cmv[clus][i][j].direction == 90.0)
               { x2 = x1-length; direction = e; }
               else
               { x2 = x1+length; direction = w; }
            }
            else if (cmv[clus][i][j].direction < 90.0)
            {
                y2 = y1+(int)((float)length/sqrt(1.0+SQUARE(slope)));
                x2 = x1-(int)sqrt((float)(SQUARE(length)-SQUARE(y1-y2)));
                if (cmv[clus][i][j].direction < 5.0)       direction = n;
                else if (cmv[clus][i][j].direction < 85.0) direction = ne;
                else                                       direction = e;
            }
            else if (cmv[clus][i][j].direction > 90.0 &&
                     cmv[clus][i][j].direction < 180.0)
            {
               y2 = y1-(int)((float)length/sqrt(1.0+SQUARE(slope)));
               x2 = x1-(int)sqrt((float)(SQUARE(length)-SQUARE(y1-y2)));
               if (cmv[clus][i][j].direction < 95.0)       direction = e;
               else if (cmv[clus][i][j].direction < 175.0) direction = se;
               else                                        direction = s;
            }
            else if (cmv[clus][i][j].direction > 180.0 &&
                     cmv[clus][i][j].direction < 270.0)
            {
               y2 = y1-(int)((float)length/sqrt(1.0+SQUARE(slope)));
               x2 = x1+(int)sqrt((float)(SQUARE(length)-SQUARE(y1-y2)));
               if (cmv[clus][i][j].direction < 185.0)       direction = s;
               else if (cmv[clus][i][j].direction < 265.0) direction = sw;
               else                                        direction = w;
            }
            else
            {   /* 270 < direction < 360 */
               y2 = y1+(int)((float)length/sqrt(1.0+SQUARE(slope)));
               x2 = x1+(int)sqrt((float)(SQUARE(length)-SQUARE(y1-y2)));
               if (cmv[clus][i][j].direction < 275.0)       direction = w;
               else if (cmv[clus][i][j].direction < 355.0) direction = nw;
               else                                        direction = n;
            }
            if (displayonmap) draw_vector_onmap(x2, y2, x1, y1, direction);
            else              draw_vector(x2, y2, x1, y1, direction);
        }
}
```

```
draw_vector_onmap(x1, y1, x2, y2, direction)
int x1, y1, x2, y2; /* x2, y2 should always be the centre of subwin */
enum direct direction;
{/* draw vector on mercator projected map */
     int  x_off, y_off;
     double  p1, l1, p2, l2,
             lat1, long1, lat2, long2,
             mappix_size, longit_unit, init_mp;

     x_off = x1-x2; y_off = y1-y2;
     pixel_to_map((double)x1, (double)y1, (double)x2, (double)y2,
                  &lat1, &long1, &lat2, &long2);
     mappix_size = PIX_SIZE/(2.0*M_PI*EARTH_RADIUS/360.0);
     longit_unit = 1.0/(60.0*mappix_size);
     init_mp = 3437.747*log(tan(M_PI*(45.0+UPPER_LAT/2.0)/180.0));
     /*
     p1 = (long1-LEFT_LONG)/mappix_size;
     l1 = longit_unit*(init_mp-3437.747*log(tan(M_PI*(45+lat1/2.0)/180.0)));
     */
     p2 = (long2-LEFT_LONG)/mappix_size;
     l2 = longit_unit*(init_mp-3437.747*log(tan(M_PI*(45+lat2/2.0)/180.0)));
     /*x1 = anint(p1); y1 = anint(l1);*/
     x2 = anint(p2); y2 = anint(l2);
     x1 = x2+x_off; y1 = y2+y_off;

     draw_vector(x1, y1, x2, y2, direction);
}

draw_vector(x1, y1, x2, y2, direction)
enum direct direction;
int x1, y1, x2, y2; /* x2, y2 should always be the centre of subwin */
{
     Ccoor  head, vector[2];
     Ccoorlist  coorlist;

     /* vector[1] is the head, ie start from vector[0] */
     vector[0].x = x1;
     vector[0].y = y1;
     head.x = vector[1].x = x2;
     head.y = vector[1].y = y2;
     switch (direction)
     {
         case calm:
             coorlist.n = 1;
             coorlist.ptlist = &head;
             cgipw_polymarker(&vpw, &coorlist);
             break;
         case n:
             pw_polypoint(pw, x2-3, y2, 7, clist0, PIX_SRC|PIX_COLOR(1));
             break;
         case ne:
             pw_polypoint(pw, x2-3, y2, 7, clist1, PIX_SRC|PIX_COLOR(1));
             break;
         case e:
             pw_polypoint(pw, x2-3, y2-3, 7, clist2, PIX_SRC|PIX_COLOR(1));
             break;
         case se:
             pw_polypoint(pw, x2-3, y2-3, 7, clist3, PIX_SRC|PIX_COLOR(1));
             break;
         case s:
             pw_polypoint(pw, x2-3, y2-3, 7, clist4, PIX_SRC|PIX_COLOR(1));
             break;
         case sw:
             pw_polypoint(pw, x2, y2-3, 7, clist5, PIX_SRC|PIX_COLOR(1));
             break;
         case w:
             pw_polypoint(pw, x2, y2-3, 7, clist6, PIX_SRC|PIX_COLOR(1));
             break;
         case nw:
             pw_polypoint(pw, x2, y2, 7, clist7, PIX_SRC|PIX_COLOR(1));
             break;
     }
     coorlist.n = 2;
     coorlist.ptlist = vector;
     cgipw_polyline(&vpw, &coorlist);
}

/*****************************************************************
 * You are reading rad_temp.c                                    *
 * This is the function to convert METEOSAT 4 CHANNEL IR1 pixel *
 * count into temperature. For period Jan. to March 1991         *
 * Ref: Annexe to the METEOSAT4 calibration report.              *
 * K.S.LAU  13/5/91                                              *
 *****************************************************************/
#include  "cmv.h"

#define   SPACECOUNT        5.0
#define   COEFFICIENT       0.077/* average calibration coefficient */
#define   NUM_TEMPERATURE   102
#define   UP                1
#define   DOWN              2
```

```
rad_temp(clus)
int  clus;/* which cluster */
{
int  i, j, k,
     direction,/* search direction */
     numxwin, numywin;
float  temp_offset, rad, temp_rad[NUM_TEMPERATURE];

       numxwin = numywin = (wholexsize-2*win_offset)/target_size;
temp_offset = 200.0;/* temp[0] = 200K */

temp_rad[0] = 1.446;   temp_rad[1] = 1.492;   temp_rad[2] = 1.540;
temp_rad[3] = 1.588;   temp_rad[4] = 1.637;   temp_rad[5] = 1.687;
temp_rad[6] = 1.738;   temp_rad[7] = 1.791;   temp_rad[8] = 1.844;
temp_rad[9] = 1.899;   temp_rad[10] = 1.954;  temp_rad[11] = 2.011;
temp_rad[12] = 2.068;  temp_rad[13] = 2.127;  temp_rad[14] = 2.187;
temp_rad[15] = 2.248;  temp_rad[16] = 2.310;  temp_rad[17] = 2.373;
temp_rad[18] = 2.438;  temp_rad[19] = 2.503;  temp_rad[20] = 2.570;
temp_rad[21] = 2.638;  temp_rad[22] = 2.707;  temp_rad[23] = 2.777;
temp_rad[24] = 2.848;  temp_rad[25] = 2.921;  temp_rad[26] = 2.995;
temp_rad[27] = 3.070;  temp_rad[28] = 3.146;  temp_rad[29] = 3.223;
temp_rad[30] = 3.302;  temp_rad[31] = 3.381;  temp_rad[32] = 3.462;
temp_rad[33] = 3.545;  temp_rad[34] = 3.628;  temp_rad[35] = 3.713;
temp_rad[36] = 3.799;  temp_rad[37] = 3.886;  temp_rad[38] = 3.975;
temp_rad[39] = 4.064;  temp_rad[40] = 4.155;  temp_rad[41] = 4.248;
temp_rad[42] = 4.341;  temp_rad[43] = 4.436;  temp_rad[44] = 4.532;
temp_rad[45] = 4.630;  temp_rad[46] = 4.728;  temp_rad[47] = 4.828;
temp_rad[48] = 4.930;  temp_rad[49] = 5.032;  temp_rad[50] = 5.136;
temp_rad[51] = 5.242;  temp_rad[52] = 5.348;  temp_rad[53] = 5.456;
temp_rad[54] = 5.565;  temp_rad[55] = 5.676;  temp_rad[56] = 5.788;
temp_rad[57] = 5.901;  temp_rad[58] = 6.015;  temp_rad[59] = 6.131;
temp_rad[60] = 6.248;  temp_rad[61] = 6.367;  temp_rad[62] = 6.487;
temp_rad[63] = 6.608;  temp_rad[64] = 6.730;  temp_rad[65] = 6.854;
temp_rad[66] = 6.979;  temp_rad[67] = 7.106;  temp_rad[68] = 7.234;
temp_rad[69] = 7.363;  temp_rad[70] = 7.494;  temp_rad[71] = 7.626;
temp_rad[72] = 7.759;  temp_rad[73] = 7.894;  temp_rad[74] = 8.030;
temp_rad[75] = 8.167;  temp_rad[76] = 8.306;  temp_rad[77] = 8.446;
temp_rad[78] = 8.588;  temp_rad[79] = 8.731;  temp_rad[80] = 8.875;
temp_rad[81] = 9.020;  temp_rad[82] = 9.167;  temp_rad[83] = 9.316;
temp_rad[84] = 9.465;  temp_rad[85] = 9.616;  temp_rad[86] = 9.769;
temp_rad[87] = 9.922;  temp_rad[88] = 10.078; temp_rad[89] = 10.234;
temp_rad[90] = 10.392; temp_rad[91] = 10.551; temp_rad[92] = 10.712;
temp_rad[93] = 10.874; temp_rad[94] = 11.037; temp_rad[95] = 11.201;
temp_rad[96] = 11.367; temp_rad[97] = 11.535; temp_rad[98] = 11.703;
temp_rad[99] = 11.873; temp_rad[100] = 12.045; temp_rad[101] = 12.218;

for (i = 0; i < numywin; i++)
    for (j = 0; j < numxwin; j++)
    {
        if (cmv[clus][i][j].mean == 0.0) continue;
        rad = (255.0-cmv[clus][i][j].mean-SPACECOUNT)*COEFFICIENT;
        if (temp_rad[NUM_TEMPERATURE/2] > rad) direction = DOWN;
        else                                   direction = UP;
        switch (direction)
        {
            case UP:
                for (k = NUM_TEMPERATURE/2; k < NUM_TEMPERATURE; k++)
                    if (temp_rad[k] >= rad) break;
                break;
            case DOWN:
                for (k = NUM_TEMPERATURE/2; k >= 0; k--)
                    if (temp_rad[k] <= rad) break;
                break;
        }
        cmv[clus][i][j].temp = ((float)k+temp_offset)
                               -273.0;/* in celcius */
    }
}
```

Appendix K

Published Papers

Spatial-spectral clustering using recursive spanning trees

K.S. Lau
G. Wade

Indexing terms: Remote sensing, Pattern recognition, Meteorology

Abstract: The inherent contextual property of spanning trees is exploited in a nonparametric contextual clustering algorithm for multispectral satellite data. The linkage problem associated with shortest spanning trees is avoided by making extensive use of global information, and a two-stage algorithm (segmentation then clustering) is described, each stage being based upon recursive spanning trees and minimax variance techniques. A conditional entropy or 'segmentation loss' derived from mutual information is shown to provide a useful indication of the number of segments needed before clustering. The performance of the algorithm is compared with a single-pixel clustering algorithm and shows significant reduction in classification noise, both at class boundaries and within classes, while the spatial resolution of the single-pixel classifier is retained.

1 Introduction

Multispectral data from the Meteosat weather satellite is frequently analysed with the objective of extracting specific cloud classes. Information such as cloud height, type and distribution can then be deduced. Individual classes can be tracked from transmission to transmission using crosscorrelation or other techniques to estimate wind vectors at particular altitudes or pressure levels [1, 2]. A first step in the cloud classification is to cluster the data in multispectral space and this often done by fitting Gaussian PDFs to a multispectral histogram [2]. Usually the histogram is derived from the visible and thermal infra-red bands and only two or three distinct cloud layers or classes are identified [3].

Like the Bayes classifier, the histogram approach is a simple form of single-pixel classifier and takes no account of the context or spatial relationship of individual pixels. It is widely recognised that single-pixel classifiers are prone to classification noise, particulary (but not exclusively) at class boundaries [4–6]. It is also widely recognised that the use of contextual or spatial information can improve classification accuracy. For example, Kittler and Pairman [6] describe a parametric (i.e. PDF based) approach to contextual classification of weather satellite data and report significant improvement in cloud classification, particularly at cloud boundaries. Such improvement can significantly affect the subsequent shape analysis of the clouds which may be needed to recognise larger scale weather patterns.

This paper describes a nonsupervised approach to the problem of cloud classification, based upon graph-theoretic algorithms (although, since no training data is used we restrict discussion to clustering rather than true classification). The inherent contextual advantage of a graph-theoretic approach has been verified by Morris *et al.* [7] who claim very accurate boundary determination when segmenting monochrome images. Here we extend the concept to multispectral images and so use both spatial and multispectral information to minimise classification noise. Another advantage of the graph-theoretic approach is that it automatically yields hierarchical clustering, in the sense that the most significant clusters are generated first.

The clustering algorithm is based on nondirected graphs and is done in essentially two stages. The first or segmentation stage attempts to partition the image into homogeneous regions, and automatically terminates using information theory concepts. The second stage of the algorithm attempts to cluster the segments into a few classes.

