2003

INTELLIGENT VISION-BASED NAVIGATION SYSTEM

KOAY, KHENG LEE

INTELLIGENT VISION-BASED NAVIGATION SYSTEM

By

KHENG LEE KOAY

A THESIS SUBMITTED TO THE UNIVERSITY OF PLYMOUTH
IN A PARTIAL FULFILLMENT FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING
FACULTY OF TECHNOLOGY

March 2003

Research Supervisor: Dr. Guido Bugmann

To my parents and my grandmother,
Ah The and Aik Jin Koay
Siew Eng Quah

Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was supported by DEVR funding from HEFCE for "Facility in Mobile Robotics".

A programme of advanced study was undertaken, including two postgraduate courses in Neural Computation (COIN 501 and COIN 506)

Relevant scientific seminars and conferences were regularly attended at which work was often presented; external institutions were visited for consultation purposes and several papers were prepared for publication.

Publications:

- Koay, K. L., Bugmann, G., Barlow, N. and Philips, M. (1998). Representation of Trajectories for Mobile Robot. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 185-194.
- Bugmann G., Koay K., Barlow N., Phillips M. and Rodney D. (1998). Stable encoding of robot trajectories using normalised radial basis functions: Application to an autonomous wheelchair. In: D Caldwell, J Gray and P Robinson (eds), *Proceedings of 29th International Symposium Robotics (ISR'98)*, Pages 232-235. DMG Publishers: London.

Presentations and Conferences attended:

- 6th International Symposium on Intelligent Robotic Systems '98 (SIRS98), Edinburgh, Scotland, UK, July 21-23, 1998
- Centre for Neural and Adaptive Systems (CNAS) and School of Computing research seminars.

Signed.....

Date...19-5-2003....

Author: **Kheng Lee Koay**

Title: **Intelligent Vision-based Navigation System**

Abstract

This thesis presents a complete vision-based navigation system that can plan and follow an obstacle-avoiding path to a desired destination on the basis of an internal map updated with information gathered from its visual sensor.

For vision-based self-localization, the system uses new floor-edges-specific filters for detecting floor edges and their pose, a new algorithm for determining the orientation of the robot, and a new procedure for selecting the initial positions in the self-localization procedure. Self-localization is based on matching visually detected features with those stored in a prior map.

For planning, the system demonstrates for the first time a real-world application of the neural-resistive grid method to robot navigation. The neural-resistive grid is modified with a new connectivity scheme that allows the representation of the collision-free space of a robot with finite dimensions via divergent connections between the spatial memory layer and the neuro-resistive grid layer.

A new control system is proposed. It uses a Smith Predictor architecture that has been modified for navigation applications and for intermittent delayed feedback typical of artificial vision. A receding horizon control strategy is implemented using Normalised Radial Basis Function nets as path encoders, to ensure continuous motion during the delay between measurements.

The system is tested in a simplified environment where an obstacle placed anywhere is detected visually and is integrated in the path planning process.

The results show the validity of the control concept and the crucial importance of a robust vision-based self-localization process.

Acknowledgements

In the early 18th century, men begun taking on the challenge of making mechanisms that imitate parts of human body. Since then, the target has moved on to making intelligent machines to support human needs, and among the research community, to test research theories and investigate human intelligent behaviour. The development of mobile robots marks the turning point of the development of sophisticated machines. The work presented here represents a small step in the ongoing effort to develop machine intelligence.

First of all, I would like to give my most heartfelt thanks to my supervisor Dr. Guido Bugmann who have been supervising my work diligently over these years, and most of all for his consistent guidance, ideas and kindness.

I am also grateful to Ian Rowlands from the University's Tech Service for his kind support and John Eastman from the Department of Communication and Electronic Engineering for designing a new shaft encoder disc which has been used in the project.

I would also like to extend my thanks to the members of our group at the Centre for Neural and Adaptive Systems of the School of Computing in the University of Plymouth for sharing, and the assistance and advices generously given.

Not forgotten are all my good friends, Yann Chew Tan and his wife Ngan, Ling-Ti Lin, Lisa He, Trung Nguyen, Steven Lee, Elisa Mak, Jaslyn Yap, Kaemi Yamamoto, Angie Ng, Lavender Liong, Bee Ling Tan, Kian Lip Kee, Junko Hozu, Nicole Ng, Cathy Radix and others whom I have forgotten to mentions, thank you all for everything you all have done for me, I cherish our friendship.

Last but not least, to HEFCE for the DEVR funding for a "Facility for Mobile Robotics" that made this study possible.

Kheng Lee Koay

Plymouth, March 2003

Contents

Abstract i

Acknowledgements ii

Contents iv

List of Figures vii

List of Tables x

1. Introduction 1
1.1. Aims of this Project 1
1.2. Method 3
1.3. Overview of the Thesis 3

2. Literature Review 6
2.1. Intelligent Vision-based Navigation in Robotics 7
2.2. Control with Intermittent Sensing 8
2.3. Navigation 10
2.3.1. Self-localization 10
2.3.2. Map Building 13
2.3.3. Path Planning 16
2.4. Spatial Vision 19

3. Experimental Setup 22
3.1. The Robotic System 23
3.1.1. The Modified Rug Warrior Robot 23
3.1.2. Video Camera 24
3.1.3. The Video Sender 25
3.1.4. Micro Fast Servo 25
3.1.5. Serial Servo Controller 25
3.1.6. The Serial Transceiver (418 MHz FM - 1200 Baud) 26
3.2. The Computer System "Remote Brain" 27
3.2.1. Software "Cortex-Pro" 28
3.2.2. Win Vision Framegrabber (QUANTA) 28
3.3. Environment 29

4. Vision-based Obstacles Detection, Self-localization and Map Updating 30
4.1. Correction of Lens Distortion ... 33
4.1.1. Fish -eye Lens Effect ... 33
4.1.2. Fish-eye Lens Effect Correction ... 34
4.2. Floor-specific Edge Detection ... 38
4.2.1. Image Segmentation (Floor/non-floor) ... 38
4.2.2. Vertical and Horizontal Edges Filter Design ... 40
4.2.3. Calculation of the Edge Position and Orientation ... 42
4.2.3.1. Edge Orientation ... 42
4.2.3.2. Edge Positioning - Basic Method ... 48
4.2.3.3. Edge Positioning - Refined Method ... 49
4.3. Coordinate Transformations for Vision System ... 51
4.3.1. Image Coordinate Frame to Camera Coordinate Frame Transformation ... 51
4.3.2. Camera Coordinate Frame to Map Coordinate Frame Transformation for Obstacles and Walls ... 55
4.3.2.1. Camera Coordinate Frame to Robot Coordinate Frame Transformation ... 57
4.3.2.2. Robot Coordinate Frame to Map Coordinate Frame Transformation ... 58
4.4. Vision-based Self-localization ... 59
4.5. Self-localization Tests ... 62
4.5.1. Results ... 62
4.5.2. Discussion of the Vision-based Self-localization Results ... 66
4.6. Obstacle Detection and Registration ... 69
4.7. Discussion ... 70

5. Path Planning and Encoding 72
5.1. Path Planning through the Neural-resistive Grid ... 73
5.1.1. Neural-resistive Grid ... 73
5.1.2. Representation of Robot and Obstacles in the Neuro-resistive Grid ... 76
5.1.3. Path Planning through Gradient Climbing in the Neuro-resistive Grid ... 78
5.2. Path Encoding and Decoding through NRBF Nets ... 79
5.2.1. The Normalised Radial Basis Function (NRBF) Net ... 80
5.2.2. Path Encoding ... 82
5.2.3. Path Decoding ... 83

6. Motion Control with Intermittent Delayed Measurements 85
6.1. The Time Delays Problem ... 87
6.1.1. Sequential Control "Compute then Move" ... 87
6.1.2. Concurrent Control "Compute while Moving" ... 91
6.2. Proposed Implementation of Concurrent Control ... 93
6.2.1. Receding Horizon Control Strategy ... 95
6.2.2. Modified Smith Predictor for Intermittent Delayed Feedback ... 98
6.3. Fast Feedback Loop in the Smith Predictor ... 101
6.3.1. Building a Dynamical Model of the Robot ... 101
6.3.1.1. Collecting Modelling Data ... 102
6.3.1.2. Derivation of the Rug Warrior's Motion Model ... 104
6.3.1.3. Parameters Fitting ... 108
6.3.2. Odometric Motion Tracking ... 111
6.3.2.1. The Robot in Straight Motion ... 111

6.3.2.2. The Robot in a Curved Motion 112
6.3.2.3. Conversion to Map Coordinate System 114
6.4. On-board Path Control 116
6.5. Retroactive Position Calibration Using Visual Feedback 119
6.5.1. Recalibration Equations 119
6.5.2. Discussion 124

7. Results **128**
7.1. Experiments and Results 128
7.1.1. Experiments Description 128
7.1.2. Results 133
7.1.3. Problems Encountered 135

8. Conclusions and Future Work **138**
8.1. Contributions to Knowledge 138
8.2. Problems and Difficulties Encountered 140
8.3. Future Work 143

Appendices

A. Solving the Rug Warrior's Motion Model using Linear Differential Equation of 1st order **145**

B. Publications **147**

C. CD-ROM contains video clips and program source codes. **162**
1. Video Clips demonstrating the system performing the navigation tasks.
2. Program Source Codes of the Computer System "Remote Brain".
3. Program Source Code of the Robotic System.
4. Program Source Code of the Robot Tracking with an Overhead Camera.

Bibliography **163**

List of Figures

3.1 The commercial Rug Warrior robot 23
3.2 The modified Rug Warrior robot 26
3.3 The architecture of the Robotic System 27
3.4 The architecture of the Computer System "Remote Brain" 28
3.5 The modified Rug Warrior in its environment with the presence of an obstacle 29

4.1 Vision-based processes for obstacles detection, self-localization and map updating 31
4.2 The fish-eye lens effect on a square grid 33
4.3 This figure illustrates the fish-eye lens effect and its correction 34
4.4 The robot's vision system image sampling process 35
4.5 The fish-eye lens distorted image correction model 36
4.6 The selected portion of the distortion free image used for vision processing 37
4.7 The automatic thresholding process 39
4.8 The horizontal and the vertical filters 40
4.9 The edge filtering process for detecting edges and determining their positions and orientations 40
4.10 The four cases with each representing a quadrant within the circle 42
4.11 Examples of possible image configurations encountered during the filtering process 43
4.12 Edge filter in case 1 44
4.13 Edge filter in case 2 45
4.14 Edge filter in case 3 46
4.15 Edge filter in case 4 47
4.16 Comparison between the basic and the refined methods of edges positioning 49
4.17 This figure shows the camera module 51
4.18 The side view of the camera geometry map used for determining the y coordinates of the detected edges 52
4.19 The camera geometry map used for determining the x coordinates of the detected edges 53
4.20 Side view of the concept diagram used to determine the angle for each line in the calibration grid 54
4.21 The Pixel-Angle relationship graph showing the measured data plot and their fitted function 55
4.22 Illustration of the relationship between the camera coordinate frame, the robot's egocentric coordinate frame and the robot's coordinate frame 56
4.23 The location of pixel P in the robot coordinate system 58
4.24 This figure illustrates the transformation of coordinate from robot coordinate system to the map coordinate system 59

4.25 Illustration of the vision-based self-localization process 60
4.26 The positions and orientations used in the vision-based self-localization test 63
4.27 Examples of vision-based self-localization tests plots 64
4.28 Standard deviation of vision-based self-localization errors 65
4.29 Example of the image taken at coordinate (45, 16) 66
4.30 Examples of the vision-based self-localization tests results 68
4.31 The process of registering the detected obstacle into the spatial memory layer of the neural-resistive grid 70

5.1 The neural-resistive grid planner is composed of the neuro-resistive grid layer and the spatial memory layer 74
5.2 Transfer function of the neurons representing nodes of the neuro-resistive grid 75
5.3 Modified neural-resistive grid with one-to-many connections from the spatial memory layer to the resistive grid layer 77
5.4 Representation of walls, obstacle and collision free space within the neural-resistive grid 77
5.5 The neural-resistive grid representation 78
5.6 Waypoints representation of the path within the neuro-resistive grid 79
5.7 The NRBF path encoder 79
5.8 Network architecture for standard RBF nets and Normalized RBF nets .. 80
5.9 Comparison between standard RBF nets and Normalized RBF nets with three hidden nodes on an example of a 1-Dimensional path 82
5.10 A simulation of the NRBF path encoder attractive field 84

6.1 Vision-based Navigation System Sequential Control Flow Diagram 88
6.2 Tasks scheduling diagram for the sequential control vision-based navigation system 89
6.3 Vision-based Navigation System Sequential Control timing diagram 90
6.4 Timing diagram of a Vision-based Navigation System with Concurrent Control 91
6.5 Tasks scheduling diagram for the concurrent control vision-based navigation system 92
6.6 Concurrent control process flow diagram 93
6.7 The concept of receding horizon control strategy 95
6.8 This figure illustrates the advantage of applying the receding horizon control strategy using the waypoints method 97
6.9 Classical diagram of a control system incorporating a Smith Predictor .. 98
6.10 Modified Smith Predictor for Intermittent Delayed Feedback 99
6.11 Operation sequence in the proposed system that uses a modified Smith Predictor 100
6.12 Operation Sequence in the final system inspired by Smith Predictor 101
6.13 Illustration of the data collection protocol and an example of the encoder's strip pattern (i.e. black strip absorb light and white strip reflect light) glued to the wheel 102
6.14 The motor dynamics plot for 10 runs 103
6.15 The motor dynamics plot for 10 runs after data correction 104
6.16 This figure illustrates the relation between each of the motor coefficients and their effect on the model output 108

6.17 This figure shows the plot of data collected from the motor (dots) and the motor model (solid line) 109
6.18 The actual maximal velocity as a function of the speed commands 109
6.19 The test results of the robot following two prescribed paths based on the motors model as feedback 111
6.20 Conceptual diagram for the robot doing a leftward motion 113
6.21 Conceptual diagram used for updating the robot position in the model map 115
6.22 The result of the controller steering the robot from the initial position toward the goal based only on the shaft encoders input as feedback 118
6.23 This figure illustrates the concepts of coordinate recalibration 120
6.24 Diagram for the robot orientation recalibration algorithm 121
6.25 Diagram for the robot coordinate recalibration algorithm 123
6.26 The modified Smith Predictor for navigation applications 126
6.27 The modified Smith Predictor for navigation application with intermittent feedback 127

7.1 The overhead camera setup used to record the robot's motion during the experiments 129
7.2 Example of the concurrent control system performing the navigation task of experiment 2 132
7.3 The Distance vs. Time plot of two systems that uses different control methods 133
7.4 The experimental paths produced by the sequential control system (a, b) and the concurrent control system (c,d) 134

List of Tables

3.1 The commercial Rug Warrior robot's technical specifications 24

4.1 The equations system used to determine the orientation of the detected edge 41

6.1 The coefficients for each of the command speed obtained through curve fitting 108

Chapter 1

Introduction

1.1 Aim of this Project

Research in the field of mobile robotics has received considerable attention in the past decade due to its wide range of potential applications. One area of special interest is household robotics for the disabled and the elderly persons. In general, a household robot needs to be able to perform several tasks. Among these, object fetching is a generic task which itself consists of several sub-tasks. One of the sub-tasks this research focuses on is goal directed navigation.

Given the need for artificial vision in most domestic tasks, this study also uses vision to acquire spatial information for navigation. Designing an effective navigation system requires the integration of current knowledge or development of new methods in various fields such as object recognition, spatial vision, spatial knowledge representation, path planning, motion control, obstacle avoidance and power resources management.

The scenario forming the background of this work is that of a domestic robot fetching an object in a room cluttered with obstacles. A number of simplifications were made to the scenario so that more emphasis can be placed on issues related to the interaction between vision, planning and navigation functions.