2 Recursive spanning tree segmentation of multispectral images

A simple image to graph mapping maps a pixel gray level intensity to a node weight in graph G, as in Fig. 1*a*. The graph is a weighted graph if we assign edge (or link) weights, and the simplest assignment is on a local basis, i.e. the weight of an edge between nodes i and j could be simply

$$w_{ij} = |v_i - v_j| \tag{1}$$

where v_i and v_j are the node weights. The shortest spanning tree (SST) of this graph is a set of edges linking every node in G such that there are no loops (cycles) and such that the sum of the edge weights is a minimum. The SST for Fig. 1*a* can be obtained using Prim's or Kruskal's algorithm and is shown in Fig. 1*b*. Hierarchical segmentation into N segments can be achieved by simply cutting the SST at the $N-1$ most costly edges, and three segments are shown in Fig. 1*b*. Ideal segmentation generates regions which are homogeneous in some image sense and which are statistically independent of their neighbours. Unfortunately, as pointed out in Reference 7, SST segmentation has several significant shortcomings. For example, it is possible that two nodes differ markedly in weight but are connected by a series of edges each with a low weight. In this case, SST segmentation will tend to assign the two nodes to the same segment, causing significant segmentation error (the linkage problem).

Paper 8092I (E4), first received 23rd October 1989 and in revised form 18th February 1991

The authors are with the School of Electronic and Electrical Engineering, Faculty of Technology, Polytechnic South West, Drake Circus, Plymouth, Devon PL4 8AA, United Kingdom

The use of a recursive spanning tree (RST) provides a solution to this problem since edge weights are assigned on a global rather than a local basis [7]. Given a

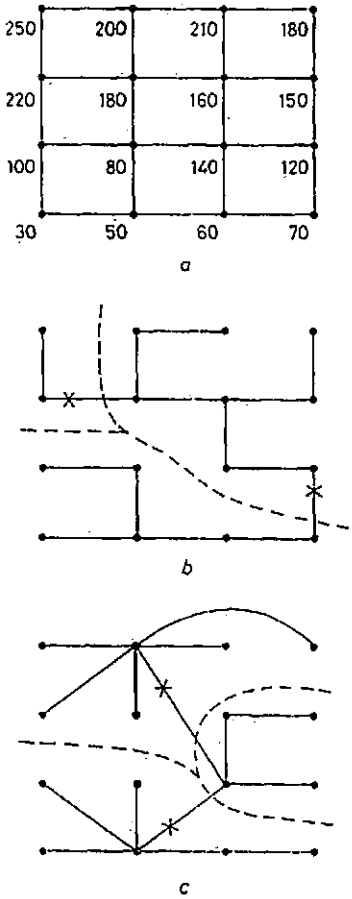


Fig. 1 *Four-connected graph, SST and RST*

a Four-connected graph generated by mapping pixel gray levels (0–255 scale) to node weights
b SST for *a* and hierarchical segmentation for $N = 3$
c RST for *a* and hierarchical segmentation for $N = 3$

weighted graph G, an RST needs only a small modification to Kruskal's algorithm and can be generated as follows:

(*a*) while there is more than one node in G:

(i) save the next least weighted edge, say edge e_{pq} between nodes p and q. (In general, node p will represent n_p original node weights and node q will represent n_q original node weights. The weight of n_p (n_q) will be the mean of the weights of all the nodes it represents.)

(ii) merge the two nodes p and q to make a new node r with weight equal to the mean of all the node weights in nodes p and q, i.e.

$$v_r = \frac{1}{n_p + n_q} \sum_{p,q} (v_p + v_q) \tag{2}$$

(iii) find the new edge weights (as in eqn. 1) for all edges which were connected to nodes p and q and which are now connected to node r (frequently this process leaves redundant edges, which are discarded).

(*b*) generate a spanning tree (link every node in G) with the saved edges. This is not an SST, but it provides a better representation of the relationship between pixels than the SST.

The significant point to note about the RST algorithm is that the edge weights w_{ij} are computed from an ever increasing neighbourhood as iteration proceeds, rather than from just local nodes. In fact, the final edge links the two remaining 'multinodes' and so its weight is a function of all node weights in G. Fig. 1*c* shows the RST algorithm applied to Fig. 1*a*; the final spanning tree is not an SST but the mean small-sample variance of the segments is lower than that for Fig. 1*b*, indicating more homogeneous regions on average.

In our study, each node in G is connected to its eight nearest neighbours, rather than to just four nearest neighbours as in Fig. 1*a*. Also, for multispectral (m-band) data, each node weight in G is a m-dimensional pattern or vector

$$\bar{v}_i = [v_{1i}, v_{2i}, \ldots, v_{mn}]^T \tag{3}$$

and, using the Euclidean distance metric, eqn. 1 becomes

$$w_{ij} = \sum_{k=1}^{m} |v_{ki} - v_{kj}|^2 = \|\bar{v}_i - \bar{v}_j\|_2^2 \tag{4}$$

2.1 RST-minimax segmentation

Given the RST we could perform hierarchical segmentation into N segments by cutting it at the $N-1$ most costly edges, as in Fig. 1*c*. Alternatively, the RST could be cut such that an objective function is minimised. For example, the first cut in the RST could generate two trees T_1, T_2 such that the maximum of two cost functions $c(T_1)$ and $c(T_2)$ associated with these trees is minimised, i.e. the initial partitioning of the RST could correspond to

$$\min [\max [c(T_1), c(T_2)]] \tag{5}$$

Since it is generally accepted that segmentation should account for the statistical properties of an image, the cost function $c(T_\alpha)$ could based upon an intraset distance measure [13]. For multispectral data we then have

$$c(T_\alpha) = \frac{1}{N_\alpha(N_\alpha - 1)} \sum_{i=1}^{N_\alpha} \sum_{j=1}^{N_\alpha} \sum_{k=1}^{m} (v_{ki} - v_{kj})^2 \qquad v_{ki}, v_{kj} \in T_\alpha \tag{6}$$

where N_α is the number of nodes in T_α. This can be reduced to

$$c(T_\alpha) = 2 \sum_{k=1}^{m} \sigma_{k\alpha}^2 \tag{7}$$

where $\sigma_{k\alpha}^2$ is the variance of tree T_α in band k. This statistical segmentation scheme can be extended to yield N regions using the following algorithm:

do $N-1$ times

(i) find tree T_{max} in G such that

$$c(T_{max}) = \max [c(T)] \quad \forall T \in G$$

(ii) cut T_{max} at edge $e_{i,j}$ linking nodes i, j which gives

$$\min [\max [c(T_i), c(T_j)]] \quad \forall e_{ij} \in T_{max}$$

This generates two trees, T_i and T_j from T_{max}

The advantage of the RST-minimax approach is that global (statistical) information is again taken into consideration, and it is intuitively reasonable to partition those trees with the largest intraset distance since they are less likely to be homogeneous. Similar minimax segmentation schemes have been investigated in References 7 and 8.

2.2 Automatic segmentation

Ideally, the segmentation process should automatically terminate once near-homogeneous regions have been found and popular ways of doing this are based upon

entropy measures [8–10]. This is reasonable since the zero-order entropy of a near-homogeneous segment, for example, tends to zero.

Information measure has been applied to feature selection in many pattern recognition problems [11]. The general information measure of a set of segments $Y = \{Y_1, Y_2, \ldots, Y_N\}$ can be defined as [10]

$$\begin{aligned} I(Y_1; Y_2; \ldots; Y_N) &= \sum_{i}^{Y_1} \sum_{j}^{Y_2} \cdots \sum_{k}^{Y_N} p(v_{1i}, v_{2j}, \ldots, v_{Nk}) \\ &\quad \times \log \frac{p(v_{1i}, v_{2j}, \ldots, v_{Nk})}{p(v_{1i})p(v_{2j}), \ldots, p(v_{Nk})} \\ &= \sum_{i}^{Y_1} \sum_{j}^{Y_2} \cdots \sum_{k}^{Y_N} p(v_{1i}, v_{2j}, \ldots, v_{Nk}) \\ &\quad \times \left[\log p(v_{1i}, v_{2j}, \ldots, v_{Nk}) + \log \frac{1}{p(v_{1i})} \right. \\ &\qquad \left. + \log \frac{1}{p(v_{2j})} + \cdots + \log \frac{1}{p(v_{Nk})} \right] \\ &= H(Y_1) + H(Y_2) + \cdots + H(Y_N) \\ &\quad - H(Y_1, Y_2, \ldots, Y_N) \end{aligned} \tag{8}$$

where v_{Nk} is the kth vector in segment N. Eqn. 8 can be interpreted as the total information conveyed by the segment set $\{Y\}$ with properties

(1) $I(Y) \geq 0$,

(2) $I(Y) = 0$ if and only if the vectors are independent, therefore

$$p(v_{1i}, v_{2j}, \ldots, v_{Nk}) = p(v_{1i})p(v_{2j}), \ldots, p(v_{Nk})$$

For an ideal segmented image all the terms of eqn. 8 are zero, and in practice we seek to minimise $I(Y)$. It should be noted that $H(Y_1, Y_2, \ldots, Y_N)$ is the joint entropy of the segments and is independent of the partition (constant), since it is a function of the probability distribution of the image itself. Therefore we could terminate segmentation when $H(Y) = \sum_i^N H(Y_i) < \theta$, where θ is a threshold and $H(Y)$ is the segment entropy. In practice, due to problems of computing the segment entropy, only the zero-order approximation would be used. This is reasonable, since, when the segmentation proceeds, interaction between segments will decrease and most higher-order terms can be neglected.

Daskalakis *et al.* [9] used a criterion similar to eqn. 8 to monitor the segmentation. They assume the segment entropy $H(Y)$ is composed of two components by

$$H(Y) = H_s + \sum_j p(j)H(j) \tag{9}$$

where H_s is the entropy due to the existence of segments, $p(j)$ is the probability of occurence of a particular segment Y_j and $H(j)$ is the entropy of segment Y_j. They assume the existence of segments obeys a Rayleigh PDF and pixels within a segment obey a normal PDF.

In this paper we use the concept of mutual information and model the segmentation process as a noisy communication channel (Fig. 2). For such a channel, the mutual information (the information common to both ends of the channel) is given by

$$I(X; Y) = H(X) - H(X \mid Y) \tag{10}$$

where $H(X)$ is the source entropy and $H(X \mid Y)$ is an indication of the loss of information during transmission. Alternatively, if $H(X \mid Y) = 0$, there is no ambiguity in the channel output. For the segmentation problem, $H(X)$ is the entropy of the multispectral data source and finite $H(X \mid Y)$ denotes an uncertainty in the segmentation or a 'segmentation loss'. It follows that if $H(X \mid Y) = 0$ we could consider the segmentation process complete in the sense that the source data has been segmented into homogeneous regions (see Appendix). For any real pictorial data the segments always have some residual variance and so, in general, we look for a significant reduction in the rate of change of $H(X \mid Y)$ with segmentation, rather than $H(X \mid Y) = 0$. Automatic segmentation can then be achieved by terminating segmentation when the rate of change falls below a nominal threshold.

It is shown in Appendix 10 that the segmentation loss is given by

$$H(X \mid Y) = -\sum_j \sum_i p(j \mid i)p(i) \log \left[\frac{p(j \mid i)p(i)}{\sum_i p(j \mid i)p(i)} \right] \tag{11}$$

where $p(i)$ is the probability of input vector $\bar{v}_i$, and is deduced from the multispectral histogram of the input data. Bearing in mind that, in general, a set of identical vectors will be distributed amongst spatially unconnected segments, and later these segments will be identified as belonging to the same class (through clustering), the conditional probability $p(j \mid i)$ can be computed as

$$p(j \mid i) = \frac{\#\{\bar{v}_i \in \text{segment } j\}}{\#\{\bar{v}_i \in G\}} \tag{12}$$

where $\#$ is the number of elements in the set.

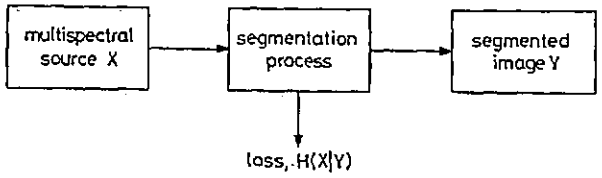


Fig. 2 *Segmentation modelled as an information flow process*

3 Spatial spectral clustering

Ideally, the segmentation process generates homogeneous regions in the image, each corresponding to a particular signature in m-dimensional spectral space. In general, some of these regions will have similar (or even identical) spectral signature, even though they may be spatially separated, and clustering is required to group these regions into a relatively few classes (an underlying assumption here is that each region or segment contains only one class). The clustering process was again based upon a spanning tree (RST) and so is a hierarchical form of unsupervised classification. The overall spatial-spectral clustering algorithm can be viewed as an essentially two-stage (segmentation then clustering) process, as shown in Fig. 3.

A simple way of mapping a segmented image onto a graph G is to assume near homogeneous regions so that the mean vector for segment i can be used as the weight of node i in G. The edge weights could then be computed as the Euclidean distance between node vectors, as in eqn. 4. In practice, this approach gave unsatisfactory clustering, and it was necessary to account for the residual variance within each segment. Interset distance might seem most appropriate here, but several of these measures have been found unsuitable due either to division by class variance [12] (which could give near infinite distance for sparse segments) or due to the difficulties of reducing the measure to a closed form [13].

The clustering algorithm was therefore based upon intraset distance (a mean square measure which is essentially a measure of class variance) and from eqn. 7 we can write this distance as [13]

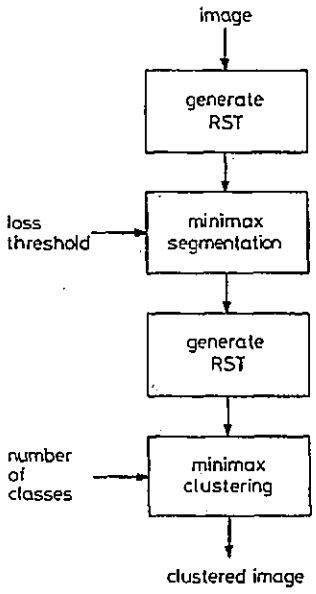


Fig. 3 *Contextual nonparametric clustering algorithm*

$$\bar{D}^2 = 2\sum_{k=1}^{m} \sigma_k^2 \tag{13}$$

where σ_k^2 is the variance of the segment in band k. Clearly, the measure for sparse segments now tends to zero rather than infinity.