The aims are to:

1. design and demonstrate a navigation system that can plan an obstacle-avoiding path to a desired destination on the basis of an internal model (map), updated with information gathered from its visual sensors.

2. investigate and demonstrate a control technique that addresses concurrent image processing and planning while the robot is in motion.

To achieve these aims, the following simplifications were used:

- the goal is at a predefined location that can be varied by the experiment, but does not requires competences in visual object localization.

- obstacles are simple white rectangular blocks, each with the height of 1.5 centimetre, small enough for 2-D approximation, that can be detected visually by the robot's onboard camera as 2-D forbidden areas standing out from the dark floor of the environment. These obstacles can be placed anywhere and considered during planning.

- the environment is a small-scale box of size 125×89 centimetres developed to emulate a room in the real world. The walls and the floor of the small-scale robot's environment were painted with white colour and black colour respectively. This setup simplifies image processing and frees time for exploring other issues related to the overall aim of this research.

- a circular cross-section robot with two drive wheels (which enable the robot to spin around a centre point) is used as a prototype of a domestic robot. This cylindrical robot is free from both the geometric constraints and the piano-mover's problem (Schwartz and Sharir, 1983), therefore the 3-D planning problem is simplified to a 2-D planning problem.

1.2 Method

The work was divided into two stages. The first stage is to develop a visual system that informs a planner about the positions of the robot and obstacles. A simple stop-and-go motion controller was used to test the validity of the approach. In stage two, the motion control problem was addressed. The issue here was to enable uninterrupted motion of the robot to the goal despite long intervals (i.e. of the order of 1 second) between image acquisition and the delayed access to visual information.

1.3 Overview of the Thesis

This thesis consists of eight chapters. This chapter provides an overview of the research, the thesis and the research activity.

Chapter 2 contains pointers to previous work in topics related to this project.

Chapter 3 provides an overview of the experimental setup. This includes a Rug Warrior robot from the MIT modified for a remote-brained control architecture where all computation-intensive processes such as image processing and planning are performed on a remote computer.

Chapter 4 describes vision-based obstacles detection, self-localization and map updating. This chapter begins with the correction of the camera lens distortion (Fish-eye effect), then moves on to the design of novel task-specific floor and non-floor edges detectors, followed by the use of projective geometry for coordinate transformation. The projective geometry coordinate transformation is used to transform the processed image information (i.e. the detected edges and their orientation) from the camera coordinate system to the map coordinate system. The end of this chapter shows how map updating with visually detected obstacles and self-localization are done on the basis of this information.

Chapter 5 deals with path planning and encoding. This chapter begins by describing the use of a neural-resistive grid for path planning and how sections of the pre-planned path are prepared for sending to the robot controller. The use of a Normalised Radial Basis Function (NRBF) neural network for encoding and decoding the path in the robot controller is described.

Chapter 6 looks at the problem of motion control with time delays, and how it is solved. This chapter proposes a solution to the stop-and-go motion problem using a new control technique that combines traditional control methods, which are the Smith Predictor and the receding horizon control strategy, to overcome the problems of computational complexity and speed in image processing and action planning.

Chapter 7 shows the results of a series of navigation experiments and discusses the problems encountered.

Chapter 8 contains the conclusions and describes work suggested for future research.

Chapter 2

Literature Review

A vision-based navigation system (i.e. a mobile robot) must be able to reach an assigned goal by moving and reasoning within its environment without direct human intervention and control. Therefore a navigation system that exhibits such autonomous ability must first be able to perform the sense-think-act process. Such a system is usually equipped with a vision system to sense its environment, a mapping module for prior map updating or building a new map, a planning module for path planning, and a controller for path following. Many different techniques and approaches for mobile robotics on vision, mapping, planning, control and navigation have been developed since the mid-twentieth century to achieve the aim of self-contained autonomy but each has its own advantages and disadvantages.

In general, a vision-based navigation system (mobile robot) is complex to build, difficult to maintain and extremely fragile, as each part of the system depends on all others to function (e.g. the mapping process depends on the vision system).

2.1 Intelligent Vision-based Navigation in Robotics

The ultimate aim of a vision-based navigation system is to be able to act as a reliable moving platform for the environment they are design for, if not for any environment. If this is achieved, it opens the door to a variety of possible applications such as household robotics, autonomous vehicles or wheelchairs, etc. A household robot and autonomous wheelchair must be able to recognise visual patterns, navigates around the environment smoothly and freely, and perform the tasks they were designed to do. This includes object retrieval (mainly for household robotics), goal directed navigation, etc.

The first intelligent mobile robot that had vision capability dated back to 1969. Shakey was constructed at Stanford Research Institute (Nilsson, 1969). It is able to distinguish objects of given sizes, shapes and colours, and interacts with them to move them to a designated position. Shakey is equipped with two stepper motors and uses the differential drive method to control its steering action, and avoid any obstacles encountered. The name Shakey is derived from its irregular and jerky motion. Shakey uses STRIPS (the Stanford Research Institute Problem Solver), a logic based problem solving system to develop navigation plan (Fikes and Nilsson, 1971). STRIPS required symbolic information from input sensors which Shakey had difficulty generating from raw data. As Hans Moravec remembers, "An entire run of Shakey could involve the robot getting into a room, finding a block, being asked to move the block over the top of the platform, pushing a wedge against the platform, rolling up the ramp, and pushing the block up. Shakey never did this as one complete sequence. It did it in several independent attempts, which each had a high probability of failure. You will be able to put together a movie that had all the pieces in it, but it was really flaky." (Crevier, 1993).

The Stanford Cart (Moravec, 1983) is a mobile robot that uses stereo vision to locate objects and plans obstacle-avoiding paths to desired destinations on the basis of an internal model derived from stereo data. The robot was controlled by an off-board computer program and its motion was determined through comparison of images over time. A complete cycle of sense-think-act process with the robot moving a meter forward takes about 10-15 minutes to complete. After moving a meter, the robot stops and begins a new sense-think-act process. This process is repeated until the robot reaches its final destination. It takes about 5 hours to complete a 20 meter route in an environment with three to four obstacles to avoid. The system exhibits a stop-and-go motion which is largely cause by the computationally expensive stereo vision task. This includes feature detection, correlation, distance estimation and localization.

2.2 Control with Intermittent Sensing

The stop-and-go problem is a problem of control with intermittent sensing. It is due to the long time required for processing the image (i.e. delayed measurement) and for planning the movement. Nowadays computer have become much faster but there is still a delay between sensing and the moment when a new control becomes effective. To overcome the stop-and-go motion, and enable the robot to exhibit a smooth continuous motion, this delay has to be handled.

Kosaka, Meng and Kak (1993) introduced FINALE-II, an improvement over their earlier system FINALE (Koaska and Kak, 1992), a vision-guided mobile robot navigation system which had to stay static for the self-localization task (i.e. capture an image and processing the captured image to reduce the uncertainty of the robot position). FINALE-II eliminates the need for the robot to remain stationary when the vision data is being processed. This reduces the duration of the robot static state to the time needed for

capturing a new image for self-localization and the time to use the vision information (updated position uncertainty) to re-estimate the current robot position. Processing of the captured image in FINALE-II is done while the robot is in motion, following its previously calculated path toward the goal. Once the self-localization task is completed, the robot motion is stopped and its current position is re-estimated retroactively based on the stored motion history. The system then re-plans a path from the newly updated position to the goal position and restarts its motion toward the goal. Self-localization is done by matching the features extracted from the images with the expected landmarks extracted from the prior model-based map, using the expected robot's position. The robot position uncertainties are then reduced with the use of a Kalman filter.

Maeyama, Ohya and Yuta (1995) proposed a non-stop outdoor navigation system using retroactive positioning data fusion, the data being calculated using increments of the robot position vector and its covariance matrix obtain by dead reckoning. In their system, the robot keeps the position and the covariance at sensing time (i.e. t_0) for correction when the processing of landmark information finishes (i.e. $t_0+n\tau$, where $n\tau$ is the time needed to process landmark information) using maximum likelihood estimation. The current position (at time $t_0+n\tau$) is then recalculated using the total increment of parameters such as location, heading and the covariance from time t_0 to the current time $t_0+n\tau$.

Larsen, Andersen and Ravn (1998) proposed a simple and computational cheap way of compensating delays based on the extrapolation of the measurement to the present time using past and present estimates of the Kalman filter and calculating an optimum gain for this extrapolated measurement. The proposed method is a solution to the problem of designing discrete-time Kalman filters for systems where some results of measurements are delayed.

All these methods (Kosaka, Meng and Kak, 1993; Maeyama, Ohya and Yuta, 1995 and Larsen, Andersen and Ravn, 1998) are essentially using the same concept, i.e. using an estimate of the delayed measurement, then applying a correction factor when this becomes available. The method proposed here is a modification of the Smith Predictor along similar lines (chapter 6).

2.3 Navigation

Navigation involves Self-localization, Map building or updating, and Path Planning. For a successful navigation, a robot must be able to localize itself within its environment, tracks its own position and use its sensor data to built an internal map or map the sensed data onto its internal prior map, which will be used for path planning, a process which searches for an obstacle-free path from the robot's initial position to the goal.

2.3.1 Self-localization

Self-localization is a process performed on the basis of the robot's sensory readings to determine the robot's actual position within its environment. In most mobile robots shaft encoders readings can be used to track the robot's position but, due to unavoidable odometry errors such as wheels slippage and drift, the error in the estimated position increases over time. Therefore, a self-localization process is necessary to correct this error and help increase the accuracy of the estimated robot's position and improve path planning. Apart from that, the self-localization process also helps in the mapping process (i.e. map updating or construct a new map), as detected obstacles relative to the robot position can be placed accurately into the model map. Hereafter are presented only some of the most interesting self-localization algorithms, as it is impossible to cover all the approaches to self-localization found in the literature.

Cox (1989) proposed a self-localization method that uses odometry and laser range sensing to sense the environment for pose estimation. The idea was to use odometry for position tracking while overcoming the shaft encoders drift by combining odometry with laser range sensing data for self-localization. This is done by matching the sensed data to the prior map.

Janet, Gutierrez-Osuna, Chase, White and Luo (1995) proposed the use of a self-organizing Kohonen neural network based on a process similar to optical character recognition by assuming that the mapped sonar data forms a pattern unique to that room. The aim is to determine in which room the robot is on the basis of sensory data. The disadvantage of this system is that it only works in a static environment with no additional furniture or rearrangement of existing furniture, as this will change the characteristic signature of that room.

Giuffrida, Massucco, Morasso, Vercelli and Zaccaria (1995) proposed an active localization system that uses triangulation-based reference guidance (i.e. active beacons are distributed over the operating area and an onboard rotating unit is used to pick up the signal) and dead reckoning for self-localization.

Atiya and Hager (1993) proposed a real-time localization method based on visual landmarks. The idea of this approach is to recognise in the image those entities that stay invariant with respect to the position and orientation of the robot as it moves around its environment, i.e. landmarks (DeSouza and Kak, 2002), and determine their correspondence within a stored map to compute the location of the robot. A set-based algorithm is used for solving the matching problem and computing the location of a mobile robot in typical indoor environments. Interestingly, the set-based algorithm defines the error in position as the dimension of the overlapping areas of the tolerance zones around

the positions given by individual sensory measurements, instead of making assumptions based on distributions.

Jensfelt and Kristensen (1999) proposed an active global localization method using multiple hypothesis tracking. The algorithm is based on Bayesian probability theory and multiple hypothesis tracking using Kalman filtering of Gaussian pose hypotheses. The algorithm first produces pose hypotheses based on features extracted from the sensor data. Then, by making more observations of features in the environment, additional support is given to a subset of the pose hypotheses. The idea is that the hypothesis corresponding to the robot true position will gain most evidence and will be selected as the robot's position. In this approach, the robot is initially taught by interactively leading the robot through the environment while having the robot actively extracting features from its sensory data and building a world model. This system was designed to handle incomplete and partly incorrect world model. According to the authors, when their global localization failed during the experiment, it was mostly because their exploration strategy had not been able to guide the robot to points where an essential feature could be seen, or that the robot got stuck while pursuing a wrong hypothesis.

Kosaka and Kak (1992) proposed a self-localization algorithm for their system (Finale system), but the algorithm is implemented in such a way that it's only activated whenever the variances associated with the positional parameters exceed a certain predetermined threshold. Ohya, Kosaka and Kak (1998) adopt the Finale system self-localization algorithm but in their system, the self-localization algorithm is carried out on a continuous basis. The self-localization algorithm begins by generating an expectation image based on the best estimate of the robot's current position. The edges extracted from the expectation image are then compared with the edges extracted from the camera image

to find a match through an extended Kalman filter. The extended Kalman filter then produces updated values for the location and the orientation of the robot.

The approach of vision-based self-localization used in thesis involves determining "*what is being observed and where it is observed from*" (Atiya and Hager, 1993). A similar assumption to Cox (1989), Kosaka and Kak (1992) and Ohya, Kosaka and Kak (1998) was used, i.e. there is only a small difference between the expected view and the actual one. Therefore it is reasonable to attempt to match an edge found by sensors with the nearest edge in the map. The main difference with Cox (1989), Kosaka and Kak (1992) and Ohya, Kosaka and Kak (1998) is that the used edge detector can also determine the edge's orientation. This enables direct calculation of the difference between the estimated orientation and the actual orientation.

2.3.2 Map Building

Two of the most widely used mobile robot mapping concepts are known as the metric approach and the topological approach.

In the metric approach, the robot's environment is represented in an absolute reference frame and numerical coordinates define where the objects are in space (Dudek and Jenkin, 2000). The most used metric approach was originally proposed by Moravec and Elfes (1985) which is known as the occupancy/certainty grid. The occupancy grid consists of cells where each cell represents an area of the environment. Each cell in the grid contains a certainty value representing how confident one is that the cell is being occupied by an obstacle. The certainty value is calculated based on sensor readings. The initial aim of the invention of occupancy grid was to handle sonar data with ambiguous angular positions. Occupancy grid approaches have the advantage of being easy to construct, to represent and maintain even in large scale environment (Buhmann, Burgard,

Cremers, Fox, Hofmann, Schneider, Strikos and Thrun, 1995; Thrun and Bucken, 1996). Computation of an obstacle-free path to the goal is made possible by searching through obstacle-free cells within the grid. This map also allows the robot's position to be tracked accurately using information obtained from its sensory feedback and enables the system to overcome any dislocation problem due to different positions with similar sensory reading (Giuffrida, Massucco, Morasso, Vercelli and Zaccaria, 1995; Thrun, and Bucken, 1996; Thrun, 1998; Thrun, Gutmann, Fox, Burgard and Kuipers, 1998; Jensfelt, 2001).

In the topological approach, topological graphs are used to represents the environment. This is done by identifying and linking distinctive places and paths in the environment. In the graph-like representation, each node represents a distinctive place identified by unique sensory readings and the connecting arcs between two nodes represent the existence of a path between the two corresponding places. Thus the exact metric relationship between the distinctive places and paths is not needed for the map building process. The topological map was initially proposed by Kuipers and Byun (1991) for robot exploration, mapping and navigation in large-scale spatial environments, where a large-scale spatial environment is define in their paper as an environment with a spatial structure that is at a significantly larger scale than the sensory horizon of the observer. Ko, Seneviratne and Earles (1994) proposed a method that uses the extended triangular algorithm for partitioning free space into triangular cells for building a topological graph known as the triangulation graph. In the triangulation graph representation, each node is representing a triangular cell, and the connectors are used to represent the edges between cells.

The topological approach permits efficient planning and has low space complexity as its resolution depends only on the complexity of the environment. Accurate

determination of the robot's position is not needed as localization with the topological approach only requires finding at which node the robot is located.

However both approaches have their disadvantages, the metric approach is suffering from computational complexity (i.e. due to the high resolution grid map) and the need for accurate determination of the robot's position. As for the topological approach, localization can be difficult if there is more than one node with similar sensory readings. Note that the sensory reading is also sensitive to the point of measurement which therefore has an impact on the recognition of places. Thus building and maintaining of topological maps can be difficult since sensory information is ambiguous.

Thus, Thrun and Bucken (1996) suggest that by integrating both the grid-based and the topological approaches, they gain the best of both approaches: accuracy/consistency and efficiency. Their proposal was first to build a grid-based map, because it is easy to build, represent and maintain. The grid-based map will then enable the robot's position to be tracked accurately. Once the grid-based map is completed, it is used to build the topological map, therefore overcoming the problem of ambiguous sensory information. In their method, they employed an artificial neural network to interpret the sensory measurements of the environment and map into probabilities of the occupancy grid map. Bayes' rule was used to integrate multiple interpretations of the sensory measurements over time. The topological map is then built based on this occupancy grid, which is done by splitting the occupancy grid into coherent regions, separated by critical lines, where critical lines correspond to narrow passages such as doorways. This partitioned map is then transformed into a topological map where each region is represented by a node while the critical line is represented by an arc that connects the two nodes. The newly produced topological map is greatly reduced in resolution compared to the occupancy grid and enabled fast planning and problem solving.

Tomatis, Nourbakhsh and Siegwart (2001) also proposed to integrate both metric and topological approaches for mapping to gain from the benefits of both approach in their simultaneous localization and map building (SLAM) process. In contrast to the approach of Thrun and Bucken (1996), both the topological and metric maps are built simultaneously. Tomatis, Nourbakhsh and Siegwart (2001) use a topological graph to represent a global map (i.e. rooms in a building that are connected to a hallway) with each node (representing a room) being defined by a metric model. The metric model then contains detailed information about the room such as detected obstacles.

In this thesis, a metric approach is used, as the robot resides in a single room of known dimensions. The only unknowns to be determined from sensory data are the position of the robot and the position of obstacles (chapter 4). The metric approach is well suited for the grid-based planning method explored in chapter 5.

2.3.3 Path Planning

The planning of an optimal collision-free path in high-dimensional configuration spaces or in dynamic environments can be a computation intensive process unsuitable for real-time implementation on a robot.

Faster, but appropriate, path planning through the potential field method for obstacle avoidance was suggested by Andrews and Hogan (1983), Krogh (1984), and Khatib (1985) based on the idea of imaginary forces acting on the robot. In this approach, the robot experiences repulsive and attractive forces from obstacles and the goal respectively. The idea was to use repulsive forces to push the robot away from obstacles while using the attractive force to attract the robot toward the goal. The resultant force which is the sum of all the repulsive and attractive forces is used to determine the direction

of motion and the speed of navigation. The resulting obstacle-free path is not optimal as the robot tends to keep a maximum distance from obstacles. Murray (1997) proposed that by constraining the repulsive force within a fixed boundary, an optimal obstacle-free path can be produced. This however does not prevent the robot from being trapped in local minimum (i.e. a valley in the potential field that has only one way out and that is the way the robot came in).

Boreinstein and Koren (1989) proposed a new real-time obstacle avoidance approach know as the Virtual Force Field (VFF). This approach employed certainty (occupancy) grids for obstacle representation, and the potential field method for navigation. Note that the potential field algorithm is only applied to the grids within the active window for path planning. The active window is a window that moves with the robot in a way such that the robot is always at the centre of the moving window. The VFF method suffers also from the local minimum problem inherent to potential field method. The authors proposed to solve the local minimum problem with a method know as the Wall-following method (WFM). Other inherent limitations of the potential field method are: no passage between closely spaced obstacles, oscillations in the presence of obstacles and oscillations in narrow passages (Boreinstein and Koren, 1991a).

A new method know as the Vector Field Histogram (VFH) was then proposed by Boreinstein and Koren (1991b) to overcome the inherent limitation and improve the VFF method. This new method uses a two-dimensional Cartesian histogram grid as a world model which is updated continuously with range data. A two-stage data-reduction process is used to determine the desired control commands for the robot. The first is to reduce the histogram grid within the active window into a one-dimensional polar histogram that contains the polar obstacle density in each direction. The second stage is to search for candidate valleys of the polar histogram. Candidate valleys are those that have an obstacle

density value that falls below a pre-set threshold value. Only the candidate valley that is closest to the target direction is selected for the process of determining the best sector within that valley. The selected best sector is then used to generate a steering command for the robot. The authors consider this method as a local path planner, therefore it is prone to trap-states (and exhibits the cyclic behaviour), especially if the local minimum is larger than the active window.

Kwon and Lee (1996) proposed to overcome the local minimum method with the use of obstacle vectors and via points. When the robot is in a trap-state, the via points algorithm produces a series of via points using a similar idea to the visibility graph method proposed by Latombe (1991) where the via points are determined from the target point to the robot current position, based on available obstacle information. Each of the via points is then used as the robot temporary target point to guide the robot out of the trap-state.

Not suffering from local minimum problem are graph-based path planning methods such as spatial graphs and visibility graph (Lozano-Perez and Wesley, 1979), Voronoi diagram (Lee and Drysdale, 1981; O'Dunlaing and Yap, 1985; Iyengar, Jorgensen, Rao, and Weisbin, 1986; Takahashi and Schilling, 1989), free way (Wilfong, 1988), cell decomposition (Vasseur, Pin, and Taylor, 1991) and triangulation graph (Ko, Seneviratne and Earles, 1994). These methods aim at representing the free space with a topological graph that then allows the use of graph searching algorithm such as the A* algorithm (Nilsson, 1982) or the Dijkstra algorithm (Lui, Choo, Lok, Leong, Lee, Poon, and Tan, 1994) for determining a shortest path from a destination to the goal.

Bugmann, Taylor and Denham (1994) proposed a neural implementation of the Laplacian path planning (Connolly, Burns and Weiss, 1990) known as the Neural-resistive grid. The Neural-resistive grid consists of a neuro-resistive grid layer and a spatial

memory layer. The spatial memory layer is used to record the position of detected obstacles, while the potential distribution of the neuro-resistive grid is calculated based on the target/goal point with respect to the detected obstacles recorded in the spatial memory layer. The advantage of this method is that it does not suffer from the local minimum problem and always ensures an existing path to be found if the neuro-resistive grid is updated a sufficient number of times. Interestingly, this method has never been applied to a real world navigation system. To investigate its usability in this application, and because of its potential advantages, the neural-resistive grid is integrated into the system design to handle the path planning task. Details of the neural-resistive grid will be described in chapter 5.

2.4 Spatial Vision

Vision sensing is considered the most powerful sensory devices that provide the richest sensory information of all the sensors used on robots to date. However the extraction of this information is not an easy task (Borenstein, Everett and Feng, 1996). Research in vision sensing had received considerable attention, especially in the field of robotics for the last twenty years. There had been considerable research in the area of obstacle detection (Molton, Se, Brady, Lee and Probert, 1988), object recognition and tracking (Kosaka, and Nakazawa, 1995), visual servoing (Allotta, Conticelli and Colombo, 1998; Koreichi, Babaci, Chaumette, Fried and Pontnau, 1998; Ricardo, Michel and Viviane, 1998) and road extraction (Onoguchi, Takeda and Watanabe, 1995) just to name a few. Many of these are combined in the field of vision-based mobile robot self-localization, map building, updating and navigation (Moravec, 1983; Atiya and Hager, 1993; Maeyama, Ohya and Yuta, 1995; Li, Nagata and Tsuji, 1995; Murray, and Jennings, 1997; Ohya, Kosaka and Kak, 1998; DeSouza and Kak, 2002; Asoh, Motomura, Asano, Hara, Hayamizu, Itou, Kurita and Matsui, 2001).

Lorigo, Brooks and Grimson (1997) developed a system that deals with unknown environments and obstacles, utilising an environment-dependent algorithm approach to obstacle detection and navigation. The vision system consists of a single-camera vision system that uses three independent vision software modules for obstacle detection. Each of the vision modules uses different criteria (based on brightness gradients, RGB colour or HSV colour features) for detection purposes. The system assumes that anything in the image that is not "ground-like" is an obstacle. Only one of these modules is given the right to command the robot at any time, based on the confidence of their output.

Ohya, Kosaka and Kak (1998) employed single-camera vision and Ultrasonic sensing for their mobile robot to perform vision-based navigation. The aim was to use the camera to capture an image of the robot's environment, extract the detected edges in the image and compare them with edges in a synthetic image of the environment produced from a 3-D environment model, assuming the robot's position to be the one generated by dead reckoning.

Moravec (1983) used single-camera stereo vision in the Stanford Cart. This is done by having the camera capturing 9 pictures as it slides in precise steps from one side to the other along a 50-cm track. Atiya and Hager (1993) also used a single camera for stereo vision. This is done by mounting the camera on a slider in such a way that the camera remains perpendicular to the slider as it travels along the slider. Stereo images are obtained by capturing the same scene with the camera located at different locations along the slider.

Murray and Jennings (1997), Murray and Little (1998) and Se, Lowe and Little (2001) employed the Triclops trinocular stereo vision camera module that has three identical wide angle cameras. Their vision system used an algorithm similar to the

multi-baseline stereo developed by Okutomi and Kanade (1993) for computing the depth maps. The authors state that the advantage of using trinocular camera over typical two cameras stereo is because the second pair of cameras (i.e. the pair of cameras that are in the vertical plane) can resolve situations that are ambiguous to the first pair (i.e. the pair of cameras that are in the horizontal plane). Earlier work by Wilcox, Gennery, Mishkin, Cooper, Lawton, Lay and Katzmann (1987) used 3 camera stereo in their Mars rover for resolving the images correspondence problems. This is done by back-triangulating into the redundant images for confirmation of a correct match.

Apart from stereo vision systems, omnidirectional vision systems have been receiving considerable attention recently. Asoh, Motomura, Asano, Hara, Hayamizu, Itou, Kurita and Matsui, (2001) employed the omnidirectional camera for its large field of view which lets many landmarks be simultaneously present in the scene and leads to more accurate localization. Vlassis, Motomura, Hara, Asoh and Matsui (2001) used an omnidirectional vision system for environment modelling and navigation.

In this thesis, a single camera is used and distance information is extracted by projecting on the ground plane the edges of the navigable space detected by specially developed software filters. Details on the vision system are found in chapter 4.

Chapter 3

Experimental Setup

This chapter discusses the experimental setup which was designed to achieve the aims of the research. This research was fully conducted in the Robotics Laboratory of the School of Computing at the University of Plymouth. The experimental setup consists of a vision-based navigation system and a small scale environment. The vision-based navigation system was programmed to use its camera to guide the robot's navigation within its environment toward the goal while avoiding any detected obstacle. The vision-based navigation system consists of two sub-systems, the computer system and the mobile robot. The task of the computer system is to act as a remote brain for the mobile robot to help it navigate safely within its environment.

Section 3.1 describes the details of the mobile robot which is equipped with a monochrome video camera, a video sender for transferring video data, two servo motors with a servo controller module for controlling the viewing direction of the video camera and a wireless serial transceiver for communication with the computer system.

Section 3.2 describes the computer system which consists of a computer running *Cortex-Pro* - a neural network programming package. The computer, a 200MHz PC with a framegrabber is connected to a video receiver and a wireless serial transceiver.

Section 3.3 describes the robot's environment.

3.1 The Robotic System

3.1.1 The Modified Rug Warrior Robot

The robot used in this project is based on the commercially available Rug Warrior robot (Figure 3.1), developed by researchers from the Artificial Intelligence Lab at the Massachusetts Institute of Technology (Jones and Flynn, 1993). The Rug Warrior is delivered with various sensors (two shaft encoders, three bumper switches, two infrared detectors, a microphone and two photocells), a Motorola MC68HC11A1 microcontroller and its microcontroller circuit board equipped with 32 kilobytes of on-board RAM and some free digital and analogue input/output ports for additional sensors and modules. The microprocessor is programmed from a host computer. The programs are written in C (using the Interactive C programming environment) and downloaded to the robot via the host's serial line. This allows the robot to operate autonomously under the control of its onboard microprocessor. The technical specifications of the robot are listed in Table 3.1.

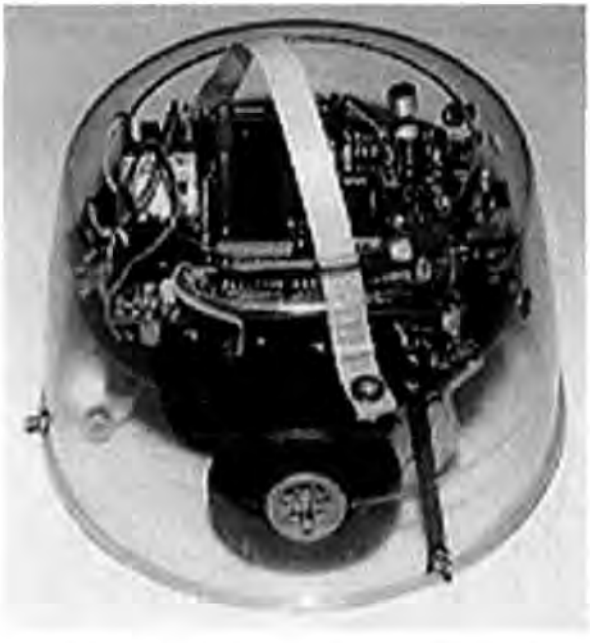


Figure 3.1: The commercial Rug Warrior robot.

Component	Typical	Min	Max	Units
Logic Supply	5.0	4.6	7.0	Volts
Motor Supply	5.0	4.0	9.0	Volts
Microprocessor Clock Freq.	2.0			MHz
Serial Line Speed	9600			Baud
IR OSC. Freq.	40	38	42	KHz
Obstacle Detection Range	38.1			Cm
Robot Speed	20.4216			Cm/sec
Encoder Clicks Per Rotation	16			Units
Robot Weight (without batteries)	0.7654365			Kg
Motor Current (each motor)			1	Amp
Wheel Diameter	6.35			Cm
Robot Diameter	18.542			Cm
Robot Height	12.065			Cm

Table 3.1: The commercial Rug Warrior robot's technical specifications.

The key to the adoption of the Rug Warrior robot as the navigation base lies in its design that enables future expansion. For the purpose of vision-based navigation, work on this thesis started with fitting the commercial available Rug Warrior robot with a VISION VM5400S camera module, a UT-66 model wireless video sender module, two HS-80 micro servo motors with a servo controller module and a wireless serial transceivers module (built by the University of Plymouth Technical Services. Additional details of these modules and their usages are discussed below.

3.1.2 Video Camera

The aim of this research is to develop a vision-based navigation system that uses computer vision to detect obstacles and searches for obstacle free path toward the goal. For that reason, a VISION VM5400S camera module was mounted on the robot and was used as the robot's visual sensor. This monochrome camera has a resolution of 240 by 387 pixels. Its small and lightweight characteristics enable it to be moved around (in pan-and-tilt motion) with the help of two servos (section 3.1.4). This allows the camera to scan its surrounding without the need to move the robot, although this feature was eventually not used. The scanned visual data from the camera are then sent to the host

computer (section 3.2) for image processing and analyses aimed at floor and obstacles detection. These results are later used for mapping and path planning. This enables the robot to interact with its surroundings without the need of additional sensors.

3.1.3 The Video Sender

The UT-66 wireless video sender module is used to transmit live video signals from the camera on-board the robot to the remote brain. The video signals are received by the computer's receiver (i.e. a video player with an antenna), which then feeds the video signals to the video capture card mounted in the computer. These live video signals are digitized by the video capture card and undergo image processing. The video sender module mounted on the robot can be seen in figure 3.2.

3.1.4 Micro Fast Servo

Two micro fast servos model HS-80 Micro from Hitec are used to provide the video camera with pan-and-tilt motion. This allows the video camera to be directed remotely. These micro fast servos were chosen because of their lightweight and high-torque characteristics. They were used to control the vertical direction of the line of sight of the camera.