An RST can now be generated by trying mergers of pairs of segments and looking for the minimum value of $\bar{D}^2$. The smallest distance then corresponds to an edge weight in the RST and the process is repeated to obtain all edges of the RST. This approach to tree generation is justified on the grounds that spectrally identical and homogeneous segments will have zero intraset distance when merged. It should be noted that the RST clustering method is remarkably similar to the average linkage method. The only difference between them is that, in the RST method, the distances between clusters are updated by recalculating the pairwise distances among the merging clusters, while in the average linkage method, the distances between clusters are updated by taking the average of all pairwise distance (no recalculation) from the original similarity matrix.

A formal algorithm for generating an RST and based on intraset distance is as follows:

(*a*) map the N segments to N nodes in G, and label all nodes

(*b*) calculate all possible link (edge) weights using intraset distance $\bar{D}^2$

while the number of nodes $N > 1$ {
(i) find the next least costly link weight w_{ij}
(ii) save the link
(iii) merge node (segment pair i, j as the union of sets N_i and N_j and decrement N
(iv) recalculate the link weights
(v) remove any links which form a cycle
}

(*c*) generate the spanning tree using the saved links.

Several clustering strategies are possible once an RST has been generated. Clustering into N classes can be achieved by simply cutting the tree at the $N - 1$ most costly edges. Alternatively, global information can again be incorporated by applying a minimax type algorithm, as in minimax segmentation, and this approach was found to yield the best results in practice.

4 Complexity of the algorithm

The complexity of segmentation depends on the incidence matrix of the image graph, and can be expressed as $O(V, L)$, where V is the number of vertices and L is the number of links. The amount of computation can be prohibitive with a moderate size image say, 200 × 200. Daskalaskis *et al.* [9] described a procedure which can reduce the overall complexity to $O(V^{3/2})$. Using this approach complexity varies as the power of vertices and so computational efficiency can be further improved by dividing the image into subimages and segmenting individual subimages. Alternatively, the spatial resolution of the image could first be reduced by averaging, and segmenting the reduced image.

In the clustering stage the RST approach involves a complexity of roughly $O(2N^2)$ [16], where N is the number of segments. This complexity assumes the similarity matrix is stored in its entirety in memory. Much of the clustering process involves a search for the most similar pair of segments and the subsequent updating of the similarity matrix. Usually the number of segments is at least an order less than the number of pixels, and a 256 × 256 image for example with 1000 segments can be clustered within a reasonable time. The real computation effort of course depends very much on the programming style and the algorithm chosen.

5 Comparison with single-pixel clustering

The performance of the contextual clustering algorithm in Fig. 3 was compared with the performance of a single-pixel (i.e. nonspatial) clustering algorithm based upon well tried techniques drawn from the ISOCLUS [12, 14] and ISODATA [13] algorithms. These are 'split and merge' routines and require input parameters such as maximum standard deviation of each class, minimum number of pixels/class, and a merging coefficient.

Essentially, the single-pixel clustering algorithm commences with a single cluster (as in the ISOCLUS algorithm) and splits clusters along the axis corresponding to the maximum standard deviation until the required number of classes have been generated.

6 Results

Fig. 4 shows typical behaviour of the segmentation loss $H(X|Y)$ during the segmentation of two-dimensional (visible and infrared) Meteosat data. Initially the loss decreases very rapidly as the major features in the data are identified, and, clearly, for 128 × 128 images, termination at 20 segments (say) appears premature. On the other hand, the graphs suggest that it may be unnecessary to generate 300 segments or more. Above approximately 150 segments the graphs exhibit near zero-gradient regions where increasing the number of segments is doing little to reduce the 'uncertainty' in the segmented image. In other words, it might be argued that near zero-gradient regions are optimal points at which to terminate segmentation, and in practice termination at around 200 segments gave good clustering for most cloud types.

In fact, as Fig. 5 shows, clustering is not a strong function of segment number (probably because spurious segments tend to be recombined at the clustering stage), and

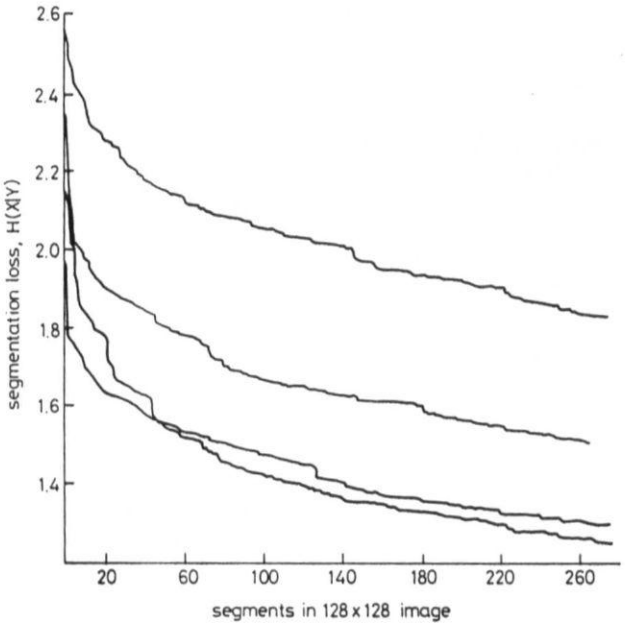


Fig. 4 *Segmentation loss for four 128 × 128 images*

so only a nominal threshold is required for automatic segmentation. However, there is some evidence to suggest that if an excessive number of segments are generated (say > 500) the spatial relationship between pixels tends to be destroyed and clustering becomes noisy, as in single-pixel clustering.

The improvement gained through the use of spatial information is illustrated in Fig. 6 for four different Meteosat images. Each of these figures shows raw visible and infrared images (digital data received on a Meteosat Primary Data User Station) and compares graph-theoretic clustering with single-pixel clustering for the same cloud class. Generally speaking, graph-theoretic clustering generates cleaner edges and more solid clusters than the ISOCLUS algorithm, while retaining the spatial resolution of the single-pixel classifier.

7 Conclusion

A nonparametric contextual clustering algorithm based upon spanning trees has been found to give improved clustering for multispectral Meteosat data when compared to single-pixel clustering. Classification noise is sig-

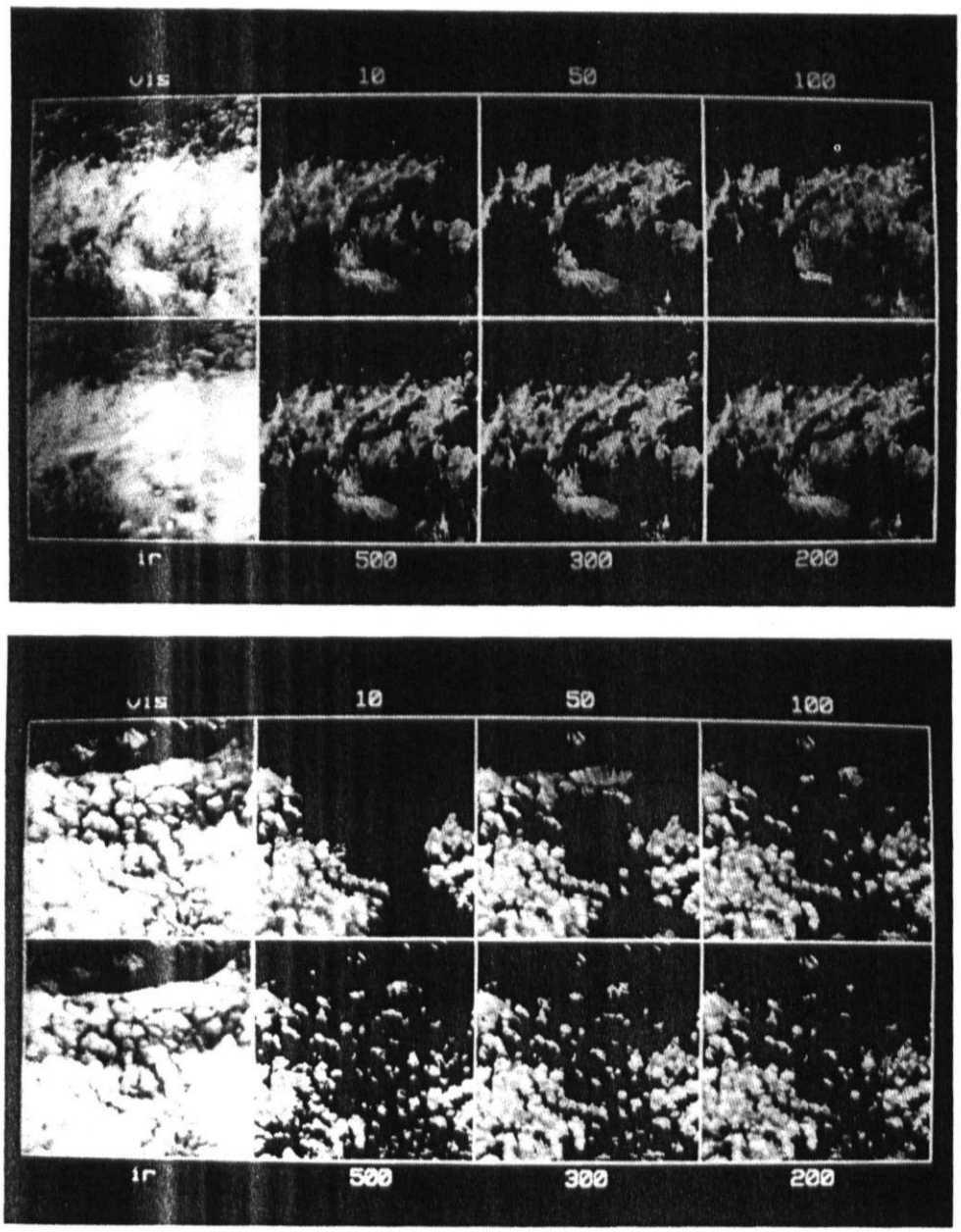


Fig. 5 *Sensitivity of clustering to the number of segments generated*

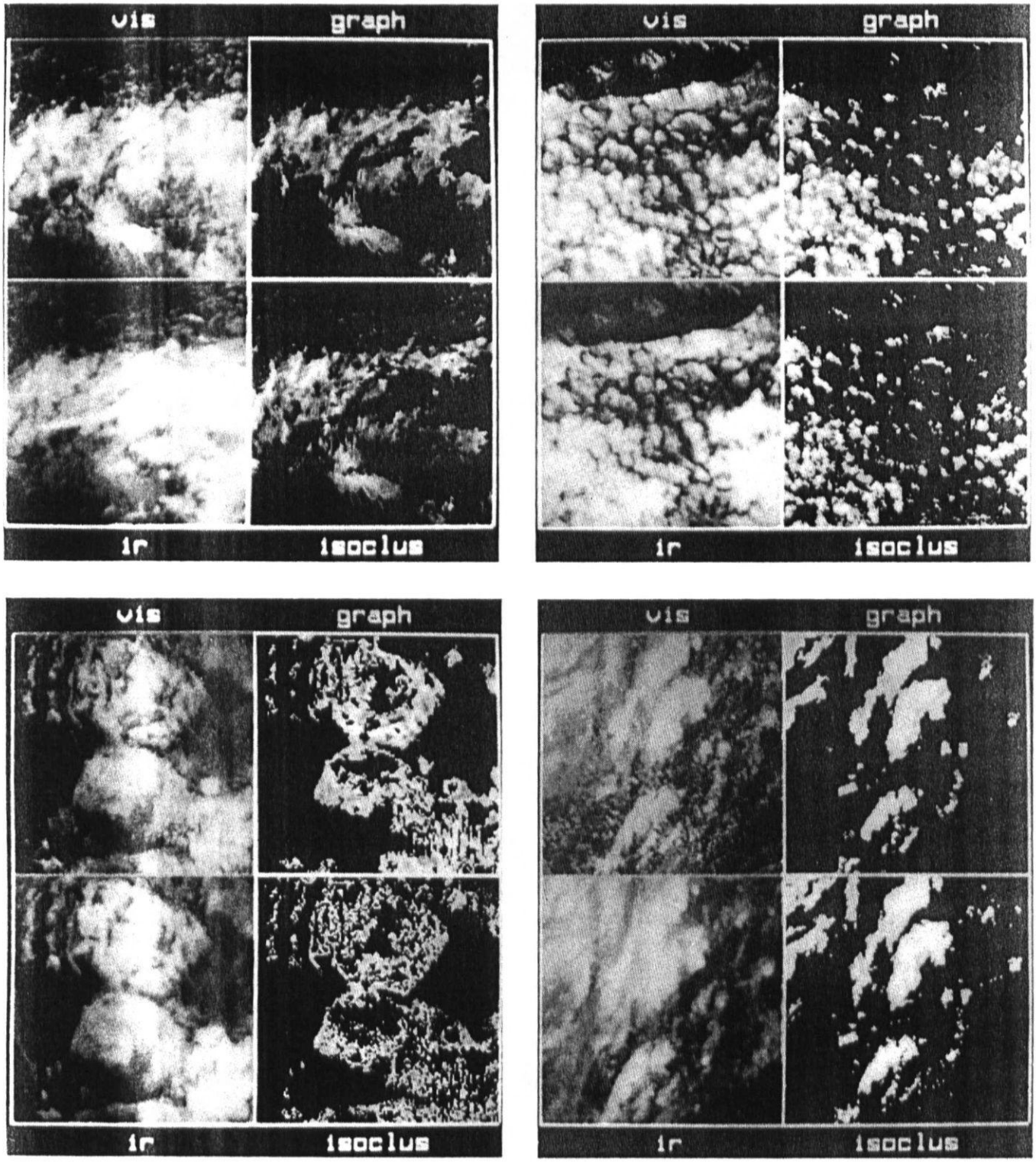


Fig. 6 *Comparison of graph-theoretic and single-pixel clustering for 128 × 128 images*

nificantly reduced in the sense that class boundaries are more well defined and more 'solid' regions are generated, while the spatial resolution of the single-pixel classifier is retained. A disadvantage of the graph-theoretic approach is that it requires at least an order of magnitude increase in CPU time compared to the single-pixel clustering algorithm, the major computational task being the generation of the RST for the multispectral image.