3.1.5 Serial Servo Controller

The commercial available Mini SSC (Serial Servo Controller) from Scott Edwards Electronics was used in this project for the purpose of controlling the HS-80 Micro servo. This Mini SSC is able to control eight servos according to instructions received over a 2400- or 9600- baud serial connection. In this project, the Mini SSC is directly connected to the robot's RS-232 serial port and the instructions are received from the microcontroller

using 9600- baud. The Mini SCC is used to control the two servo motors discussed above. These allow the video camera to be oriented to focus on a feature of interest.

3.1.6 The Serial Transceiver (418 MHz FM –1200 Baud)

Two wireless serial transceivers built by the University of Plymouth Technical Services were used in this research as communication devices. One of the transceiver was mounted on the robot (figure 3.2) while the other was used by the remote brain. These transceivers play important roles in the communication process between the robot and the remote brain. The robot and the "remote brain" PC communicate at 1200- baud. A dedicated communication protocol was designed for this experiment.

Information such as waypoints and robot coordinates are received from the remote brain through these transceivers.

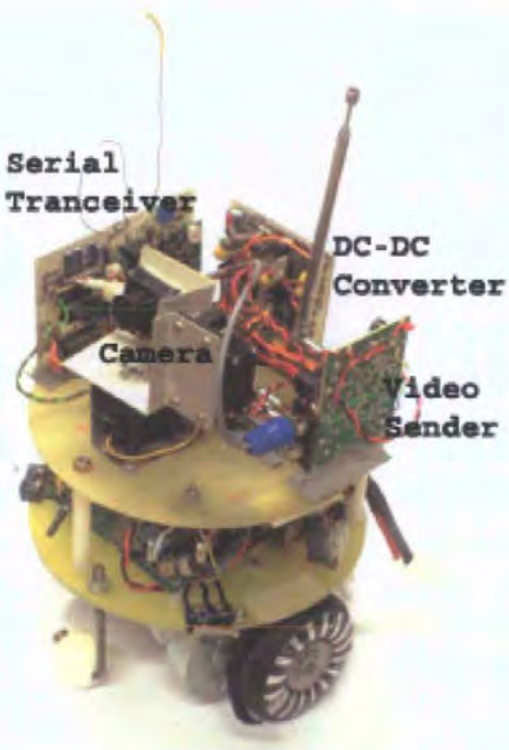


Figure 3.2: The modified Rug Warrior robot. This figure shows the added upper platform with vision and communication equipment used in this research.

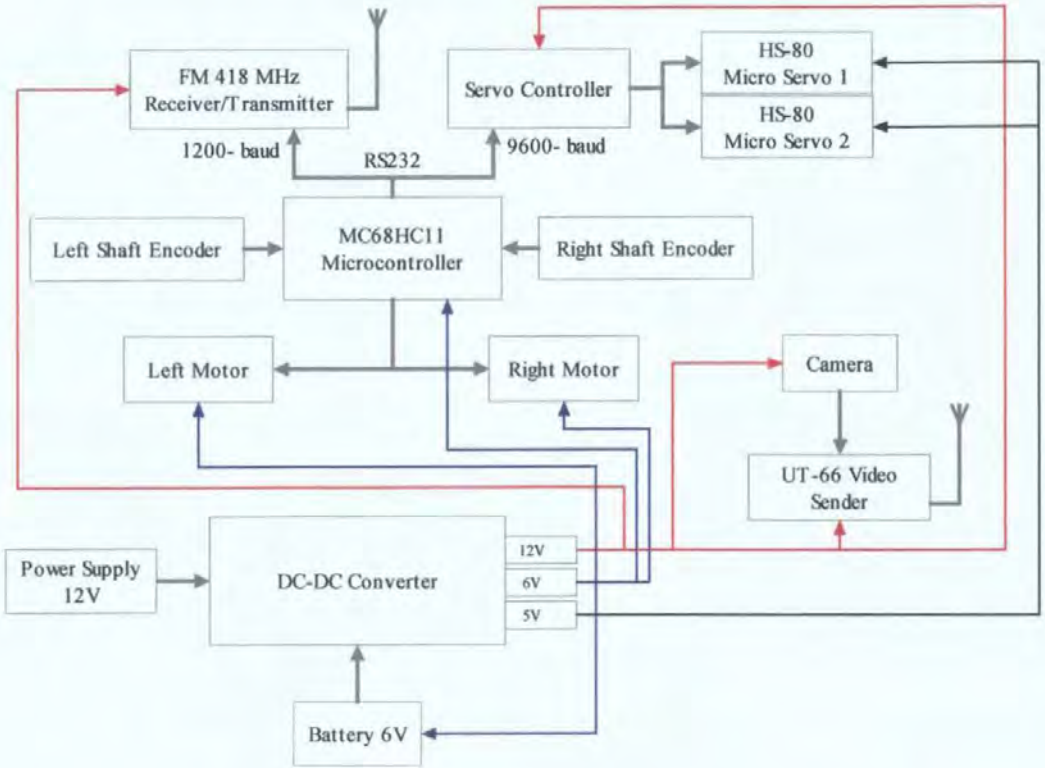


Figure 3.3: The architecture of the Robotic System.

3.2 The Computer System "Remote Brain"

The Computer System runs the user-program in a software environment called "CORTEX-Pro". The Computer System in this project acts as the remote brain for the robot. Live video signals from the robot's camera are fed through the video player via an antenna, then to the computer. The remote brain samples the appropriate live video signals of interest into an digital image. This image is then processed and analysed in order to produce an obstacle free path for the robot to navigate. This path is then transformed into waypoints before sending to the robot via the serial transceiver.

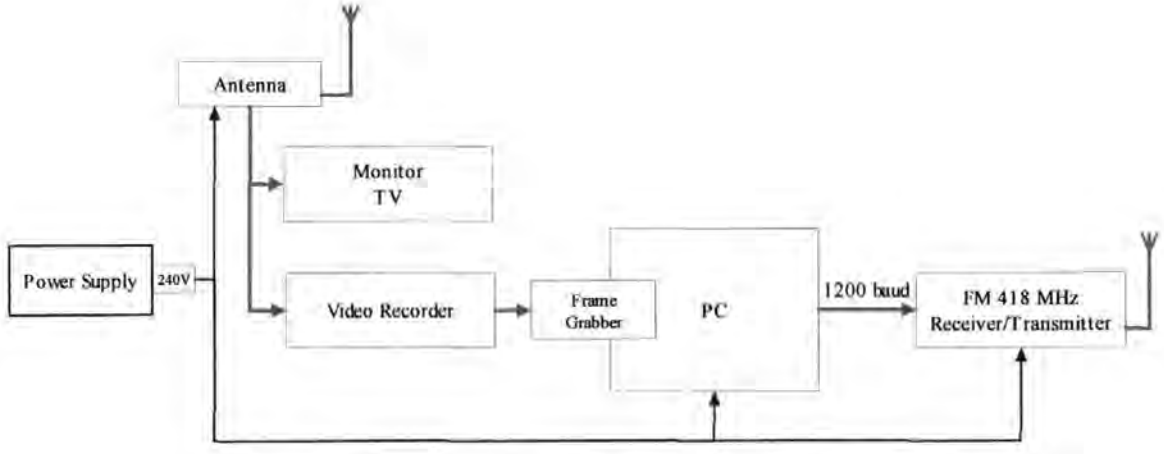


Figure 3.4: The architecture of the Computer System "Remote Brain".

3.2.1 Software "Cortex-Pro"

Cortex-Pro is a special-purpose neural networks programming environment developed at King's College, London. It is used to program the user-program that runs on the host computer. Cortex-Pro comes with built-in functions, corresponding to the needs of this research. It enables a user-program to be written in a more efficient and easy manner. The Graphics Interface of Cortex-Pro enables users to access objects/variables easily, even while the user-program is running. It can also be expanded with user-defined functions, as has been done here to add image processing capabilities.

3.2.2 WinVision Framegrabber (QUANTA)

The WinVision Framegrabber (QUANTA) located in the remote brain is used to sample the live video signal from the robot camera. The framegrabber accepts CCIR-PAL format (1 volt p-p into 75 ohms) video signals and digitizes the 320×240 pixels in the upper-left corner of the image, which are then compressed horizontally into an array of 187×240 pixels with 256 grey levels.

3.3 Environment

The robot's environment has an area of 125×89 centimetres, with white walls and a black floor. Object such as a small white block was inserted randomly in the robot's environment, acting as obstacles during the experiment (Figure 3.5). An overhead camera is mounted above the working area for recording the path of the robot. The motion tracking software was also written as part of this work, but is not described in this report.

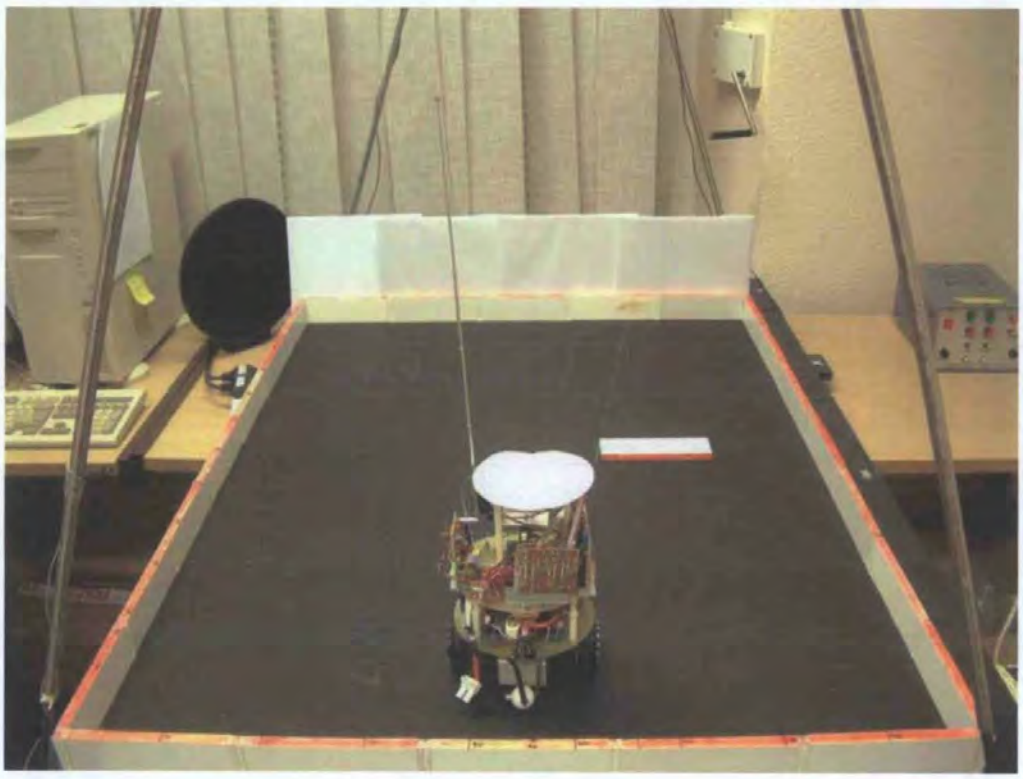


Figure 3.5: The modified Rug Warrior in its environment with the presence of an obstacle.

Chapter 4

Vision-based Obstacles Detection, Self-localization and Map Updating

Often the term "Computer Vision" is defined as a procedure that involves several processes, which consists of image acquisition, processing, classification, recognition, and to be all embracing, decision making subsequent to recognition. The aim of using a computer vision in this project is to detect the presence of obstacle and walls within the robot's environment. This allows the robot's environmental map to be updated, and supports the robot's self-localization and navigation tasks.

This chapter described vision-based obstacles detection, self-localization and map updating. The vision-based processes are shown in figure 4.1. The robot's vision system consists of a video camera, a wireless video sender, a wireless video receiver, a WinVision frame grabber and software components that processes and analyze images. The robot's video camera constantly feeds live video signals to a wireless video sender that broadcast these live video signals to the remote brain. The WinVision frame grabber onboard the remote brain samples these live video signals into a digitized image when it is needed. The sampled image is then processed by the image-processing and analyzing software module. The image-processing and analyzing software module performs the segmentation of the sampled image, dividing the sampled image into two distinct regions (walls and floor) of

similar attributes. The segmentation process prepares the sampled image for the filtering process which searches for floor edges then extracts their positions and orientations. That information is used in the self-localization module for localizing the robot in its prior map, and then in the map updating module that updates the robot's prior map with detected obstacles.

The path planner which will be described in chapter 5 can then plan a non-colliding path to the goal based on the updated robot's map.

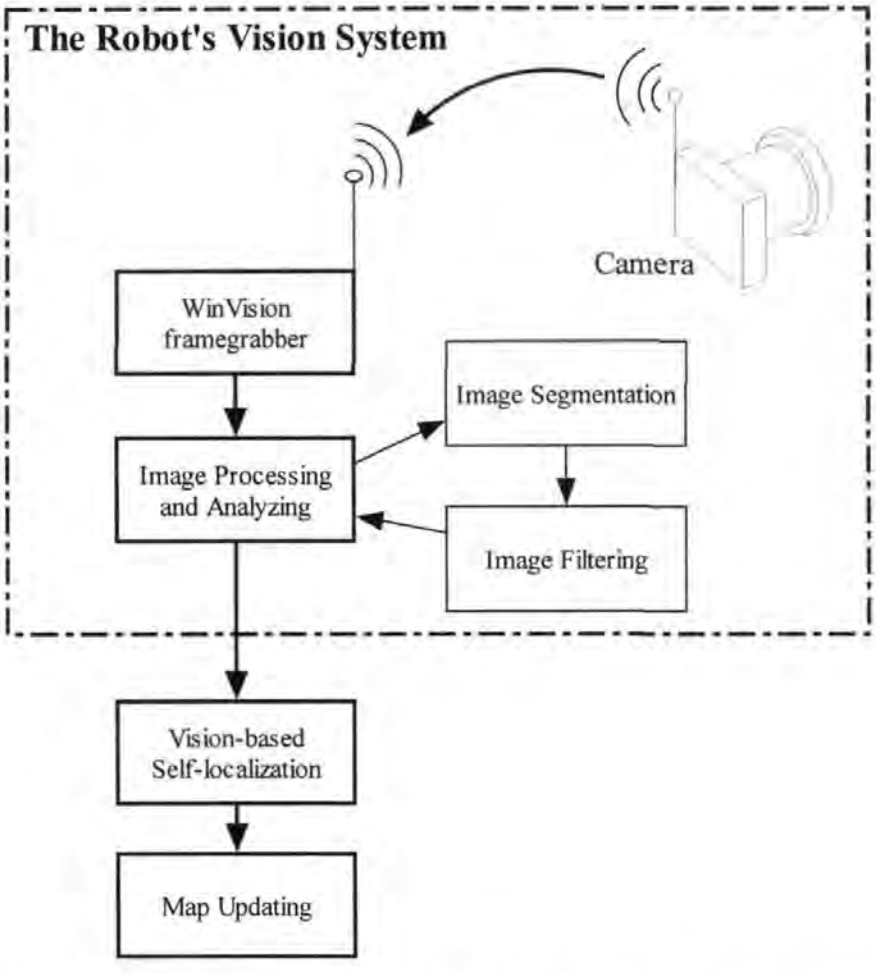


Figure 4.1: Vision-based processes for obstacles detection, self-localization and map updating.

Section 4.1 begins by discussing the fish-eye lens camera calibration process and shows how to obtain the camera lens distortion parameters that are needed to correct the

distorted image. This is important as the robot's video camera exhibits the properties of barrel distortion (fish-eye lens effect).