The mutual information (segmentation loss) concept has been found to provide a useful indication of the point at which segmentation should terminate. This enables the overall clustering algorithm to be semi-automatic in the sense that it only requires the specification of a (rather noncritical) threshold for the rate of change of segmentation loss and the required number of classes.

The best clustering results are obtained when global information is used to generate the spanning trees and when global information is used to partition the trees. The final clustering algorithm (Fig. 3) used a combination of RST and minimax variance algorithms in both the segmentation and clustering stages.

8 Acknowledgment

This work has been carried out as part of a programme supported by the UK Science & Engineering Research Council.

9 References

1 SCHMETZ, J., and NURET, M.: 'Automatic tracking of high-level clouds in Meteosat infrared images with a radiance windowing technique', *ESA Journal*, 1987, **11**, pp. 275–286
2 Meteosat System Guide — Volume 5: 'Meteorological products', ESOC, Darmstadt, Germany, 1980
3 WARNECKE, G., ZICK, C., CARUS, B., DORING, R., ERIKSSON, A., and VOELLGER, C.: 'Information extraction from meteorological satellite image sequences', *in* 'Remote sensing applications in meteorology and climatology' (D. Reidel Pub. Co., 1987), pp. 259–279
4 BRYANT, J.: 'On the clustering of multidimensional pictorial data', *Pattern Recognition*, 1979, **11**, pp. 115–125
5 KITTLER, J., and PAIRMAN, D.: 'Contextual classification of multispectral pixel data', *Image & Vision Comput.*, 1984, **2**, (1), pp. 13–29
6 KITTLER, J., and FÖGLEIN, J.: 'Contextual pattern recognition applied to cloud detection and identification', *IEEE Trans.*, 1985, **GE-23**, (6), pp. 855–863
7 MORRIS, O.J., LEE, M.J., and CONSTANTINIDES, A. G.: 'Graph theory for image analysis: an approach based on the shortest spanning tree', *IEE Proc. F, Commun., Radar & Signal Process.*, 1986, **133**, (2), pp. 146–152
8 DASKALAKIS, T.N., HEATON, A.G., and DASKALAKIS, C.N.: 'Minimax variance entropy-based image segmentation'. IERE Fifth Int. Conf. on Digital Processing of Signals in Communications, Loughborough University of Technology, 20th–23rd September 1988, pp. 291–297
9 DASKALAKIS, T.N., HEATON, A.G., and DASKALAKIS, C.N.: 'A graph-theoretic algorithm for unsupervised image segmentation', *in* 'Signal processing IV: theories and applications' (Elsevier Science Publishers, North-Holland, 1988), pp. 1621–1624
10 FARAG, R.F.H.: 'An information theoretic approach to image partitioning', *IEEE Trans.*, 1978, **SMC-8**, (11), pp. 829–833
11 BUTLER, G.A., and RITEA, H.B.: 'Estimation of mutual information in two-class pattern recognition', *IEEE Trans.*, 1974, **C-23**, pp. 410–420
12 TOWNSHEND, J.R., and JUSTICE, C.O.: 'Unsupervised classification of MSS Landsat data for mapping spatially complex vegetation', *Int. J. Remote Sensing*, 1980, **1**, pp. 105–120
13 TOU, J.T., and GONZALEZ, R.C.: 'Pattern recognition principles' (Addison-Wesley, 1974)
14 E.S.L. (Electromagnetics Systems Laboratory Inc.), IDIMS User's Guide. Technical Memorandum ESL-TM 705, Sunnyvale, California, 1976
15 HARALICK, R.M., and DINSTEIN, I.: 'A spatial clustering procedure for multi-image data', *IEEE Trans.*, 1975, **CAS-22**, (5), pp. 440–450
16 ANDERBERG, M.R.: 'Cluster analysis for applications' (Academic Press, Inc., 1973), Chap. 6

10 Appendix: Segmentation loss

The conditional entropy in eqn. 8 can be expressed as

$$H(X \mid Y) = H(X, Y) - H(Y) \tag{14}$$

where $H(Y)$ is the entropy of the segmented image. For simplicity we might assume that the N segments of the processed image are approximately statistically independent and write $H(Y)$ as the zero-order entropy

$$\begin{aligned} H(Y) &= -\sum_{j=1}^{N} p(j) \log p(j) \\ &= -\sum_j \sum_i p(i, j) \log p(j) \end{aligned} \tag{15}$$

Here $p(i, j)$ is the joint probability of pattern $\bar{v}_i$ and segment j. The zero-order entropy assumption becomes more realistic as segmentation proceeds, and individual segments become more homogeneous and statistically independent. The joint entropy in eqn. 14 can be written as

$$\begin{aligned} H(X, Y) &= -\sum_j \sum_i p(X = i, Y = j) \\ &\quad \times \log p(X = i, Y = j) \\ &= -\sum_j \sum_i p(i, j) \log p(i, j) \\ H(X \mid Y) &= -\Big[\sum_j \sum_i p(i, j) \log p(i, j) \\ &\quad - \sum_j \sum_i p(i, j) \log p(j)\Big] \\ &= -\sum_j \sum_i p(i, j) \log \left[\frac{p(i,j)}{p(j)}\right] \end{aligned} \tag{16}$$

But

$$p(j) = \sum_i p(i, j)$$

and

$$p(i, j) = p(j)p(i \mid j) = p(i)p(j \mid i) \text{ (Bayes' rule)}$$

Therefore

$$H(X \mid Y) = -\sum_j \sum_i p(j \mid i)(p(i) \log \left[\frac{p(j \mid i)p(i)}{\sum_i p(j \mid i)p(i)}\right] \tag{17}$$

Alternatively

$$H(X \mid Y) = -\sum_j \sum_i p(i \mid j)p(j) \log \left[\frac{p(i \mid j)p(j)}{\sum_i p(i \mid j)p(j)}\right] \tag{18}$$

Note that eqn. 16 can be expressed as

$$\begin{aligned} H(X \mid Y) &= -\sum_j p(j) \sum_i p(i \mid j) \log p(i \mid j) \\ &= \sum_j p(j)H(j) \end{aligned} \tag{19}$$

where $p(i \mid j)$ is the probability of pattern $\bar{v}_i$ occurring in segment j and $H(j)$ is a 'segment entropy'. If segment j is homogeneous, then $p(i \mid j) = 1$ at some input pattern $\bar{v}_i$ and $H(j) = 0$. In practice we look for a significant reduction in the rate of change of $H(X \mid Y)$ with segmentation, rather than $H(X \mid Y) = 0$.

Clustering applied to cloud wind determination

G. WADE, K. S. LAU

School of Electronic, Communication and Electrical Engineering, Polytechnic South West, Plymouth PL4 8AA, England

and N. L. H. WOOD

Institute of Marine Studies, Polytechnic South West, Plymouth PL4 8AA, England

Abstract. Reliable cloud motion wind generation from Meteosat images requires good target selection. This is usually done by examining the infrared channel and selecting target windows which have a temperature variation between an upper limit and lower limit, i.e., windows containing essentially a single cloud layer. In this paper we apply an optimised multi-spectral clustering algorithm in an attempt to extract the principal cloud targets prior to target tracking. Experimental results show an increase in the number of trackable targets compared to conventional techniques based on raw data. The paper also examines the optimal target size and compares the performance of several target tracking techniques.

1. Introduction

Cloud motion winds are usually derived by tracking targets in sequences of images captured from a geostationary satellite. Essentially, motion vectors are computed by searching for a $J \times K$ picture element (pixel) target in an $M \times N$ search window of the next image of the sequence (Leese and Novak 1971).

Since the sensed image is an overview from the cloud top, it is possible that there are multiple layers of cloud within the target window. Due to the fact that different cloud layers may move with different direction and speed, it follows that accurate estimation of wind vectors can only be achieved by selecting targets whose temperaure can be accurately estimated (Hubert 1971), and for which evaporation or formation is minimal. In other words, some mechanism for identifying essentially single layer cloud must be applied. This paper attempts to do this via multi-spectral clustering on the assumption that the resulting natural data patterns (clusters) extract the principal cloud layers. Experimental results (§ 5) show that multiple cloud types present in the window often result in serious tracking errors i.e., failure to track the same cloud feature.

Too large a target window will generate an average motion of all objects within it, whilst too small a target can lead to poor tracking and increased error in wind vector estimation. The paper therefore also examines the optimal target size for Meteosat images, as well as several image matching algorithms for target tracking.

0143–1161/92 $3.00

2. Development of clustering scheme

2.1. *Clustering*

Clustering can be defined as the automatic identification of natural groupings, or structures, within multi-spectral data. The process is to partition the data set into subsets using some distance (or similarity) measure, such that all samples in a subset are similar to each other.

Clustering has been shown to be a very effective tool for segmentation of multi-spectral cloud images. It has been applied to cloud images by Parikh and Rosenfled (1978), Desbois *et al.* (1982), Seddon and Hunt (1985) and Kittler and Pairman (1985). They all used a type of iterative clustering algorithm to partition multi-spectral cloud images and their results indicate that clustering is capable of partitioning cloud images into different cloud types. It is recognised, however, that it is not always possible to relate every cluster to a single information category. For example, one cluster may be thick cloud, while the edge of the thick cloud is usually assigned to another cluster. On the other hand, given the above experimental evidence, it is reasonable to conjecture that the clusters partition principal cloud types.

A clustering scheme has been developed to investigate the concept of individual object tracking and it is designed to be both efficient and objective. It is based on the widely used ISODATA clustering algorithm (Ball and Hall, 1967). This algorithm starts by selecting a set of pixels as cluster centres, and then assigns every pixel to the nearest centre. Next, the centres are updated using the mean value of the pixels assigned to that cluster and the process is repeated with the new centres. The algorithm terminates when there is no significant change in the new centres. Splitting and merging of clusters can be introduced to speed up convergence and to allow the number of clusters to vary such that the subset generated is closest to the inherent data structure.

This basic type of iterative mode separation algorithm (a form of dynamic clustering algorithm), has several major limitations:

1. The selection of starting centres is subjective and inefficient and the influence on the final partition can be very significant.
2. The use of the cluster mean for the cluster centre is not a good model for cloud images, since it does not allow clusters to have different population and variance.

2.2. *Selection of starting points*

The ISODATA algorithm requires initial cluster centres to be specified before iteration commences, and usually a sensible estimate of these centres is made in order to reduce convergence time (Desbois *et al.* 1982, Seddon and Hunt 1986, Pairman and Kittler 1986). For automatic clustering, manual selection of these points must be avoided and we seek some other starting procedure. Anderberg (1973) describes a number of starting techniques, most of them being based upon a random selection of initial centres such that they span the data set evenly. However, in our experience, this random selection usually fails to separate clusters with small variance.

In this article we generate the starting points using a non-parametric clustering algorithm (Narenda and Goldberg 1977). This algorithm attempts to partition the multi-dimensional histogram into unimodal regions, and the statistics of each region

are used as starting points for the ISODATA algorithm. Histogram clustering relies on a high pixel to vector ratio in order to obtain a good estimate of the probability density in the feature space. This is the case for Landsat images (for which the algorithm was designed) and is often the case for the land and sea classes in Meteosat images. However, the algorithm fails to separate overlapped clusters with no definite boundary (most middle level cloud), although the cluster statistics still provide realistic starting points for subsequent clustering.

We shall now briefly describe the histogram clustering alogrithm, as applied to Meteosat data. First, the 8-bit resolution of the visible and infrared images needs to be reduced to minimise the number of trivial clusters. We have found that a compression ratio of 4 (giving 64 grey levels/band) is required for most Meteosat images. Histogram compression (as suggested by Wharton 1983) therefore provides a first degree of histogram smoothing as well as data reduction. The histogram is stored in a hashing table (Narenda and Goldberg 1977) which is essentially a look-up table, the main purpose being to minimise memory usage.

It is also necessary to smooth the compressed histogram, again to minimise the number of trivial clusters. Smoothing is simply achieved by averaging the histogram values over the neighbourhood of a histogram cell and replacing the cell count with the mean count. A smoothing threshold is set such that cells with densities above the threshold will not be smoothed. The compressed and smoothed histogram is now clustered using the valley seeking algorithm described by Koontz *et al.* (1976) This algorithm groups the histogram cells by constructing a directed tree as shown in figure 1. A directed link is placed between each vector and the immediate neighbour

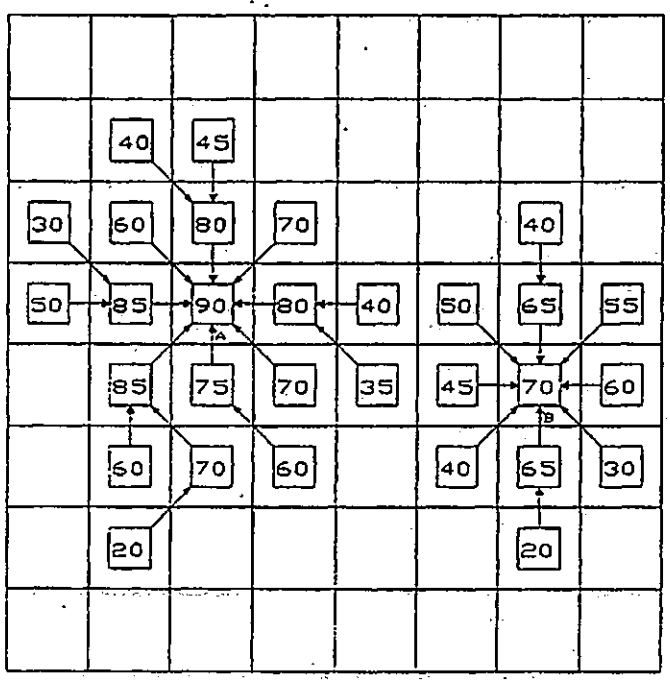


Figure 1. A two-dimensional illustration of the histogram clustering scheme. Each square denotes a histogram cell and its associated frequency count. The arrows link cells to neighbours with maximum positive gradient in frequency count. Cells A and B are local maxima (roots).

which is in the direction of the maximum positive density gradient. The gradient is defined as

$$g_{ij}=\frac{f_j-f_i}{d_{ij}} \tag{1}$$

where f_j is the frequency of cell j, and d_{ij} is the squared Euclidean distance between histogram cells i and j.

A vector is called a root and lies at a local maximum or mode of the histogram if all neighbours have density values less than itself. Ties are resolved arbitrarily. A cluster will then be determined by tracing the directed tree starting from the root.

2.3. *Selection of clustering model*

For C clusters, a dynamic clustering algorithm aims to minimise some global criterion function,

$$J(\Omega)=\sum_{i=1}^{C}\sum_{j=1}^{N_i}\Delta(x_j, K_i) \tag{2}$$

over all possible partitions Ω of the image. Here $\Delta(x_j, K_i)$ is some positive valued distance (similarity) measure between pixel vector x_j and the kernel K_i for cluster ω_i and N_i is the total number of samples in the data set. The original ISODATA algorithm (Ball and Hall 1967) used a simple cluster mean model i.e. it assigned x_j to cluster ω_i if,

$$\Delta(x_j, K_i)=\min \Delta(x_j, K_k) \quad k=1,\ldots C \tag{3}$$

where K_i is the mean of cluster ω_i.

The cluster mean model uses squared Euclidean distance to measure the distance of each sample to each centre, which implies that every cluster has an identical normal distribution and an identity covariance matrix. This assumption is clearly not true for cloud images, and it is better to model the clusters for cloud images using a general multi-variate normal distribution. In this case,

$$K_i(x_j,\forall j)=\frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}}\exp\left\{-\frac{1}{2}(x_j-m_i)^T\Sigma_i^{-1}(x_j-m_i)\right\} \tag{4}$$

where Σ_i is the covariance matrix, m_i is the mean of cluster ω_i, and D is the dimension of the feature space.

In order to allow clusters with different distribution parameters (different covariance matrices and populations) it is necessary to derive an expression for the distance of each sample to every cluster kernel. Using Baysian criteria, Kittler and Pairman (1986) give the following metric for the model of (4):

$$\Delta(x_j, K_i)=(x_j-m_i)^T\Sigma_i^{-1}(x_j-m_i)+\log|\Sigma_i|-2\log\frac{N_i}{N} \tag{5}$$

where $\frac{N_i}{N}$ is the sample estimation of the cluster population. Various simplifications of (5) are possible. For example, if we assume all classes equiprobable, i.e., $p(\omega_i)=\frac{1}{C}$, $i=1,\ldots C$ (5) reduces to

$$\Delta(x_j, K_i)=(x_j-m_i)^T\Sigma_i^{-1}(x_j-m_i)+\log|\Sigma_i| \quad (6)$$

If we further assume all covariance matrices are equal, (6) reduces to the Mahalanobis distance

$$\Delta(x_j, K_j)=(x_j-m_i)^T\Sigma_i^{-1}(x_j-m_i) \quad (7)$$

Finally, if all covariance matrices are equal to the identity matrix, (7) reduces to

$$\Delta(x_j, K_i)=(x_j-m_i)^T(x_j-m_i) \quad (8)$$

which is the squared Euclidean distance metric used in the original ISODATA algorithm.

Clustering algorithms using (6), (7), (8) tends to generate clusters with minimum within-cluster variance, and so the cluster sizes tend to be equal. By allowing different cluster sizes and variances (5) generally gives better results (Pairman and Kittler 1986).

2.4. *The hybrid clustering algorithm*

The hybrid algorithm in figure 2 is the union of the histogram clustering algorithm and the optimised ISODATA algorithm. The first step uses the histogram clustering algorithm to obtain an initial partition. This gives the user a general idea of the number of clusters and their tightness, and if the partition is believed to be suboptimal, it is supplied to the ISODATA algorithm as starting points. The second step is to optimise the partition based on the global objective function in (5). We have included split and merge routines in the ISODATA algorithm to improve convergence—a particularly important point if some cloud clusters (usually middle level cloud) generated by the histogram algorithm have high standard deviation.

Although ISODATA is regarded as an efficient algorithm, it can still be computationally intensive when applied to large, multi-spectral images. However, rather than follow the usual approach of clustering individual pixels, significant improvement can be achieved by clustering individual vectors in the histogram. For example, using visible and infrared Meteosat images yields a pixel to vector ratio in the range 7–15 for a full resolution two-dimensional histogram, indicating a significant computational saving. This more efficient approach is adopted in figure 2.

3. Wind vector determination

3.1. *Target selection*

Many operational wind generation schemes (either manual or automatic) have been derived since the first meteorological satellite was launched in the late 1960s. Hubert (1979) has an excellent review of operational wind systems. However, due to the extremely complicated nature of the atmosphere, not all cloud motion can be representative of the surrounding wind. Errors arise due to gravity waves, lee waves, banner clouds and other cloud development, as well as from multiple layer clouds within a target window. Therefore all wind system outputs usually have to be edited by a trained meteorologist.

Manual schemes generate wind by tracking targets selected by a trained operator. An image sequence is animated on a video screen and the operator selects the cloud tracer that persists over a long period. The cloud motion is then determined either by cross-correlation or by measuring the displacement by viewing

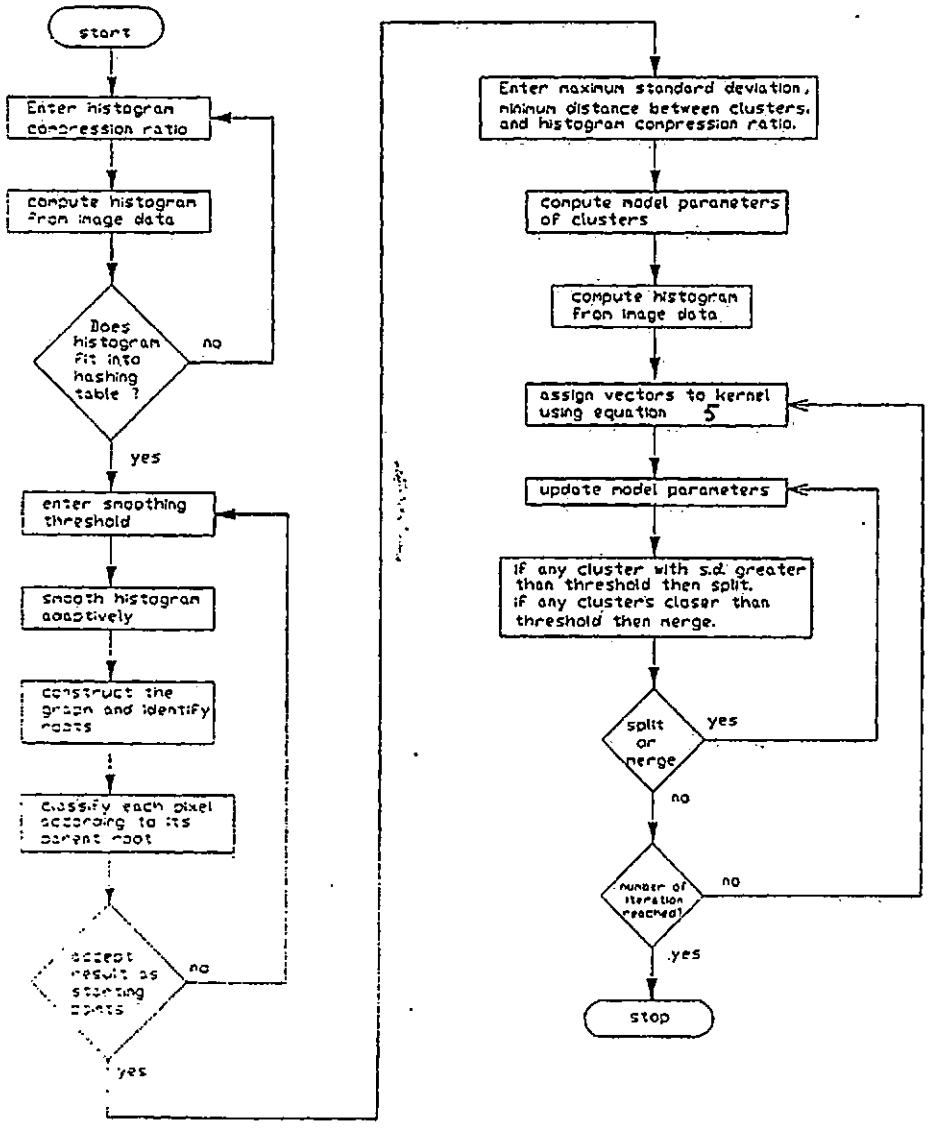


Figure 2. The hybrid clustering scheme.

the sequence. Manual schemes are inefficient (only a small number of vectors are generated) and so they tend to be used only when automatic schemes fail

Automatic schemes generate wind vectors by dividing the area of interest into overlapped or nonoverlapped target windows. the typical window size being 8 by 8, 16 by 16, or 32 by 32 pixels. A window is then analysed to determine whether it contains a suitable cloud target for tracking. A simple check is to make sure that the infrared window has a temperature variation less than a threshold (indicating the absence of multi-layer cloud). A sophisticated approach has been developed by the European Space Operations Centre (ESOC) and is applied to Meteosat images operationally. A multi-spectral histrogram is analysed for every 32 by 32 target window, and all objects are classified as sea, or as various types of land, or cloud. The original scheme was described by Bowen *et al.* (1979).

Since then more development has been done to improve tracking of target windows containing multiple cloud types. Schmetz and Nuret (1987) used a radiance

slicing technique for high level clouds. Hoffman (1990) used a filtering technique to preprocess the target window before tracking. The filtering process extracts pixels belonging to the highest cloud layer and smoothes contaminated pixels and the background. Schemetz and Holmlund (1990) have shown that using the radiance slicing and filtering technique, forecast guided tracking and better height assignment of semi-transparent clouds, the error between radiosonde wind and cloud motion wind can be reduced. It is evident from these recent investigations, that the introduction of cloud type separation before tracking can improve cloud wind quality.

3.2. *Target tracking*

Tracking is usually based on cross-correlation (Leese and Novak 1971). Essentially, an array of data (the target window) is selected from an image and correlated element by element with selected pixels (the search window) of a second image. see figure 3. The displacement is determined by the lag position which produces the maximum correlation. The cross-correlation function is defined as

$$R(u, v)=\frac{cov(u, v)}{\sigma_T \sigma_S(u, v)} \tag{9}$$

where σ_T is the standard deviation of the target window, σ_S is the standard deviation of the search window at lag position u, v, and $cov(u, v)$ is the covariance between the target window and the search window at lag position u, v. Specifically,

$$\bar{G}=\frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K} g(j, k) \tag{10}$$

$$\sigma_T=\left\{\frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K}(g_T(j, k)-\bar{G}_T)^2\right\}^{1/2} \tag{11}$$

$$\sigma_S(u, v)=\left\{\frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K}(g_S(j-u, k-v)-\bar{G}_S(j-u, k-v))^2\right\}^{1/2} \tag{12}$$

$$cov(u, v)=\frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K}[g_T(j, k)-\bar{G}_T][g_S(j-u, k-v)-\bar{G}_S] \tag{13}$$

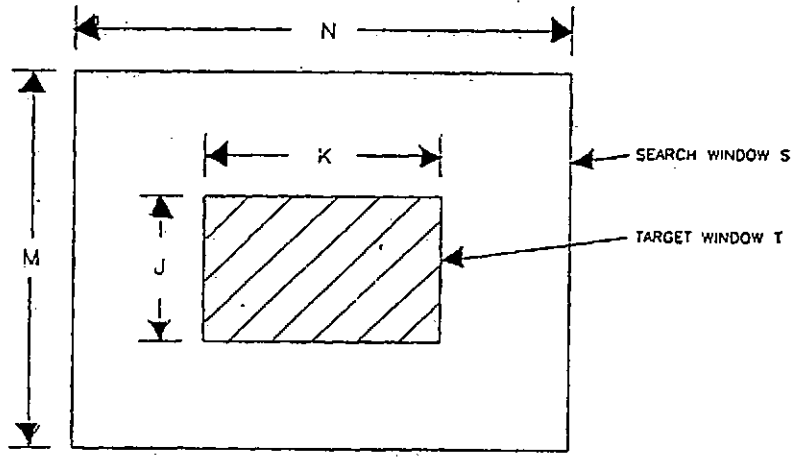


Figure 3. Definition of target and search window.

where $\bar{G}$ is the mean grey level of the window.

The lag position is given by $N-K+1$, $M-J+1$ in the horizontal and vertical directions, respectively. A disadvantage of cross-correlation is its efficiency; it is computationally demanding especially for large lag positions, although computation time can be reduced by using the Fast Fourier transform.

Another matching technique is called the Sequential Similarity Detection Algorithm (SSDA) (Barnea and Silverman 1972). This technique does not require normalisation as in cross-correlation. It is defined as the mean absolute error of the target and search window at every lag position.

$$S(u, v)=\frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K}|g_T(j, k)-g_S(j-u, k-v)| \tag{14}$$

Wilson (1984) have computed wind vectors using this simple technique, and the displacement is determined by the lag position with minimum error.

A third technique is the two-dimensional log search (2-d search) (Jain 1987). The 2-d search computes a few values of the surface coefficient and uses these to search for the local minima (or maxima). This approach reduces computation to a minimum, but the success of the method depends on the smoothness of the surface.

Finally, to note that the images must be accurately registered in order to provide unbiased wind estimation. For example, a misregistration of one infrared pixel using Meteosat imagery can produce a error of $2{\cdot}8\,\mathrm{ms}^{-1}$ at the subsatellite point. Currently real time rectification (Bos *et al.* 1990) is applied operationally to Meteosat images, and the accuracy is good enough for cloud motion tracking without further image registration.

3.3. *Strategy to reject erroneous vectors*

The correlation surface does not always provide a clear peak corresponding to the displacement and a typical correlation surface can have one of the following characteristics.

1. More than one obvious peak: the pattern in the target window is similar to more than one pattern in the search window.