Section 4.2 describes the filtering process in a systematic way and introduces the floor-edges-specific filters that are used in the filtering process for detecting floor and walls edges. This section begins with discussing the image segmentation process (4.2.1) followed by the design of the floor-edges-specific filters (known as the vertical and the horizontal edge filters) used for detecting floor edges (4.2.2), and describes a new method for determining the detected edge's position and orientation (4.2.3).

Section 4.3 deals with the coordinate transformation of the detected edges and their orientations from the image coordinate system to the map coordinate system. This process involves two sub-transformations; the first is to transform the coordinates of interest from the image coordinate system to the egocentric coordinate system using projective geometry (4.3.1), while the second transforms the coordinates of interest from the egocentric coordinate system to the map coordinate system (4.3.2).

Section 4.4 discusses a vision-based self-localization algorithm that localizes the robot in its environment based on the captured image by matching the detected floor edges with those in the internal prior map. A new method is proposed for determining orientation errors.

Section 4.5 presents tests of the vision-based self-localization algorithm and discusses the test results.

Section 4.6 explains the obstacle detection and registration process that completes the robot's vision system.

In section 4.7 the test results and encountered problems are discussed.

4.1 Correction of Lens Distortion

4.1.1 Fish-eye Lens Effect

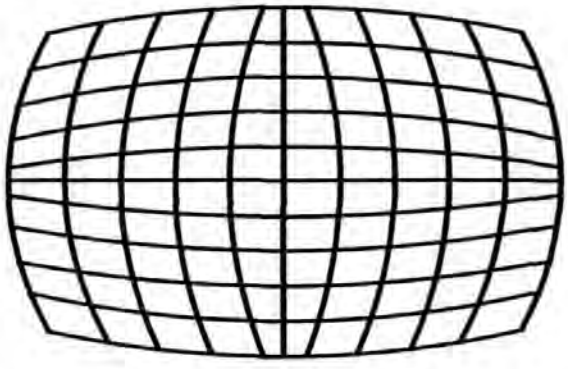


Figure 4.2: The fish-eye lens effect on a square grid.

The robot's video camera module comes with a Chinon lens that exhibits the fish-eye lens effect. The lens is used to provide the robot with a large field of view. This is particular useful when the clearance between the object and the lens is minimal, as fish-eye lens can provide a full view of the object where other lens fail. The drawback of using such a lens is that it introduces significant distortion of the captured images. This form of distortion is commonly known as the fish-eye lens effect or barrel distortion (figure 4.2). The distortion can be corrected by a procedure that involves a transformation based on the optical centre on the image plane, and the lens *distortion coefficients*. These variables are always obtained through a calibration procedure which is described in section 4.1.2.

There are many calibration methods proposed by other researchers (Beck, 1925; Miyamoto, 1964; Anderson, Alvertos and Hall, 1982; and Weng, Cohen and Herniou, 1992) for compensating the lens distortion effect. The approach used in this thesis is inspired by Williams and Becklund (1972).

4.1.2 Fish-eye Lens Effect Correction

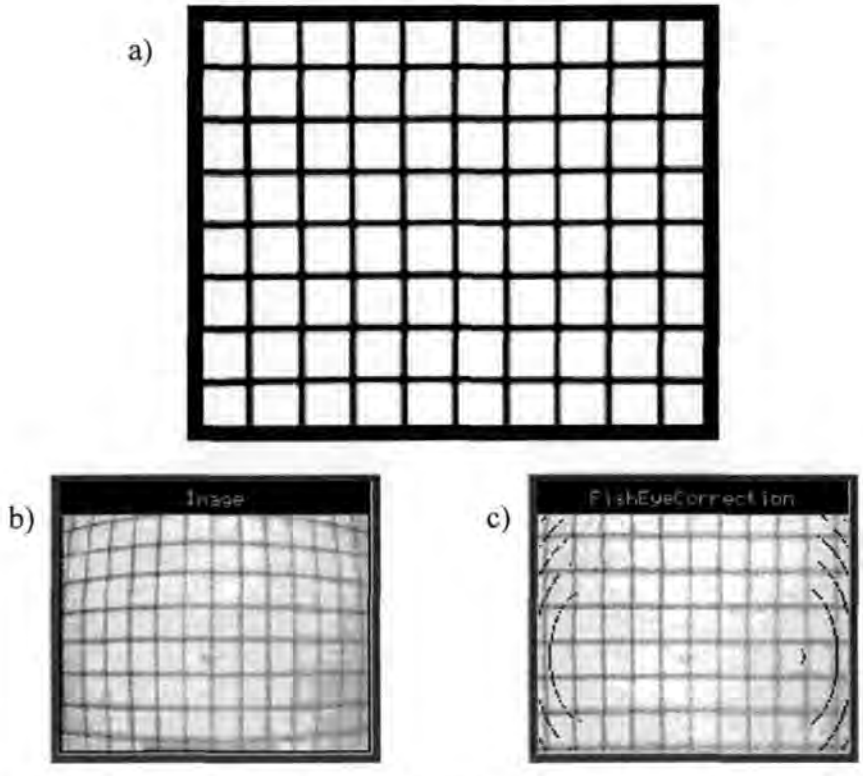


Figure 4.3: This figure illustrates the fish-eye lens effect and its correction. (a) A sample of the calibration pattern; (b) The distorted image obtained from the robot's visual sensor; and (c) The distortion-free image after applying the fish-eye lens correction sub-routine.

This section discussed the calibration procedure used for calibrating the robot's video camera. The robot's video camera is attached to a docking station with its optical axis perpendicular to the plane of the calibration pattern. The calibration pattern used is similar to the one shown in figure 4.3(a). An image of the calibration pattern taken by the robot's video camera is shown in figure 4.3(b). It is evident that the distortion results in a shifting of pixels from their original positions and creates a distorted image. This image is then used in the calibration process as the reference image for determining the location of the optical centre on the image plane and the lens distortion coefficient. Theoretically, the optical centre of the lens should always be directly perpendicular to the centre of the camera's CCD sensor array; therefore the location of the optical centre on the image plane is usually assumed to be at the centre of the image captured. But due to hardware limitations of the frame grabber, the

optical centre on the image plane is shifted from the centre of the captured image as shown in figure 4.4.

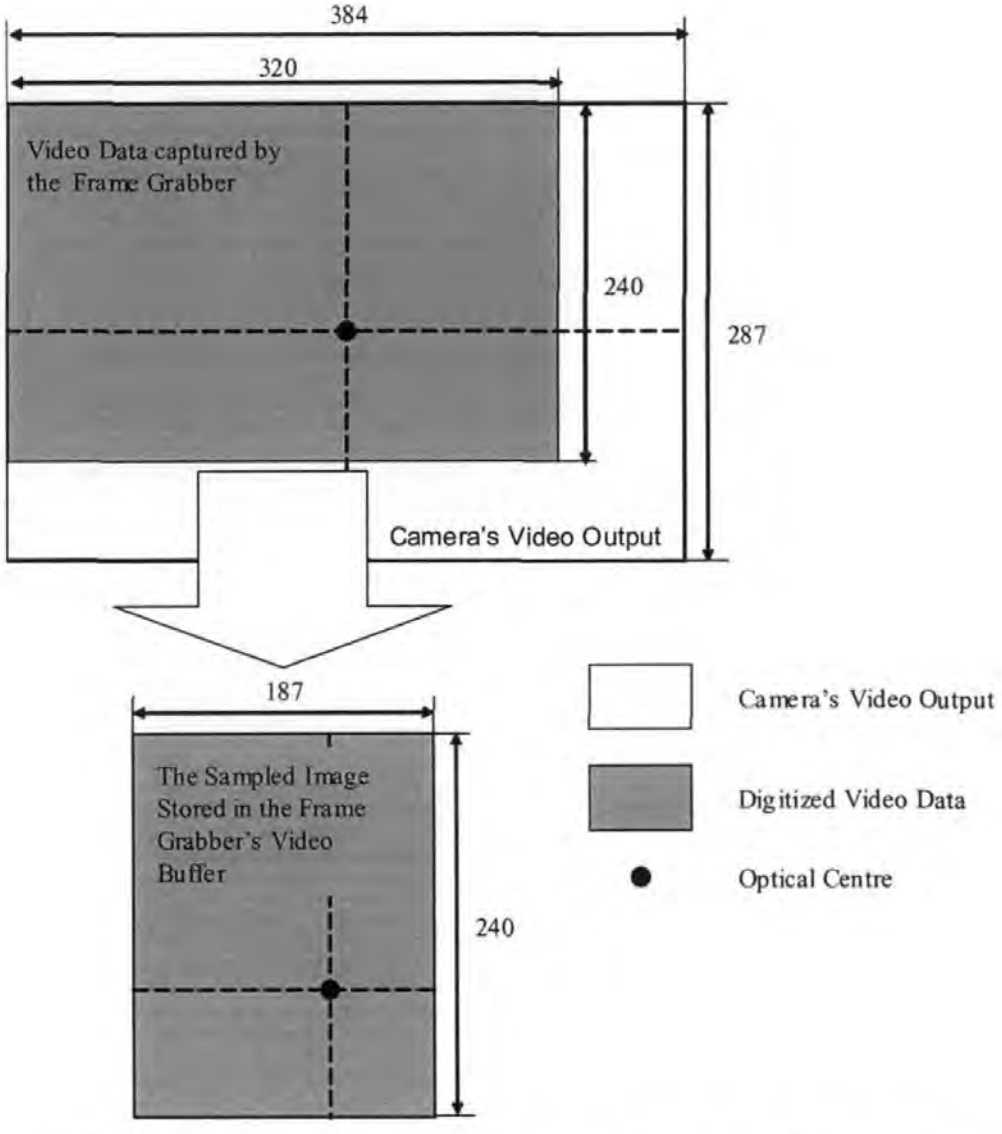


Figure 4.4: The robot's vision system image sampling process. This figure shows the sampling process from raw video signals to the final sampled image used for the vision processing. Note that the resolution changes during this process are mainly due to hardware limitations.

When capturing the reference image, it is important to make sure that the camera's field of view is directly perpendicular to the calibration pattern plane (i.e. the camera lens is parallel with the calibration pattern plane). The reason for this is to minimize any external distortion of the reference image.

Based on these assumptions and the prior knowledge of the hardware used in the vision system, the search for the location of the optical centre on the image plane can be narrowed down. The location of the optical centre on the image plane and the lens distortion coefficient can then be determined using the equations 4.1–4.5 inspired by Williams and Becklund (1972) through a trial-and-error method.

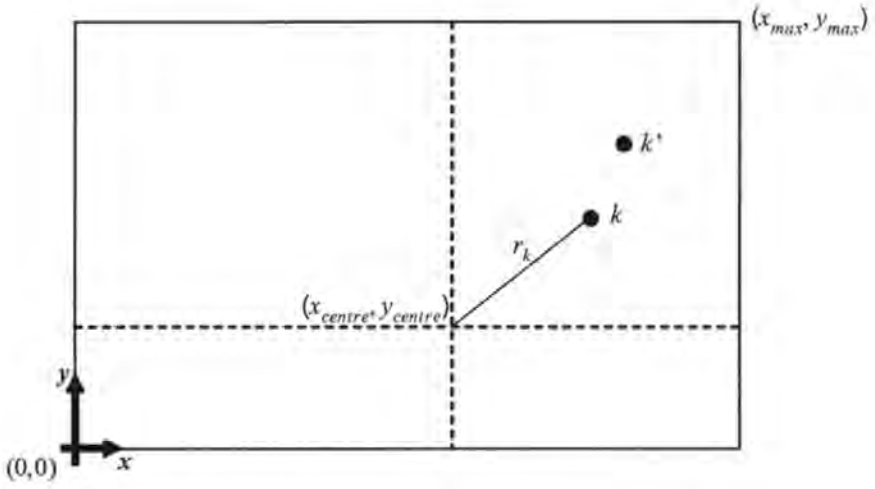


Figure 4.5: The fish-eye lens distorted image correction model. The origin of the image coordinate system used in image processing is located at the bottom left of the image. The pixel k represent the distorted pixel of interests, while k' is its new position after the fish-eye lens correction. r_k represent the distance from the optical centre on the image plane to the distorted pixel of interests, k.

These two equations shown below are used to correct the coordinates of pixels in the distorted image.

$$x_k' = [E_F \times (x_k - x_{centre})] + x_{centre} \quad (4.1)$$

$$y_k' = [E_F \times (y_k - y_{centre})] + y_{centre} \quad (4.2)$$

The function of equations 4.1 and 4.2 is to take the distorted pixel coordinate (x_k, y_k) and return their corrected position (x_k', y_k'). The correction are based on the lens optical centre on the image plane (x_{centre}, y_{centre}), as the distortion is rotationally symmetric about the lens optical centre shown in figure 4.3(b). The correction factor E_F is defined as:

$$E_F = 1 + F \times r_k^2 \quad (4.3)$$

where the value 1 is a scaling factor, r_k represent the radial distance in pixel units of the pixel k from the optical centre on the image plane and F represents the lens distortion coefficient.

The radial distance r_k with its origin located at the lens optical centre on the image plane (x_{centre}, y_{centre}) is determined using equation 4.4. The coefficient $F = 0.0000045$ and the lens optical centre on the image plane (x_{centre}, y_{centre}) = (93, 92) are obtained through a trials and errors method during the camera calibration process.

$$r_k = \sqrt{(x_k - x_{centre})^2 + (y_k - y_{centre})^2} \tag{4.4}$$

$$F = 0.0000045 \tag{4.5}$$

Once the optical centre of the lens on the image plane (x_{centre}, y_{centre}) and the optimum coefficient F are obtained, all the distorted images from the robot's video camera can be transformed to their undistorted form. The distortion-free version of the distorted image shown in figure 4.3(b) is shown in figure 4.3(c).

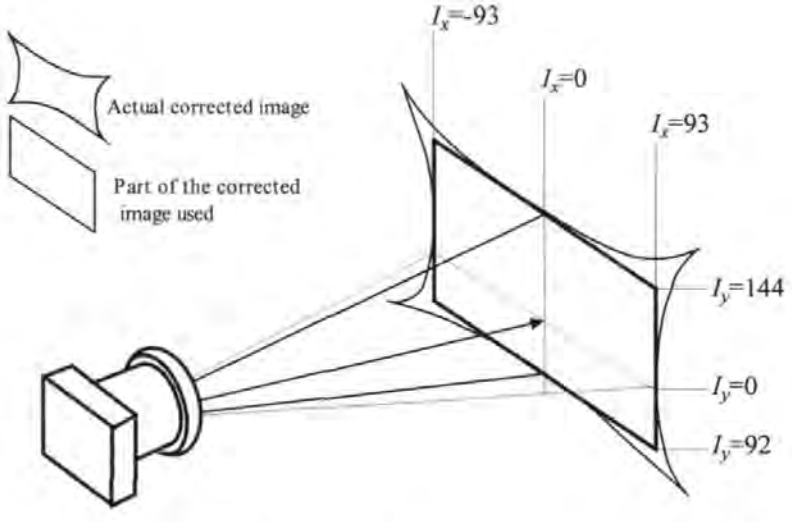


Figure 4.6: The selected portion of the distortion free image used for vision processing. The corrected distorted image captured from the video camera with resolution of 187×240 pixels is reduced to a rectangular image with resolution of 186×236 pixels. This image is then used in the image coordinate to egocentric coordinate transformation. Here I_x represents the pixel column, and I_y represents the pixel row with respect to the lens optical centre on the image plane.