2. A well-defined peak cannot be found: the target window probably contains a large area without any features.
3. Generally speaking, a large window always produces a clearer peak than a small window at the same location.

The uncertaintly in the real displacement is usually reduced by,

1. Using three images to compute two vectors and rejecting all unsymmetrical vectors (Bowen *et al.* 1979).
2. Using a hiarchical search technique: a large target window generates a first guess to guide the search and then the window size is reduced (Hubert 1979).
3. Use the most recent atmosphere analysis to provide an estimate of the lag position (Bristor 1975).

These three methods can be combined to form the most effective strategy.

In order to improve efficiency, a search strategy can be adopted. ESOC's (Bowen *et al.* 1979) search strategy finds the direction of steepest ascent from the original position and converges to the maximum correlation in this direction. The success of this search strategy depends on the smoothness of the surface.

3.4. *Height assignment*

To be meaningful, each wind vector must be assigned to a height (or pressure) level. Unfortunately, some low level winds correlate best to the cloud base, while some correlate best to the cloud top, and so cloud top temperature frequently does not provide adequate vector height information. Also, cloud top temperature cannot be estimated accurately due to the low emissivity of cloud. ESOC (Bowen and Sanders 1984) use two infrared channels to correct the emissivity of semi-transparent cloud and the mean valued of the corrected infrared pixels in the target window is used to calculate the cloud top temperature.

4. The automatic scheme

Figure 4 shows an automatic scheme for wind vector computation. It uses either raw or clustered images, and any one of three target tracking techniques discussed in §3.2 (giving six experimental approaches). Target window sizes can be 4 by 4, 8 by 8, 16 by 16, 24 by 24, or 32 by 32 (yielding one vector per target).

The basic approach is to use sequences of three (visible and infrared) images spanning a total of one hour. The second infrared image is divided into nonoverlapped areas which define possible target windows, and these are then checked for suitable targets. If the window has an infrared variance < 100 (i.e., black body scene temperature variations are usually $< 10°$ C), then it is tracked in the first and third image of the sequence.

When using the clustered approach, only the second image is clustered, and only pixels $g_T(j,k) \in \omega_i$ will be used in (9) and (14) when cluster i is used for tracking, other pixels in the target window being ignored. This results in a target window which is not necessarily filled with pixels, and so a further check must be made. At present, a window less than 30 per cent filled is rejected for a target; the threshold is not critical, but if it is too small spurious winds may be generated. Clearly, target windows containing more than one cluster may have more than one wind vector.

Targets are located in the centre of search windows (figure 3), and 28 lag positions are allowed in the horizontal and vertical direction respectively. This provides for a maximum wind speed of at least 75·5 knots. Two vectors are obtained by tracking the first and third images in the sequence. Vectors corresponding to minima (SSDA) or maxima (correlation) falling on the borders of the search area are rejected. Vectors are then checked for symmetry; if the speed difference is > 50 per cent of the smaller vector, or the direction differs by $> 30°$, the wind vector is again rejected.

5. Experimental results

Wind vectors have been computed for three image sequences taken on 5, 8 and 11 March 1991, see figure 5. The first image in the sequence was received at 11.30 GMT and the following images at 30 minute intervals. Cloud motion winds were computed in 256 pixel by 256 pixel areas (outlined), the coordinates of the corners being (moving from the top right hand corner in a clockwise direction) 61° N 5·5° E, 40° N 3·4° E, 40° N 10·7° W, 61° N 17·1° W. Tracking was performed on non-geometrically corrected images, and displacement correction was applied after tracking.

5.1. *Clustering*

Each set of images highlights three cloud signatures associated with depressions in three stages of development. The images for the 5 March show frontal cloud

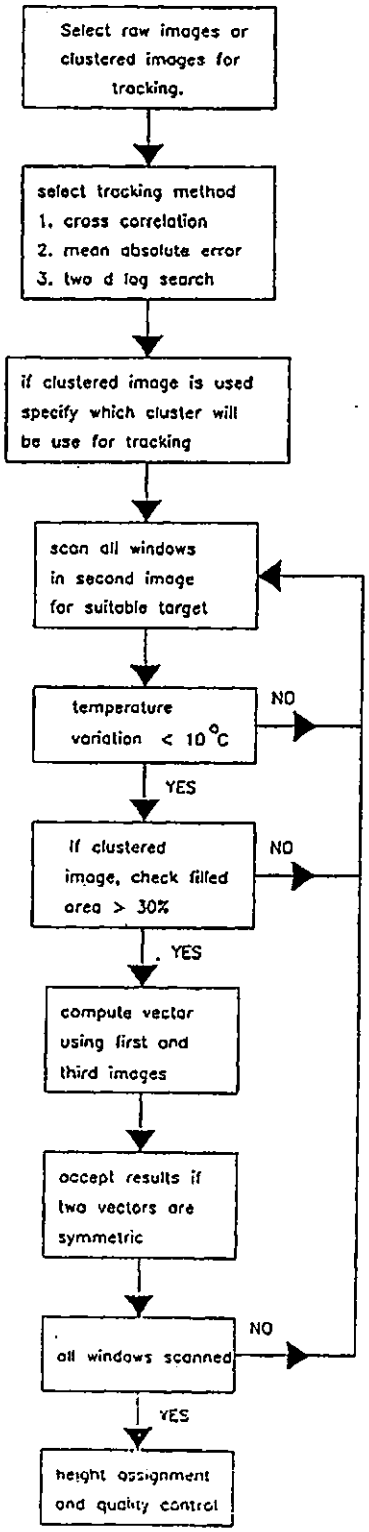
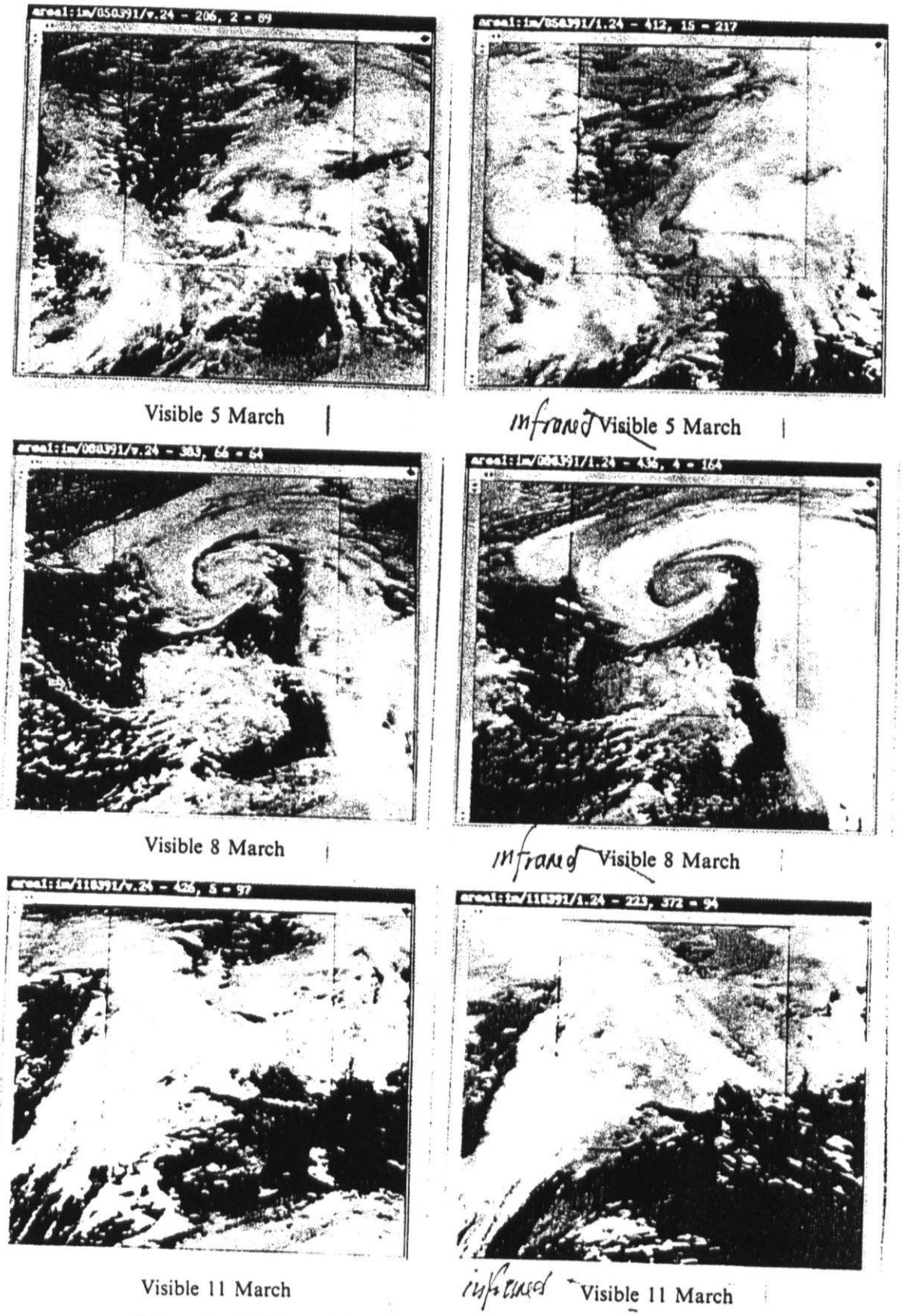


Figure 4. The automatic scheme used for experiment.



Visible 5 March | Infrared Visible 5 March

Visible 8 March | Infrared Visible 8 March

Visible 11 March | Infrared Visible 11 March

Figure 5. Visible and infrared images for 5, 8, and 11 March 1991.

associated with an occluding depression centred to the north of Scotland. The associated cold front stretches from the North Sea across southern England, with relatively cloud-free air behind. The deep frontal cloud is embedded in a predominantly southwesterly flow. On 8 March a well-occluded low pressure is centred over Cornwall and the Meteosat images show a classic spiral cloud pattern. Cloud motion vectors confirm the converging airflow associated with such a mature system. A major low pressure comlex is situated in mid-Atlantic on 11 March and there is a southwesterly turning northwesterly flow in the upper troposphere. Ahead of the warm and occluded front it is possible to identify the southeasterly winds associated with the polar trough. The clouds here are medium level and therefore are moving under the polar front cloud.

The hybrid clustering scheme was applied to the second image of the three sequences. Seven clusters were found in the 5 and 8 March images, and eight clusters were found in the 11 March image. Figure 6 shows the cluster map of the three images.

The cluster images compare favourably with the cloud types identified on the original pictures in the infrared and visible wavelengths. The important features are highlighted in terms of the following classes:

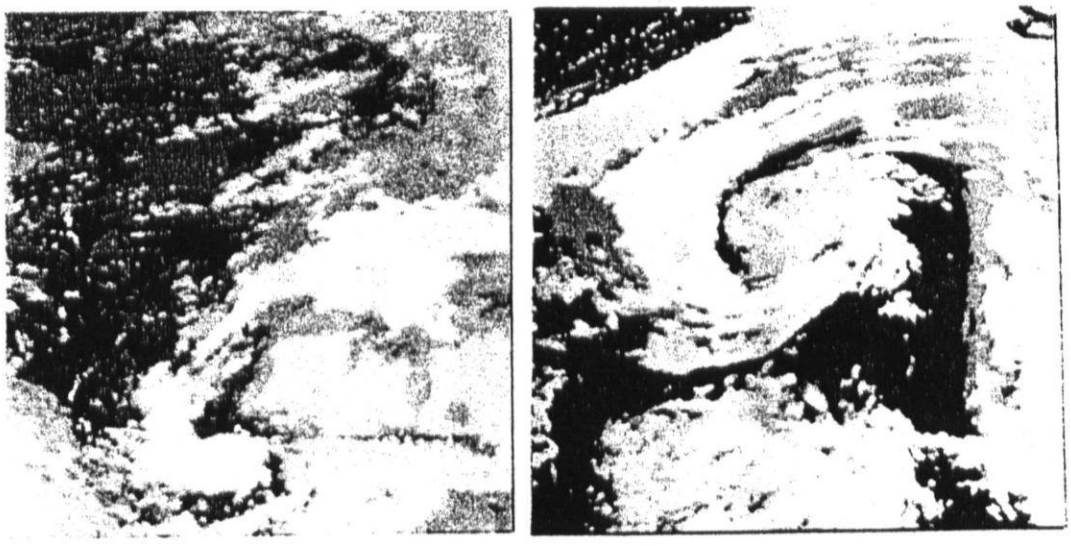
1. Deep and thick cirrus.
2. Altostratus.
3. Stratus and stratocumulus.
4. Low clouds and land.
5. Sea.

Since only visible (0·4–1·1 μm) and infrared (10·5–12·5 μm) images are used, these features are insufficient to separate all cloud types. For example, in most cases the clustering algorithm has failed to separated thin cirrus, (using the water vapour (5·7–7·1 μm) image can improve the separation of thin cirrus, but this is only available every hour). However, it is clear that clustering is able to separate *major* cloud features for subsequent tracking.

On the 5 March three thick masses of class 2 are clearly distinguished, while over Biscay a secondary mesoscale depression is comprised of stratus and stratocumulus and appears bright on the cluster map. Class 1 associated with the fronts on the occluding low is evident over the United Kingdom. On the 8 March classes 1 and 2 are well-represented in the spiral frontal cloud while the clearance to the north-west shows cellular convection of class 4. On the 11 March the complex meteorological situation presents a problem in identifying class 4. However, deep thick cirrus is identifiable in the upper jet stream flow. Class 3 is well-represented to the south over Spain and in the North Sea.

5.2. *Wind vectors*

Figure 7 shows cloud motion vectors for 5, 8 and 11 March using raw and clustered approaches. Here, all tracking was done on infrared images, using the SSDA method. The vectors shown have been selected from the original set by cross-checking with data generated by the Meteorological Office fine resolution model at three levels (850 mb, 500 mb, and 250 mb). These vectors are all within a speed deviation of less than 50 per cent and a direction deviation of less than 30°. The wind vectors are assigned to the level of best fit, i.e., to the level with minimum speed



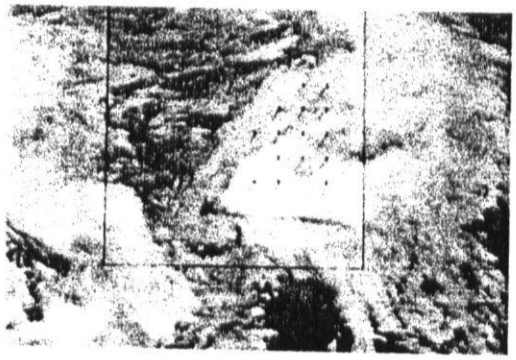
5 March 8 March

11 March

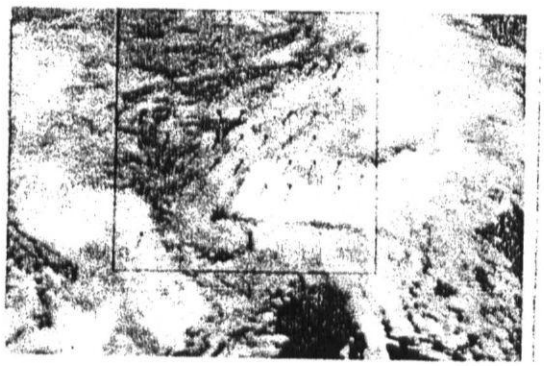
Figure 6. Cluster map of the images for 5, 8 and 11 March 1991.

deviation. It should be noted that wind density using a large template size can be increased by overlapping the target windows.

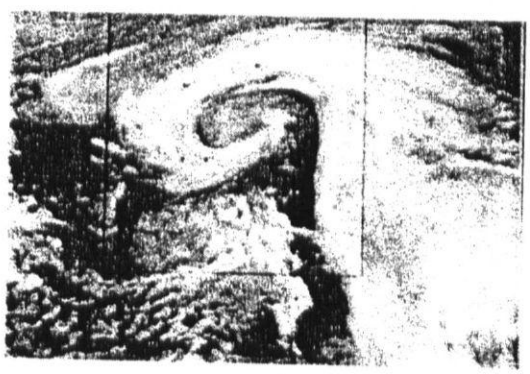
Tables 1 to 3, gives the detailed cloud motion wind results for the three sequences. These show that, for a target size greater than 8 by 8, the number of 'valid' wind vectors computed using clustered images is significantly more than for raw images (figure 8). The diminishing advantage of using clustered image tracking for target sizes below 16 by 16 may be partly due to the diminishing chance of



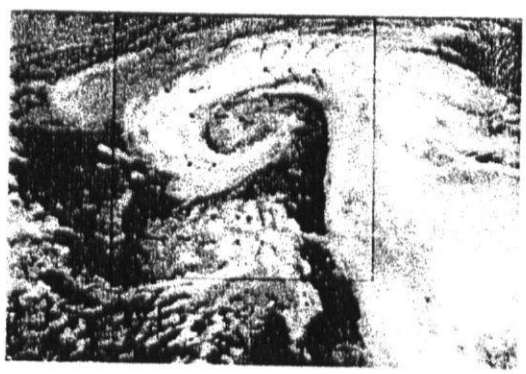
Raw image 5 March



Clustered image 5 March

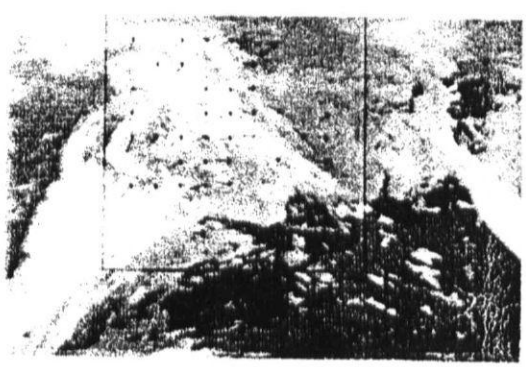


Raw image 8 March



Clustered image 8 March

Raw image 11 March



Clustered image 11 March

Figure 7. Wind field generated using 24 by 24 target window, with raw and clustered images (SSDA tracking).

Table 1 (a). 5 March 1991, cloud motion wind results using raw images, where tracking method × represents cross-correlation, ssda for sequential similarity detection algorithm, and 2-ds for two-dimensional search. 'Valid vectors' see § 5.2.

Original vectors	'Valid' vectors	Mean speed error (knots)	Mean dir. error (deg.)	R.m.s. speed error (knots)	Target size	Tracking method
178	50	4·93	−1·7	9·95	4 by 4	×
197	121	3·84	−0·9	8·39	4 by 4	ssda
208	120	1·32	−1·4	6·13	4 by 4	2ds
106	86	3·38	0·6	7·47	8 by 8	×
142	122	2·78	−1·9	7·36	8 by 8	ssda
100	77	2·03	−2·1	6·63	8 by 8	2ds
64	61	2·99	−4·6	6·63	16 by 16	×
67	61	1·61	−5·1	5·58	16 by 16	ssda
48	41	0·87	−4·2	5·75	16 by 16	2ds
27	25	2·11	−1·0	5·60	24 by 24	×
24	24	0·04	1·4	4·37	24 by 24	ssda
21	20	−1·01	3·7	4·76	24 by 24	2ds
12	11	0·13	0·2	6·19	32 by 32	×
11	11	0·41	−1·7	7·18	32 by 32	ssda
9	8	−0·33	−0·8	7·39	32 by 32	2ds

Table 1 (b). 5 March 1991, cloud motion wind results using clustered images.

Original vectors	'Valid' vectors	Mean speed error (knots)	Mean dir. error (deg.)	R.m.s. speed error (knots)	Target size	Tracking method
267	46	6·25	3·8	10·54	4 by 4	×
267	123	4·16	2·1	8·17	4 by 4	ssda
296	136	1·24	−3·8	5·71	4 by 4	2ds
123	78	3·26	−1·3	7·15	8 by 8	×
154	112	3·45	−2·0	7·58	8 by 8	ssda
123	80	1·91	−1·2	5·86	8 by 8	2ds
75	63	2·27	−1·9	5·97	16 by 16	×
77	62	1·98	−3·2	5·83	16 by 16	ssda
62	45	1·84	−1·9	4·61	16 by 16	2ds
43	28	1·69	−0·3	5·47	24 by 24	×
39	28	1·88	−3·6	5·1	24 by 24	ssda
33	22	1·77	−1·3	5·48	24 by 24	2ds
28	22	1·14	−1·1	6·28	32 by 32	×
29	22	1·01	−0·8	4·69	32 by 32	ssda
23	20	0·53	−0·9	5·49	32 by 32	2ds

selecting a target covering multi-layer cloud. The general increase in the number of vectors using clustered image tracking strongly suggests that multi-layer cloud motion can be tracked better by first separating different cloud types and tracking them individually. It is also found that clustering provides less improvement over the raw image approach when the sequence has a relatively uniform wind field (5 March).

Winds were generated using five different target sizes, and figure 9 shows the r.m.s. speed deviation versus target size. Winds are compared with interpolated values using four nearest grid points; the resolution of the analysed data is 0·75° in the north–south direction and 0·9375° in the east–west direction respectively. All

the three tracking methods, produces a clear minimum (at a target size of 24 by 24). for low level wind vectors.
Lunnon and Lowe (1990) used target sizes of 4 by 4, 4 by 8, 8 by 8, 8 by 16, 16 by 16, 16 by 32 and 32 by 32, and found the optimum target size for Meteosat images to be 16 by 16. Note that no optimum target size can be defined for the clustered approach since the window may not be filled with cloud pixels.

It is clear that if the target size is either too small or too large for a specific wind resolution, then the error will increase. In other words, there is an optimum target size for a given wind resolution. The larger error with small target size is probably due to the lack of cloud features, while for large targets, the wind is the weighted mean of the small scale winds within the window.

Table 2(a). 8 March 1991, cloud motion wind results using raw images.

Original vectors	'Valid' vectors	Mean speed error (knots)	Mean dir. error (deg.)	R.m.s. speed error (knots)	Target size	Tracking method
214	77	5·31	1·9	7·95	4 by 4	×
241	118	4·31	−0·6	7·75	4 by 4	ssda
288	138	1·57	0·6	5·06	4 by 4	2ds
106	75	3·09	0·9	7·67	8 by 8	×
131	87	3·00	2·0	6·38	8 by 8	ssda
115	72	1·45	0·3	5·03	8 by 8	2ds
43	32	4·38	4·9	10·63	16 by 16	×
42	29	1·54	5·0	4·74	16 by 16	ssda
43	29	1·07	3·9	4·24	16 by 16	2ds
13	9	−3·38	1·9	4·41	24 by 24	×
13	9	−1·53	−0·5	3·14	24 by 24	ssda
14	9	−1·89	−3·2	3·49	24 by 24	2ds
5	3	2·21	7·6	3·18	32 by 32	×
6	4	4·15	12·1	5·57	32 by 32	ssda
6	4	4·15	12·1	5·57	32 by 32	2ds

Table 2(b). 8 March 1991, cloud motion wind results using clustered images.

Original vectors	'Valid' vectors	Mean speed error (knots)	Mean dir. error (deg.)	R.m.s. speed error (knots)	Target size	Tracking method
268	62	4·43	1·2	7·5	4 by 4	×
303	107	4·56	0·6	7·71	4 by 4	ssda
370	141	2·47	2·0	6·18	4 by 4	2ds
128	81	5·12	−1·9	9·55	8 by 8	×
155	93	3·64	0·5	7·74	8 by 8	ssda
136	84	0·39	0·7	5·51	8 by 8	2ds
69	47	2·73	1·4	9·34	16 by 16	×
66	45	1·34	2·3	4·55	16 by 16	ssda
65	42	1·35	−0·1	4·0	16 by 16	2ds
41	27	2·50	1·1	6·33	24 by 24	×
44	29	1·65	2·2	4·72	24 by 24	ssda
32	20	0·58	−2·9	4·17	24 by 24	2ds
25	17	2·20	3·0	7·16	32 by 32	×
26	16	0·3	9·0	4·97	32 by 32	ssda
26	15	−0·51	11·4	6·09	32 by 32	2ds

Table 3(a). 11 March 1991, cloud motion wind results using raw images.

Original vectors	'Valid' vectors	Mean speed error (knots)	Mean dir. error (deg.)	R.m.s. speed error (knots)	Target size	Tracking method
244	118	2·88	6·3	6·19	4 by 4	×
321	197	2·06	4·2	6·51	4 by 4	ssda
303	185	1·98	1·4	5·25	4 by 4	2ds
142	79	2·82	8·0	6·19	8 by 8	×
160	119	1·62	5·3	5·56	8 by 8	ssda
116	79	1·17	3·8	5·17	8 by 8	2ds
46	36	1·45	7·2	4·41	16 by 16	×
40	31	1·34	4·4	4·14	16 by 16	ssda
39	30	0·53	4·7	3·81	16 by 16	2ds
17	14	−0·41	6·0	4·94	24 by 24	×
22	18	−0·99	3·2	3·01	24 by 24	ssda
21	17	−1·60	0·7	3·17	24 by 24	2ds
7	6	−0·52	8·5	3·26	32 by 32	×
4	3	−1·04	5·3	3·84	32 by 32	ssda
5	4	−1·13	3·1	2·83	32 by 32	2ds

Table 3(b). 11 March 1991, cloud motion wind results using clustered images.

Original vectors	'Valid' vectors	Mean speed error (knots)	Mean dir. error (deg.)	R.m.s. speed error (knots)	Target size	Tracking method
306	95	3·25	4·12	7·11	4 by 4	×
357	179	1·88	4·6	6·89	4 by 4	ssda
358	163	1·01	2·0	4·78	4 by 4	2ds
147	103	2·79	8·6	6·43	8 by 8	×
172	117	1·53	6·0	5·08	8 by 8	ssda
129	85	0·44	0·41	4·52	8 by 8	2ds
82	58	1·59	7·6	5·25	16 by 16	×
83	58	1·57	5·8	5·47	16 by 16	ssda
66	44	1·30	5·2	4·51	16 by 16	2ds
46	36	1·87	9·6	4·52	24 by 24	×
46	36	0·56	6·5	4·05	24 by 24	ssda
38	31	0·22	4·2	4·09	24 by 24	2ds
27	22	0·69	8·0	4·96	32 by 32	×
27	22	0·43	6·1	4·45	32 by 32	ssda
31	26	−0·5	4·8	3·85	32 by 32	2ds

Figure 10 compares the number of vectors generated using different tracking methods. The number of vectors generated by cross-correlation is much less than SSDA for target size less than 16 by 16, but for larger targets the number of vectors are approximately the same. This suggests that SSDA is a more reliable method than cross-correlation. Also, the number of vectors generated by 2-d search is comparable to that for SSDA and cross-correlation, and, since the computation time is only a fraction of that for cross-correlation, a 2-d search provides a quick alternative for cloud motion tracking.

6. Conclusion

A new clustering scheme has been developed for automatic clustering of cloud images. The hybrid scheme uses a histogram algorithm to select the starting centres.