4.2 Floor-specific Edge Detection

The aim of this project is to use the robot's video camera to assist the robot in navigation process such as obstacle detection, and self-localization. Thus the robot's vision system must be able to distinguish floor regions and extract useful information such as the edges positions and their orientations. A segmentation process based on automatic thresholding was implemented and will be described section 4.2.1. The result of the segmentation process provides the robot with knowledge of its navigation space but no boundary edges information (i.e. the position of the floor edges and their orientation) can be extracted. Therefore an edge detection process is proposed. Section 4.2.2 discusses the edges detection process and proposes two new floor-edges-specifics filters (i.e. the horizontal filter and the vertical filter) which are used to determine the presence of floor edges. If a floor edge is detected, the filter's outputs will be used to select the appropriate equation from the equations system, and are then applied to the selected equations to determine the orientation of the detected edge. The equations system consists of four different equations derived based on trigonometric rules. Details of the equations system are described in section 4.2.3. Section 4.2.3 also discusses how the positions of the edges are determined; the basic method is described in section 4.2.3.1 while a refined method which make used of edges orientations information is described in section 4.2.3.2.

4.2.1 Image Segmentation (Floor/non-floor)

The segmentation of the image captured by the robot into floor and non-floor regions is in general a complex process for which several methods have been proposed (Haralick and Shapiro, 1985; Pappas, 1992; Pal and Pal, 1993; Bezdek and Hall, 1993; Morel, J.-M. and Solimini, 1995 and Belongie, Carson, Greenspan and Malik, 1998).

In the present case, the image is in a monochrome grey scale. Despite the floor being painted matt black and the walls being white, in the image, these appear as dark and light shades of grey with an intensity dependent on the illumination conditions. To enable a reliable detection of floor and walls, an automatic thresholding method was designed.

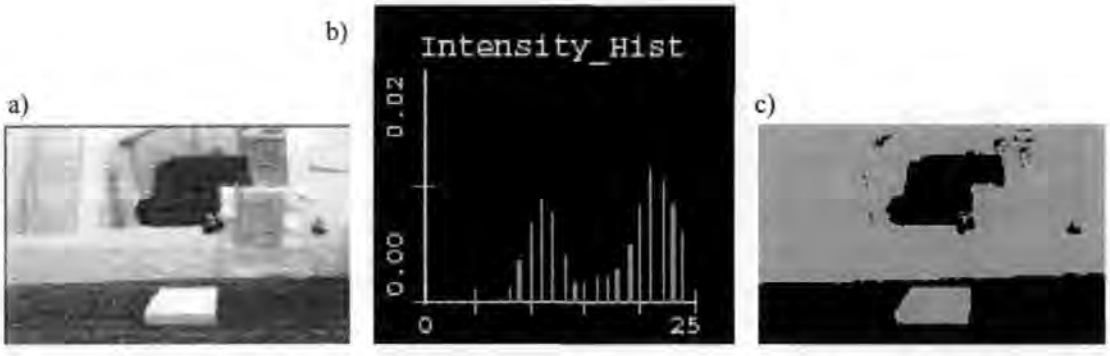


Figure 4.7: The automatic thresholding process. (a) The grey scale image, (b) the intensity histogram of the pixels in image (a), and (c) the binary image (after segmentation) with floor represented by the black colour and non-floor represented by the white colour.

The automatic thresholding sub-routine first analyses the given image (i.e. figure 4.7a) and computes the intensity histogram of that image (i.e. figure 4.7b). In the intensity histogram, it searches for two peaks, the left-most peak and the right-most peak. The sub-routine then searches for the valley between these two peaks and set its intensity value to be the threshold value. The segmentation process segments the image based on this threshold value. The intensity values that are equal or less than the threshold value are the dark regions, which are assumed to represent the floor while the intensity values that are higher than the threshold value are defined as walls or obstacles (i.e. figure 4.7c). Pixels in the region that represents the floor have their intensity values set to -1 while the other pixels which represent the walls or obstacles are set to +1. This segmentation process converts the original image into a binary image that will later used by the filtering sub-routine to detect floor edges and their orientations.

4.2.2 Vertical and Horizontal Edges Filter Design

To detect the presence of edges in the image, a filtering process is performed on the image based on the two filters shown in figure 4.8. By using the moving window method, the image is divided into 784 (28x28) sub-images. Each of these sub-images is then filtered individually by a centred horizontal and vertical filter.

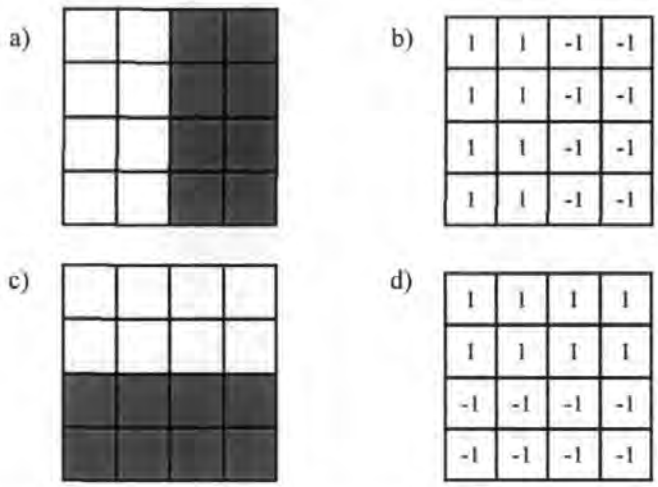


Figure 4.8: The vertical and the horizontal filters. (a) The vertical filter. (b) The matrix representation of the vertical filter; (c) Horizontal filter and (d) the matrix representation of the horizontal filter.

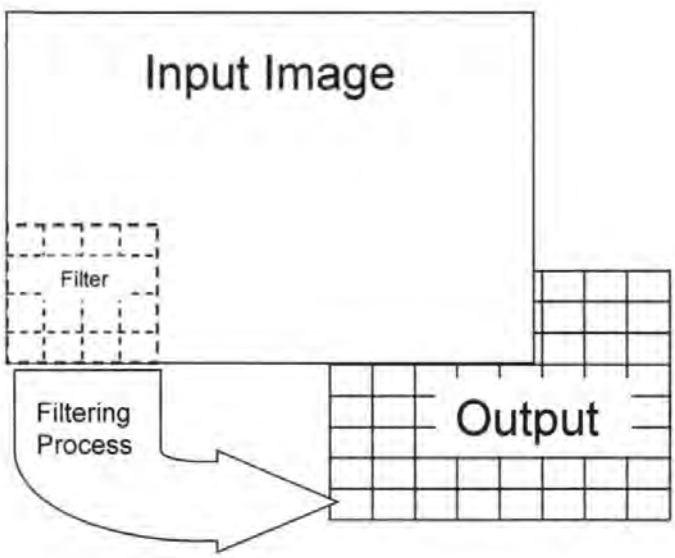


Figure 4.9: The edge filtering process for detecting edges and determining their positions and orientations. The input is a group of pixels from the input image that are passed through the filter and the output value of the filter is stored in a new grid that constitutes an output image.

The filters outputs are recorded with their coordinates. Note that these coordinate are the coordinate of the center of the sub-images within the image coordinate system. The centers of all the sub-images can be seen as the evenly space dots in figure 4.16a and 4.16c.

In each sub-image, the presence of an edge is determined using equation 4.6 and 4.7

$$A = O_H(x,y) = \frac{\sum_{j=1}^{4}\sum_{k=1}^{4}[I(m+j,n+k)F_H(j,k)]}{16} \tag{4.6}$$

$$B = O_V(x,y) = \frac{\sum_{j=1}^{4}\sum_{k=1}^{4}[I(m+j,n+k)F_V(j,k)]}{16} \tag{4.7}$$

Where: $I(m+j,n+k)$ is the input image pixel value at the coordinate$(m+j,n+k)$.
$F_V(j,k)$ is the vertical filter pixel value at coordinate(j,k). (see figure. 4.8.b)
$F_H(j,k)$ is the horizontal filter pixel value at coordinate(j,k). (see figure. 4.8.d)
(m,n) is the coordinate of the image pixel at the bottom-left of the filter.
(j,k) is the pixel coordinate relative to (m,n).
(x,y) is the output image pixel coordinate.
$A=O_H(x,y)$ is the output value of the horizontal filter.
$B=O_V(x,y)$ is the output value of the vertical filter.

If no edge is detected, the outputs of the horizontal filter A and vertical filter B will be zero. If an edge is detected, the outputs A and B are used to select an equation from the equations system shown in table 4.1. Its derivation is detailed in the next section.

O_V \ O_H	A<0	A>0
B>0	$\alpha = \frac{(90B-180A)}{(B-A)}$ (4.9)	$\alpha = \frac{90B}{(A+B)}$ (4.8)
B<0	$\alpha = \frac{(180A+270B)}{(A+B)}$ (4.10)	$\alpha = \frac{(360A-270B)}{(A-B)}$ (4.11)

Table 4.1: The equations system used to determine the orientation of the detected edge. A is the output from the horizontal filter while B is the output from the vertical filter. α is the orientation of the edge in degree.

The filters outputs A and B are then applied with the selected equation to determine the detected edge orientation α. The detected edges positions and orientations are then stored in an array that will later be used for robot self-localization and obstacles registration.

4.2.3 Calculation of the Edge Position and Orientation

The outputs A and B of the filtering process are used for the purpose of defining the orientations and positions of the detected edges in the input image. The edge's position and orientation information are important as they are needed to perform self-localization.

4.2.3.1 Edge Orientation

The edge orientation is determined using the equations system shown in table 4.1. The equations system is derived based on the trigonometric properties shown in figure 4.10. By inspecting the horizontal and vertical filters outputs A and B, the edge orientation α can be narrowed down to the quadrant it belongs to and the equation for each quadrant can be obtained.

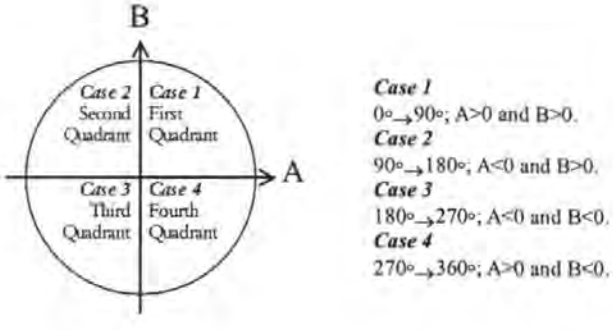


Figure 4.10: The four cases with each representing a quadrant within the circle.

If both outputs A and B are positive, α falls into the first quadrant, and is determined using equation 4.8. If A is negative while B is positive, α falls into the second quadrant, and is determined using equation 4.9. If both outputs A and B are negative, α belongs to the third quadrant, and is determined using equation 4.10. If A is positive while B is negative, α belongs to the fourth quadrant, and is determined using equation 4.11.

Hereafter, each derived equation will be tested with one of the image configurations as seen by the filter shown in figure 4.11.

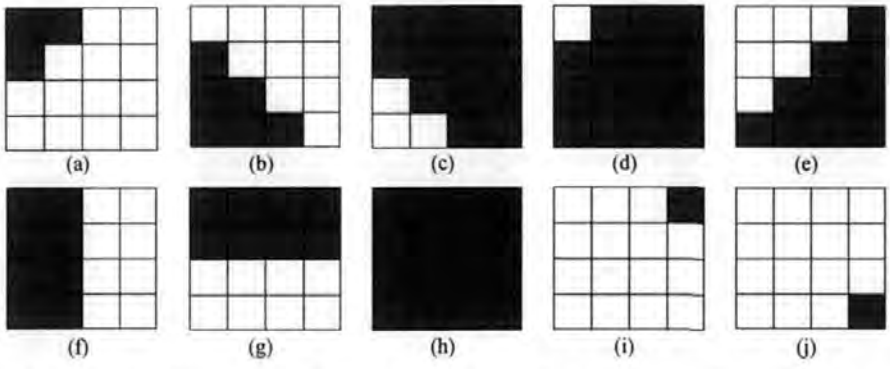


Figure 4.11: Examples of possible image configurations encountered during the filtering process. The white colour represents an obstacle or a wall, while black represents the floor.

CASE 1: A > 0 and B > 0,

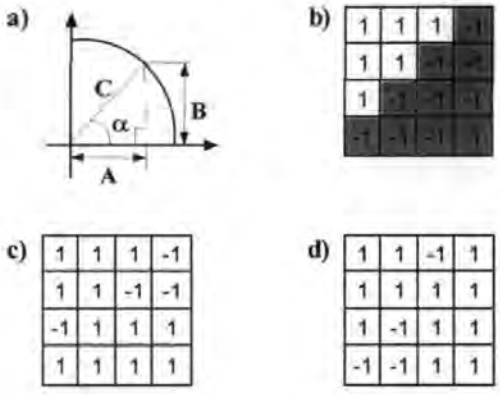


Figure 4.12: Edge filter in case 1. (a) Trigonometry drawing for case 1, (b) matrix representation of the image shown in figure 4.11e, (c) horizontal filter output and (d) vertical filter output. From (c) and (d) we obtain A = 0.5 and B = 0.5. By applying the equation system (table 4.1, equation 4.8) we obtain $\alpha = 45°$

Let

$$A = \left(1 - \frac{\alpha}{90}\right)C \qquad \text{(i)}$$

$$B = \left(\frac{\alpha}{90}\right)C \qquad \text{(ii)}$$

Proof: if $\alpha=0$ therefore A=C, B=0.

if $\alpha=90$ therefore A=0, B=C.

To determine the angle α, we transform (ii) and obtain

$$C = \frac{90B}{\alpha} \qquad \text{(iii)}$$

Substituting (iii) into (i) we obtain

$$A = (1 - \frac{\alpha}{90})\frac{90B}{\alpha}$$

$$\alpha A = (1 - \frac{\alpha}{90})90B$$

$$\alpha A = (90B - \alpha B)$$

$$\alpha A + \alpha B = 90B$$

$$\therefore \alpha = \frac{90B}{(A+B)} \qquad (4.8)$$

CASE 2: A < 0, B > 0

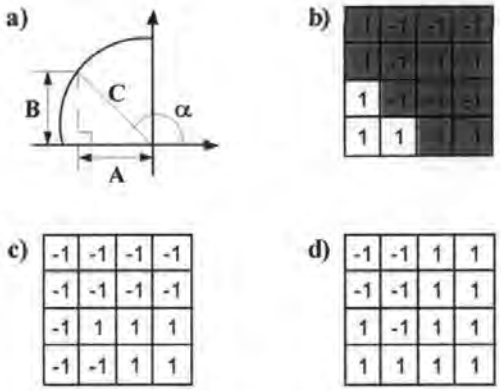


Figure 4.13: Edge filter in case 2. (a) Trigonometry drawing for case 2, (b) matrix representation of the image shown in figure 4.11c, (c) horizontal filter output and (d) vertical filter output. From (c) and (d) we obtain A = -0.375 and B = 0.375. By applying the equation system (table 4.1, equation 4.9) we obtain $\alpha = 135°$

Let

$$A = (1 - \frac{\alpha}{90})C \qquad \text{(i)}$$

$$B = (2 - \frac{\alpha}{90})C \qquad \text{(ii)}$$

Proof: if α=90 therefore A=0; B=C

if α=180 therefore A=–C; B=0

To determine the angle α, we transform (ii) and obtain

$$C = \frac{90B}{(180 - \alpha)} \qquad \text{(iii)}$$

Substituting (iii) into (i) we obtain

$$A = (1 - \frac{\alpha}{90})\frac{90B}{(180 - \alpha)}$$

$$(180 - \alpha)A = (1 - \frac{\alpha}{90})90B$$

$$180A - \alpha A = 90B - \alpha B$$

$$\alpha B - \alpha A = 90B - 180A$$

$$\alpha(B - A) = 90B - 180A$$

$$\therefore \alpha = \frac{90B - 180A}{(B - A)} \qquad (4.9)$$

CASE 3: A < 0, B < 0

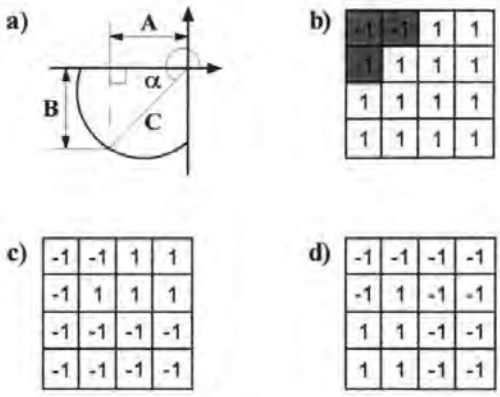


Figure 4.14: Edge filter in case 3. (a) Trigonometry drawing for case 3, (b) matrix representation of the image shown in figure 4.11a, (c) horizontal filter output and (d) vertical filter output. From (c) and (d) we obtained A = -0.375 and B = -0.375. By applying the equation system (table 4.1, equation 4.10) we obtain $\alpha = 225°$

Let

$$A = (\frac{\alpha}{90} - 3)C \tag{i}$$

$$B = (2 - \frac{\alpha}{90})C \tag{ii}$$

Proof: if α=180 therefore A=—C; B=0

if α=270 therefore A=0; B=—C

To determine the angle α, we transform (ii) and obtain

$$C = \frac{90B}{(180 - \alpha)} \tag{iii}$$

Substituting (iii) into (i) we have obtain

$$A = (\frac{\alpha}{90} - 3)\frac{90B}{(180 - \alpha)}$$

$$(180 - \alpha)A = (\frac{\alpha}{90} - 3)90B$$

$$180A - \alpha A = \alpha B - 270B$$

$$\alpha A + \alpha B = 180A + 270B$$

$$\alpha(A + B) = 180A + 270B$$

$$\therefore \alpha = \frac{180A + 270B}{A + B} \tag{4.10}$$

CASE 4: A > 0, B < 0

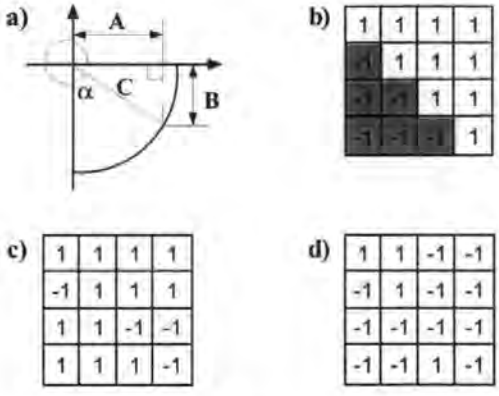


Figure 4.15: Edge filter in case 4. (a) Trigonometry drawing for case 4, (b) matrix representation of the image shown in figure 4.11b, (c) horizontal filter output and (d) vertical filter output. From (c) and (d) we obtained A = 0.5 and B = -0.5. By applying the equation system (table 4.1, equation 4.11) we obtain $\alpha = 315°$

Let

$$A = (\frac{\alpha}{90} - 3)C \qquad \text{(i)}$$

$$B = (\frac{\alpha}{90} - 4)C \qquad \text{(ii)}$$

Proof: if $\alpha = 270$ therefore A=0; B= –C

if $\alpha = 360$ therefore A=C; B=0

To determine the angle α, we transform (ii) and obtain

$$C = \frac{90B}{(\alpha - 360)} \qquad \text{(iii)}$$

Substituting (iii) into (i) we obtain

$$A = (\frac{\alpha}{90} - 3)\frac{90B}{(\alpha - 360)}$$

$$(\alpha - 360)A = (\frac{\alpha}{90} - 3)90B$$

$$(\alpha - 360)A = (\alpha B - 270B)$$

$$\alpha A - \alpha B = (360A - 270B)$$

$$\alpha(A - B) = (360A - 270B)$$

$$\therefore \alpha = \frac{(360A - 270B)}{(A - B)} \qquad (4.11)$$

In summary, these four equations are:

Case 1: $\alpha = 90B/(A+B)$ (4.8)

Case 2: $\alpha = (90B-180A)/(B-A)$ (4.9)

Case 3: $\alpha = (180A+270B)/(A+B)$ (4.10)

Case 4: $\alpha = (360A-270B)/(A-B)$ (4.11)

Where

α: the edge angle in degree

A: the output from the horizontal filter (equation 4.6)

B: the output from the vertical filter (equation 4.7)

4.2.3.2 Edge Positioning – Basic Method

Once the orientation of the edge within the filter window is obtained, the orientation information α is assigned to its coordinates (i.e. the coordinates x, y as shown in equation 4.6 and 4.7) in the image. This allows the system to know where the floor region ends and the edge's orientation. Using this information the system can perform a fairly accurate self-localization operation.

As discussed previously, the image is divided into 784 (28x28) non-overlapping sub-images (the distance between the center of two sub-images is equal to the size of a sub-images) for the filtering process. This reduces the amount of information to be processed, hence increases the computation speed.

The position of the detected edge within the sub-image is assumed to be the position of the center of that sub-image. The centers of all the sub-images can be seen as the evenly space dots in figure 4.16a and 4.16c.

The drawback of this method is that the detected edge is not assigned to its actual position, if the detected edge does not pass through the centre of the sub-image. This

inaccurate positioning leads to errors in self-localization and the presence of phantom obstacles (i.e. observed figure 4.16a and figure 4.16b)

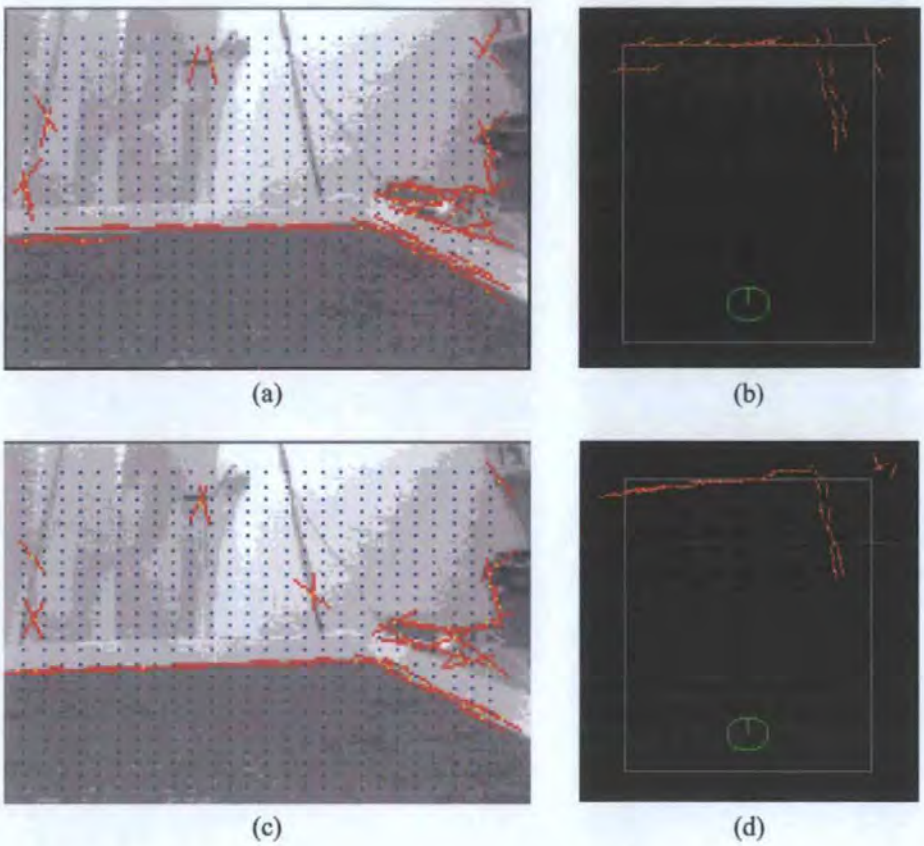


Figure 4.16: Comparison between the basic and the refined methods of edges positioning. (a) and (b) show the result of the basic method while (c) and (d) show the result of the refined method. Figure (b) shows that the detected edges at the top left corner can be mistaken for an as obstacle hence the presence of a phantom obstacle. Figure (d) show how this problem can be overcome by using the refined method.

4.2.3.3 Edge Positioning – Refined Method

In order to solve the problem of inaccurate positioning discussed above, firstly the positions of the detected edge within every sub-image has to be determined accurately, only then can its position relative to the actual image and the map be determined.

Based on the information that the floor is represented by black pixels, fraction of area covered by the floor within the sub-image can be determined. By knowing the detected edge orientation and the fraction of floor area, the exact position of the detected edge can be determined.

Since each sub-image is sampled into a 4x4 grid for the filtering process, the fraction N_{black} of black pixels occupying the grid can be determined by dividing the number of black pixels P_{black} within the grid by the size of the grid (i.e. 16). This is shown in equation 4.12.

$$N_{black} = \frac{P_{black}}{16} \tag{4.12}$$

Based on the orientation of the detected edge, the appropriate equation is chosen to calculate the actual edge position with respect to the image. There are four conditions with a total of eight equations where each condition is represented by two equations. These equations are shown below.

Condition: $0° \rightarrow 90°$

$$x = (1 - n_{black}) \times F_x + (x_s - F_x \times 0.5) \tag{4.13}$$
$$y = n_{black} \times F_y + (y_s - F_y \times 0.5) \tag{4.14}$$

Condition: $90° \rightarrow 180°$

$$x = (1 - n_{black}) \times F_x + (x_s - F_x \times 0.5) \tag{4.15}$$
$$y = (1 - n_{black}) \times F_y + (y_s - F_y \times 0.5) \tag{4.16}$$

Condition: $180° \rightarrow 270°$

$$x = n_{black} \times F_x + (x_s - F_x \times 0.5) \tag{4.17}$$
$$y = (1 - n_{black}) \times F_y + (y_s - F_y \times 0.5) \tag{4.18}$$

Condition: $270° \rightarrow 360°$

$$x = n_{black} \times F_x + (x_s - F_x \times 0.5) \tag{4.19}$$
$$y = n_{black} \times F_y + (y_s - F_y \times 0.5) \tag{4.20}$$

where F_x and F_y are the filter width (x-axis) and length (y-axis) with respect to the image, x_s and y_s are the coordinate of the centre of the sub-image with respect to the image coordinate system (i.e. the evenly space dots in figure 4.16a and 4.16c).

Figure 4.16c shows the better positioning of the edges using this refined method. By comparing figure 4.16b with 4.16c, one can see that the doubling of the edge line at the top left of figure 4.16b is eliminated in figure 4.16c. It seems that the refined method is not as effective for side walls. The reasons for that are unclear at present. They are unlikely to be a

software implementation problem as this has been checked several times. Thus the proposed refined algorithm is effective in reducing the problem of phantom obstacles (section 4.5). The refined method is used in the self-localization process described in section 4.4.

4.3 Coordinate Transformations for the Vision System

The VC5400S Camera Module used in this project is attached to the top of the robot (figure 4.17) with height h from the ground (figure 4.18). In order to build the model map, the relationship between image coordinates and the real world coordinates have to be established. This section describes the required coordinates transformations.

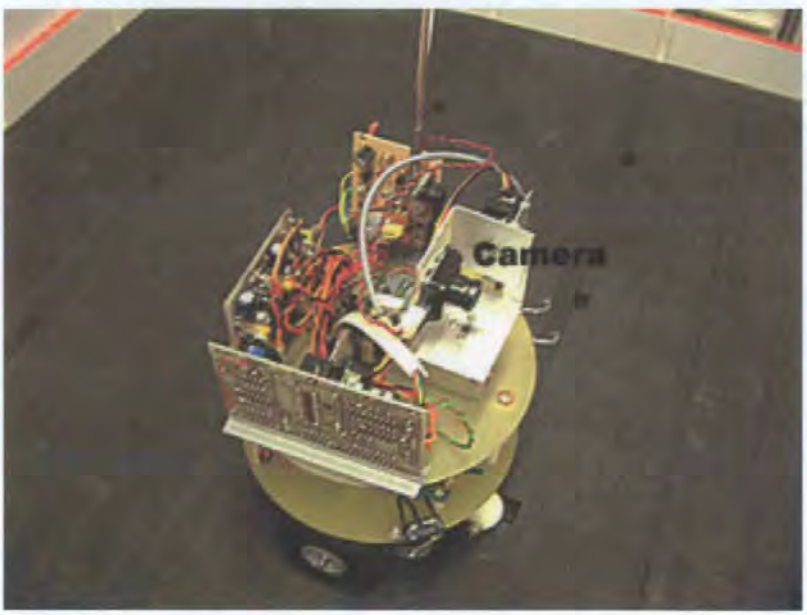


Figure 4.17: This figure shows the camera module.

4.3.1 Image Coordinate Frame to Camera Coordinate Frame Transformation

The first step in coordinate transformation is to understand the relationship between the two coordinate systems of interest (i.e. the image coordinate system and the camera coordinate system) and make a link between them. The camera coordinate system is centred at floor level vertically under the camera. It rotates horizontally with the camera (see e.g. figure 4.22). The objective is to convert the 2-D position of an object (e.g. pixel) in the

image into its position in the 3-D coordinate frame of the camera. Figure 4.18 and 4.19 shows the camera geometry maps used to determine the relationship between the image coordinate system and the camera coordinate system.

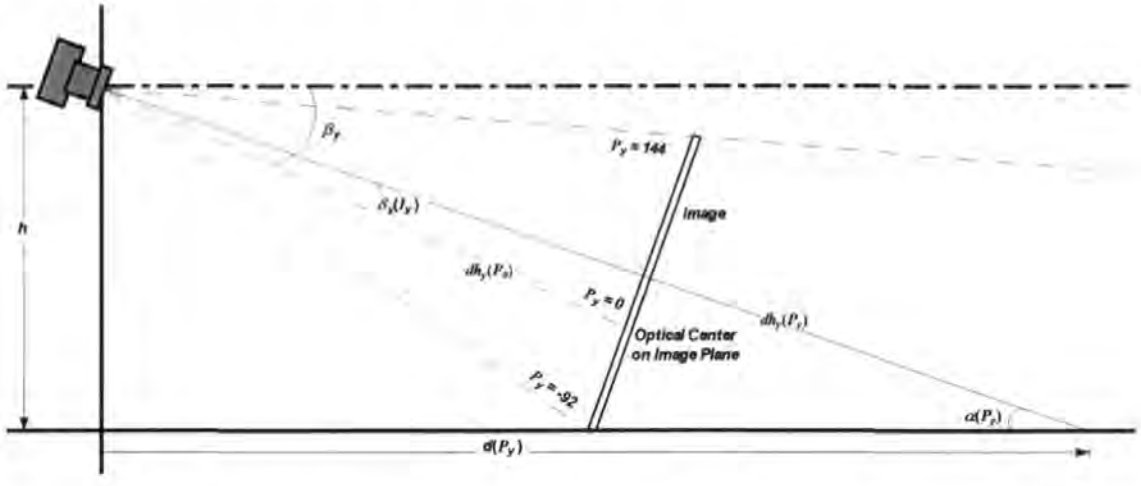


Figure 4.18: The side view of the camera geometry map used for determining the y coordinates of the detected edges. The variables shown are β: camera pitch angle, h: height of the camera, P_y: the vertical image coordinate, $dh_y(P_y)$: the imaginary line originating from the lens optical centre and passing through P_y, $\delta(P_y)$: the angle of $dh_y(P_y)$ with respect to the line originating from the lens optical centre and passing through lens optical centre on the image plane (P_y=0), $\alpha(P_y)$: the intersection angle of the imaginary line P_y and the ground plane, and $d(P_y)$: the distance from the camera to the intersection point of P_y on the floor.