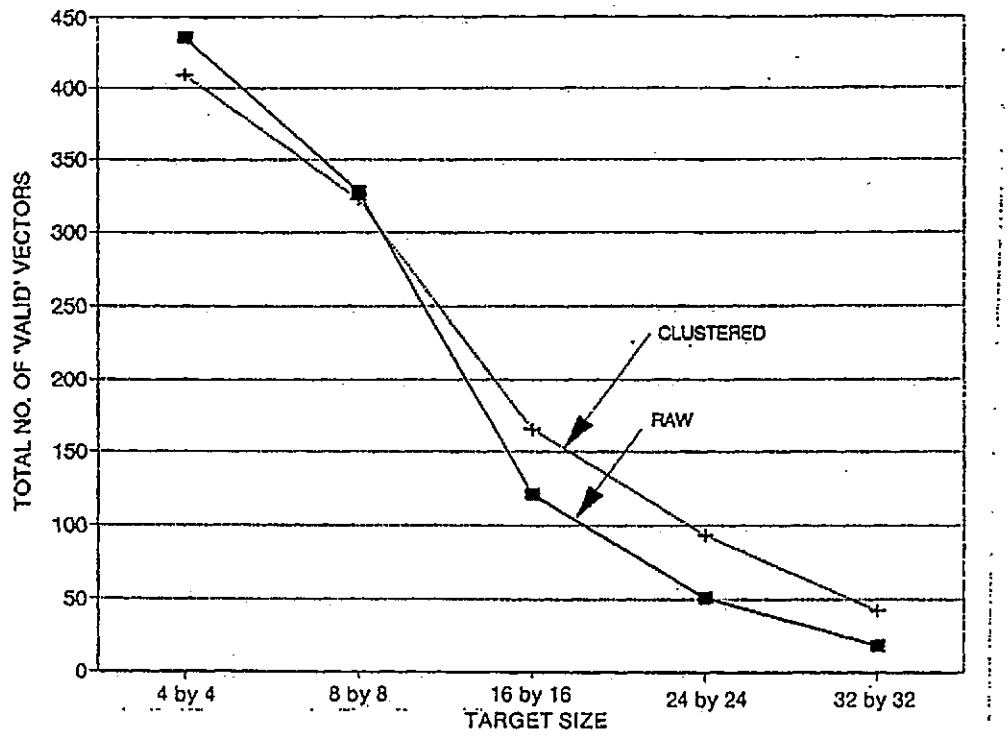


Figure 8. Comparison of total number of 'valid' vectors using raw and clustered images (SSDA tracking).

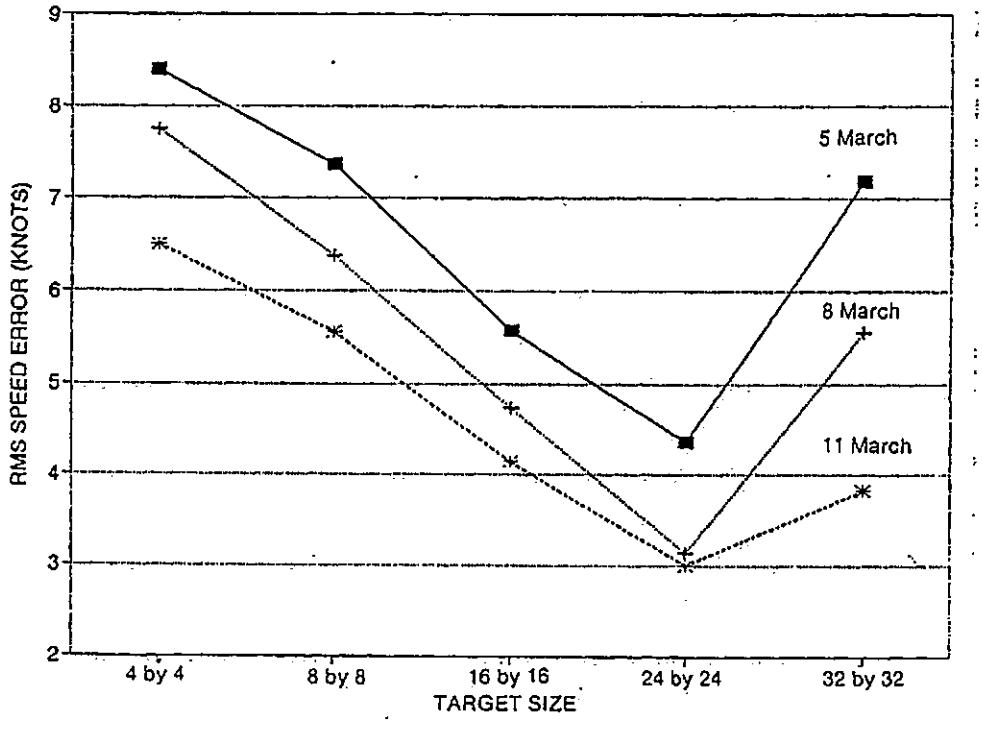


Figure 9. R.m.s. speed error vs. target size using SSDA for tracking (raw images).

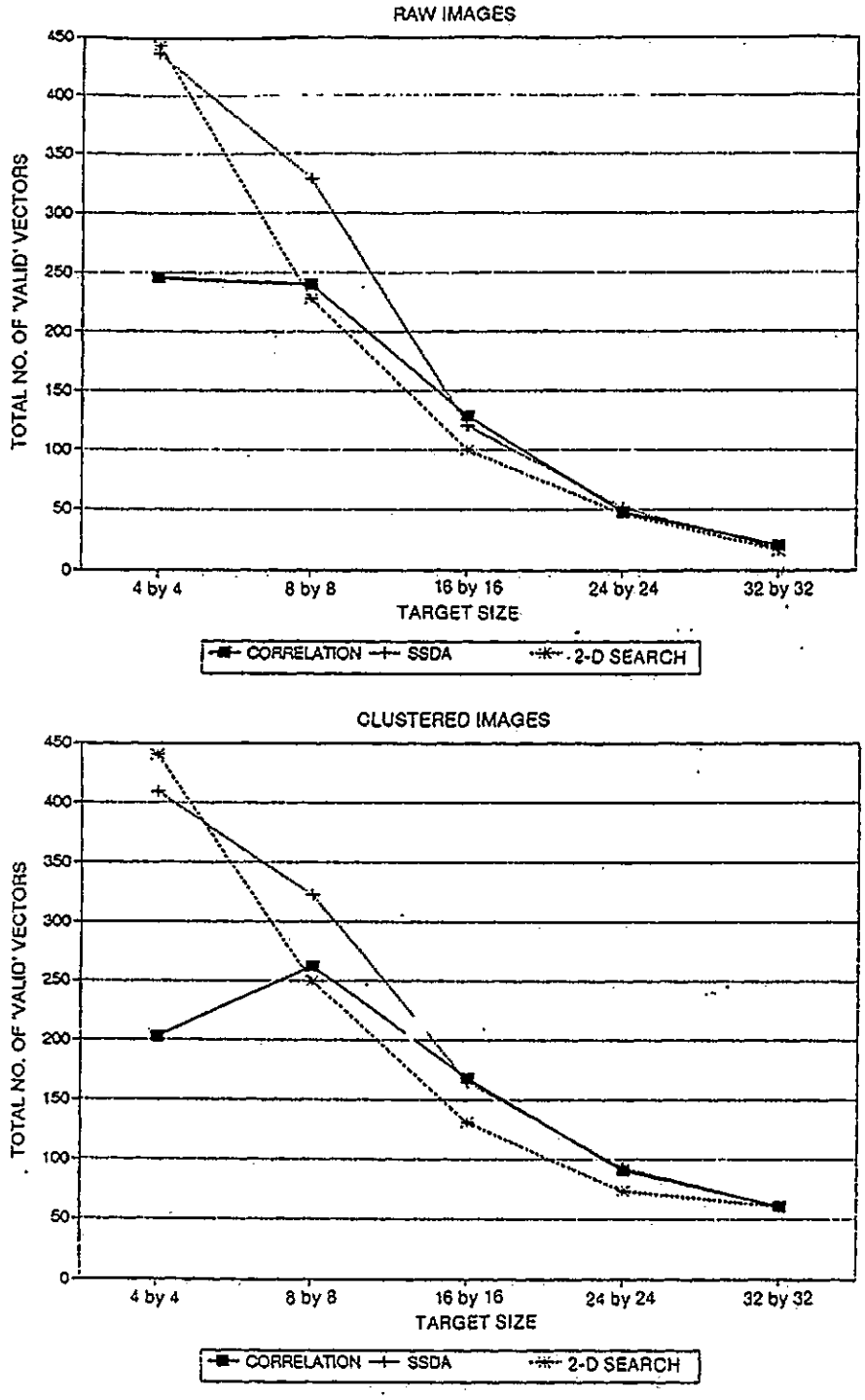


Figure 10. Comparison of the number of 'valid' vectors generated by cross-correlation. SSDA and 2-d search using raw and clustered images.

This replaces manual selection (which is inefficient and subjective) and also reduces convergence time sinced the startrring centres generated are a good representation of the underlying clusters. Convergence time has also been reduced by clustering the multi-dimensional histogram rather than individual pixels. An optimised model has been used, which allows clusters with different variance and population. The hybrid algorithm has not been extensively tested e.g., for all four seasons, but has been found to perform well on data sets spread over several months.

Experimental results suggest that clustering before target tracking can significantly improve cloud motion wind estimates, although any advantage may be small for uniform wind fields. The advantage stems from the ability of clustering to select natural data patterns, which in turn tend to correspond to different cloud types.

It is also found that there is an optimum target size for a given wind resolution. In our case, the optimum target size is around 24 by 24 for a wind resolution of 0·75° by 0·9375°. However, this criteria does not apply to clustered image tracking, since, in this case, the effective target size is variable.

A comparison between the classic cross-correlation approach and the SSDA indicates that SSDA is a more reliable method for target tracking; it is also much faster since it avoids the need for normalisation. The 2-d search can be used when a large wind field is required.

Acknowledgment

This work is carried out under SERC grant GR/E74007. We are also grateful to Dr Lyne of the Meteorological Office, Bracknell, U.K., for supplying data from their fine resolution model.

References

ANDERBERG, M. R., 1973, *Cluster analysis for applications*, (Academic Press, Inc), chapter 7.

BALL, G. H., and HALL, D. J., 1967, A clustering technique for summarizing multivariate data. *Behavioural Science*, **12**, 153–155.

BARNEA, D. I., and SILVERMAN, H. F., 1972, A class of algorithms for fast digital image registration. *I.E.E.E. Transactions on Computers*, **C-21**, 179–186.

BOS, A. M., DE WAARD, J., and ADAMSON, J., 1990, Real-time rectification of Meteosat images. *European Spaced Agency Journal*, **14**, 179–191.

BOWEN, R. A., and SAUNDERS, R. W., 1984, The semi-transparency correction as applied operationally to Meteosat infrared data: A remote sensing problem. *European Space Agency Journal*, **8**, 125–131.

BOWEN, R. A., FUSCO, L., MORGAN, J., and ROSKA, K. O., 1979, Operational production of cloud motion vectors (satellite winds) from Meteosat image data. Use of Data from Meteorological Satellites, Technical Conference, Lannion, France, E.S.A. SP-143 (Paris: E.S.A.) pp. 27–37.

DESBOIS, M., SEZE, G., and SEWJWACH, G., 1982, Automatic classification of clouds on Meteosat imagery: Application to high-level clouds. *Journal of Applied Meteorology*, **21**, 401–412.

ENDLICH, R. M., and WOLF, D. E., 1981, Automatic cloud tracking applied to GOES and Meteosat observations. *Journal of Applied Meteorology*, **20**, 309–310.

HUBERT, L. F., 1979, Wind derivation from geostationary satellites. In *Quantitative Meteorological Data from Satellites. World Meteorological Organization Technical Note No. 66*, edited by J. S. Winston, pp. 33–59.

HUBERT, L. F., and WHITNEY, L. F. JR., 0000, Wind estimation from geostationary satellite pictures. *Monthly Weather Review*, 99, 665–672.

HOFFMAN, J., 1990, Cloud motion wind retrieval in multilayered areas. *Journal of Geophysical Research*, awaiting publication.

JAIN, J. R., and JAIN, A. K., 1981, Displacement measurement and its application in interframe image coding. *I.E.E.E. Transactions on Communications*, COM-29, pp. 1799–1808.

KITTLER, J., and PAIRMAN, D., 1985, Segmentation of multispectral imagery using iterative clustering. *Proceedings of the 4th Scandinavian Conference on image analysis, Trondhein*, I, 39–40.

KITTLER, J., and PAIRMAN, D., 1988, Optimality of reassignment rules in dynamic clustering. *Pattern Recognition*, 21, 169–174.

KOONTZ, W. L. G., NARENDRA, P. M., and FUKUNAGA, K., 1976, A graph-theoretic approach to nonparametric cluster analysis. *I.E.E.E. Transaction on Computers*, C-25, 936–944.

LESSE, J. A., and NOVAK, C. S., 1971, An automated technique for obtaining cloud motion from geosynchronous satellite data using cross-correlation. *Journal of Applied Meteorology*, 10, 119–132.

LUNNON, R. W., and LOWE, D. A., 1990, Spatial scale dependency of errors in satellite cloud track winds. COSPAR, the Hague, No. MA 1.1.7, 26 June 1990.

NARENDRA, P. M., and GOLDBERG, M., 1977, A non-parametric clustering scheme for Landsat. *Pattern Recognition*, 9, 207–215.

PAIRMAN, D., and KITTLER, J., 1986, Clustering algorithms for use with images of clouds. *International Journal of Remote Sensing*, 7, 855–866.

PARIKH, J. A., 1977, A comparative study of cloud classification techniques. *Remote Sensing of Environment*, 6, 67–81.

PARIKH, J. A., and ROSENFLED, A., 1978, Automated segmentation and classification of infrared meteorological satellite data. *I.E.E.E. Transaction System, Man and Cybernetic*, SMC-8, 736–743.

SEDDON, A. M., and HUNT, G. E., 1985, Segmentation of clouds using cluster analysis. *International Journal of Remote Sensing*, 6, 717–731.

SCHMETZ, J., and NURET, M., 1987, Automatic tracking of high-level clouds in Meteosat infrared images with a radiance windowing technique. *European Space Agency Journal*, 11, 275–286.

SCHMETZ, J., and HOLMLUND, K., 1990, Operational cloud motion winds from Meteosat and the use of cirrus clouds as tracers. COSPAR, the Hague, No. MA 1.1.1, 26 June 1990.

TURNER, J., and WARREN, D. E., 1989, Cloud track winds in the polar regions from sequences of AVHRR images. *International Journal of Remote Sensing*, 10, 695–703.

WHARTON, S. W., 1983, A generalised histogram clustering scheme for multidimensional image data. *Pattern Recognition*, 16, 193–199.

WILSON, G. S., 1984, Automated meoscale wind fields derived from GOES satellite imagery. *American Meteorology Society Conference on Satellite/Remote Sensing Application* ( ) pp. 164–171.