Figure 4.18 shows the side view of the camera geometry map used for determining the coordinate of the pixel of interest with respect to the camera coordinate system. Based on the camera geometry map shown in figure 4.18, $d(P_y)$, or simply y_c, is the y-coordinate of the pixel of interest P (i.e. whose coordinates are (P_x, P_y) with respect to the image coordinate system based on the optical centre in the image plane) with respect to the camera coordinate and can be determined using equation 4.22, if $\alpha(P_y)$ is known. $\alpha(P_y)$ can be determined by using equation 4.21, where β is the camera pitch angle, and $\delta_y(P_y)$ is the angle of $dh_y(P_y)$ with respect to the line originating from the lens optical centre and passing through the lens optical centre on the image plane (P_y=0) as shown in figure 4.18. Note that β is negative as the camera is looking downward, and $\delta_y(P_y)$ can be positive or negative

depends on the location of the pixel of interest P (i.e. P_y is negative if it locate below the optical centre on the image plane).

$$\alpha_y(P_y) = -\left(\beta_y + \delta_y(P_y)\right) \tag{4.21}$$

$$y_c = d(P_y) = \frac{h}{\tan(\alpha_y(P_y))} \tag{4.22}$$

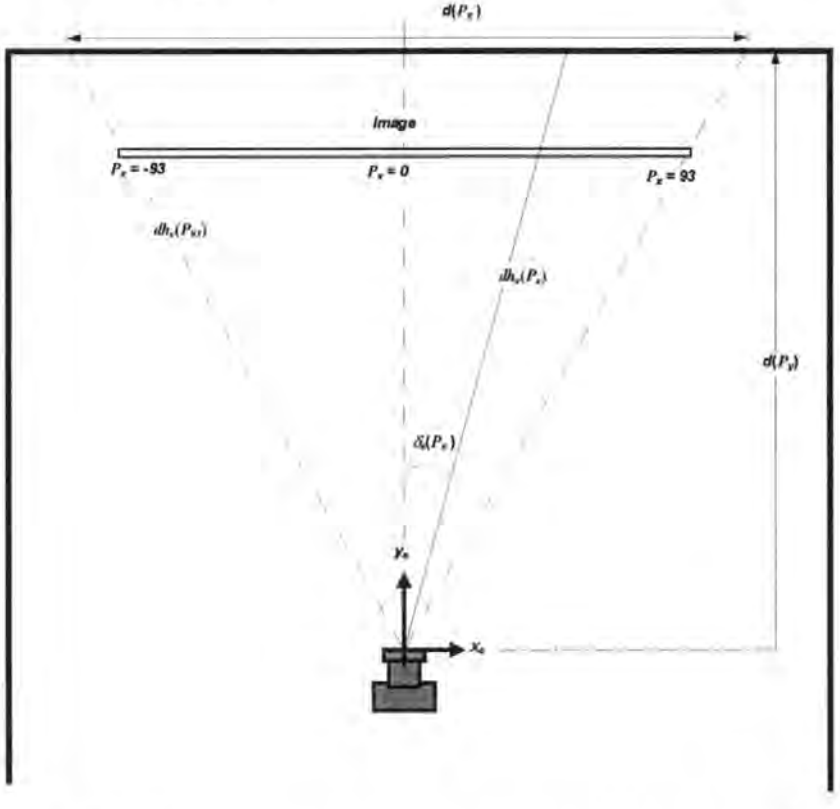


Figure 4.19: The camera geometry map used for determining the x coordinates of the detected edges. The variables defined here are I_x: the horizontal image coordinate, $dh_x(I_x)$: the imaginary line originating from the lens optical centre and passing through I_x, $\gamma(I_x)$: the angle of $dh_x(I_x)$ with respect to the line between lens optical centre and the lens optical centre on the image plane, and $d(I_x)$ is the distance between I_x and the lens optical centre on the floor plane.

Once $d(P_y)$, the distance of the pixel of interest P with respect with the camera y axis is determined, $d(P_x)$ or x_c can then be determined using equation 4.23. This equation is derived based on figure 4.19 using similar methodologies as described above.

$$x_c = d(P_x) = \frac{d(P_y)}{\tan(\delta_x)} \tag{4.23}$$

Note that $\delta_x(P_x)$ and $\delta_y(P_y)$ are needed to determined x_c and y_c respectively. To determine $\delta_x(P_x)$ and $\delta_y(P_y)$ we need to find their pixel-angle relationship.

A similar setup as described in the section 4.1.2 on Fish-Eye Effect Correction was used to make sure that the camera's optical axis is perpendicular to the calibration grid as shown in figure 4.20 to determine the pixel-angle relationship of the camera. The calibration grid is used because the distance between each line in the grid is known hence this simplifies the measuring process. Based on the calibration grid, we determine the number of pixels shifted from the optical centre to the first line of the grid and calculate its angle using equation 4.24 for the x-axis and equation 4.25 for the y-axis.

This pixel-angle data is recorded, and the whole process is repeated for the second line, the third and so on. This process was performed on all visible vertical and the horizontal lines of the grid in the image.

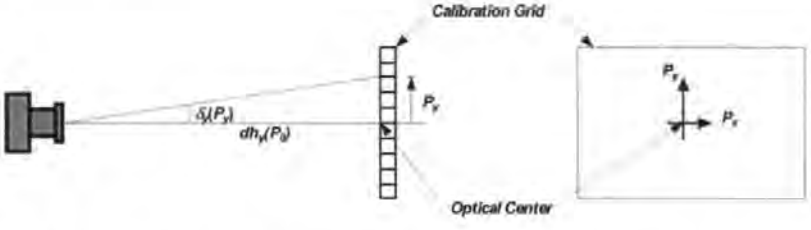


Figure 4.20: Side view of the concept diagram used to determine the angle for each line in the calibration grid.

$$\theta_{c\_x} = \tan\left(\frac{P_x}{d_c}\right) \tag{4.24}$$

$$\theta_{c\_y} = \tan\left(\frac{P_y}{d_c}\right) \tag{4.25}$$

The recorded pixel-angle data were plotted and curve-fitted to determine their relationship. This pixel-angle relationship graph is show in figure 4.21.

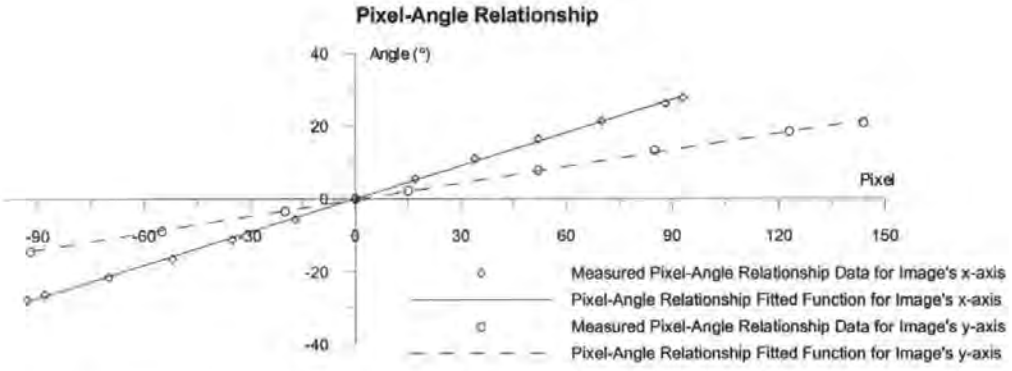


Figure 4.21: The Pixel-Angle relationship graph showing the measured data plot and their fitted function.

The pixel-angle relationship equation for the image x and y axis are shown in equation 4.26 and equation 4.27 respectively. P_x and P_y each represent the column and row number of pixel in the x and y axis with respect to the optical centre.

$$\delta_x(P_x) = 0.3022442903\,P_x + 0.02324956079 \quad (4.26)$$

$$\delta_y(P_y) = 0.1515357901P_y - 0.2358910127 \quad (4.27)$$

Once the pixel-angle relationship equations is obtained, the location of the pixel of interest relative to the camera coordinate (x_c, y_c) can be obtained using equation 4.22 and 4.23.

4.3.2 Camera Coordinate Frame to Map Coordinate Frame Transformation for Obstacles and Walls

The coordinate transformation from the Camera Coordinate frame to Map Coordinate frame plays an important role in the process of self-localization and map updating, as it allows the detected edges in the image to be mapped onto the prior map. Two transformation processes are needed to project the pixel of interest from the camera coordinate frame to the map coordinate frame. The first transformation, which will be described in section 4.3.2.1, transforms the pixel of interest from the camera coordinate frame to the robot's coordinate frame. The second transformation, which will be described

in section 4.3.2.2, transforms the pixel of interest from the robot's coordinate frame to the map coordinate frame.

The pixels of interest ($d(L_x)$, $d(L_y)$) described in previous section can then be rewritten as ($x_{c,P}$,$y_{c,P}$) which indicate the coordinates of the pixel of interest P, with respect to the camera coordinate frame c. Figure 4.22 show the relationship between each coordinate frame and helps in deriving the transformation equations shown below.

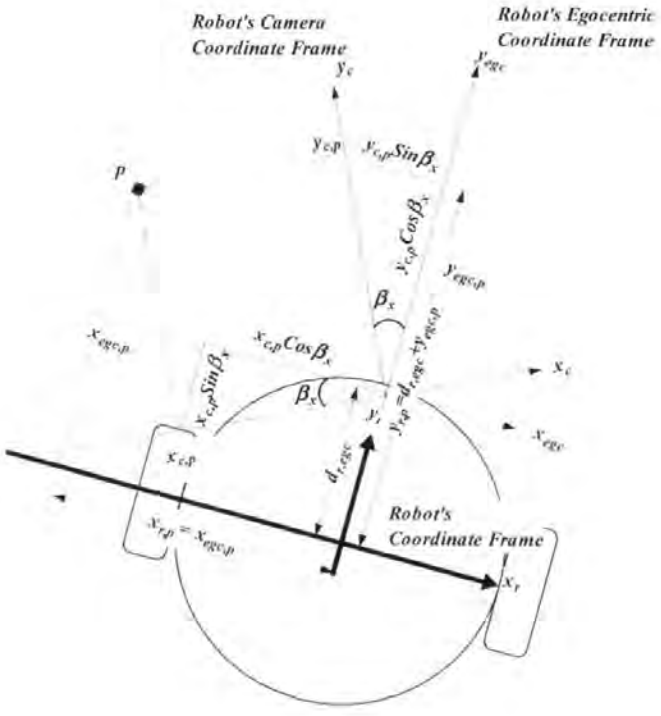


Figure 4.22: Illustration of the relationship between the camera coordinate frame, the robot's egocentric coordinate frame and the robot's coordinate frame.

4.3.2.1 Camera Coordinate Frame to Robot Coordinate Frame Transformation

The pixel point shown in figure 4.22 is an example of a pixel in an image captured by the camera. In order to place this pixel into its relative place on the map, several transformations have to be made. In this section we will concentrate on the transformation of the coordinate of interest P from the camera coordinate frame to the robot coordinate frame. That is the point $(x_{c,P}, y_{c,P})$ in the camera coordinate system transformed into a point $(x_{r,P}, y_{r,P})$ in the robot coordinate frame. As shows in figure 4.22, a combination of rotations and translations is needed. A rotation of the camera coordinate system by an angle of $\beta_{c,z}$ is needed for the transformation into the egocentric coordinate system. Therefore equations 4.28 and 4.29 are used to perform the rotation transformation with angle $\beta_{c,z}$ to bring the coordinate of point P from the camera coordinate system into the egocentric system (figure 4.23).

$$x_{egc,P} = x_{c,P}\cos\beta_{c,z} - y_{c,P}\sin\beta_{c,z} \tag{4.28}$$

$$y_{egc,P} = x_{c,P}\sin\beta_{c,z} + y_{c,P}\cos\beta_{c,z} \tag{4.29}$$

Since the egocentric coordinate system is collinear with the robot coordinate system, only a translation is needed to convert the egocentric coordinate system into the robot coordinate system. Since only an offset distance d_y along the y axis of the robot coordinate system separates the egocentric coordinate system and the robot coordinate system, we have

$$y_{r,P} = d_{r,egc} + y_{egc,P} \tag{4.30}$$

The translation equation shown above is used to transform the coordinates of point P from the egocentric system to the robot coordinate system (figure 4.23).

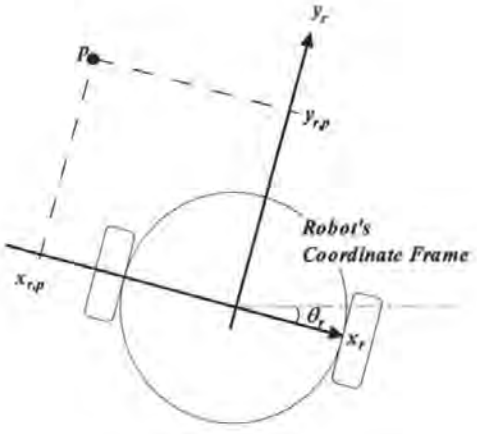


Figure 4.23: The location of pixel P in the robot coordinate system.

4.3.2.2 Robot Coordinate Frame to Map Coordinate Frame Transformation

To bring the origin of the robot coordinate systems onto the origin of the map coordinate system involves a rotation of angle θ_R that will bring the two coordinate systems parallel with each other, followed by a translation of x_r and y_r along the map x_m and y_m axes respectively. Therefore in order to transform the coordinate P from the robot coordinate system to the map coordinate system involves the same rotation and translation process that are represented in equation 4.31 and equation 4.32. These equations are derived based on the same concepts described previously:

$$x_{m,P} = x_{m,r} + x_{r,P}\cos\theta_r - y_{r,P}\sin\theta_r \tag{4.31}$$

$$y_{m,P} = y_r + x_{r,P}\sin\theta_r + y_{r,P}\cos\theta_r \tag{4.32}$$

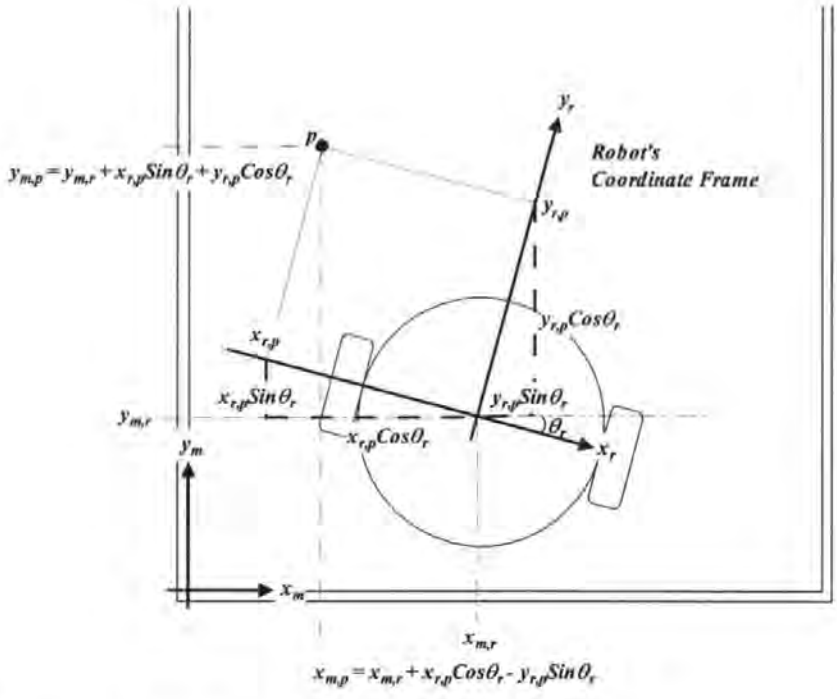


Figure 4.24: This figure illustrates the transformation of coordinate from robot coordinate system to the map coordinate system.

4.4 Vision-based Self-localization

This section describes the vision-based self-localization method used in this project. The vision-based self-localization function is used to reposition the robot on the prior map based on the image captured by the robot's camera. It is important to note that the shaft encoders localization sub-routine does not accurately estimate the robot position as the shaft encoders drift with time. Thus, vision-based self-localization is employed to overcome this problem.

The aim of the vision-based self-localization sub-routine is to determine where the robot was located when the sampled image was taken. This is done by matching the detected floor edges in the captured image with floor edges in the robot's environment (prior map) and then, by using this information, deriving the robot's pose.

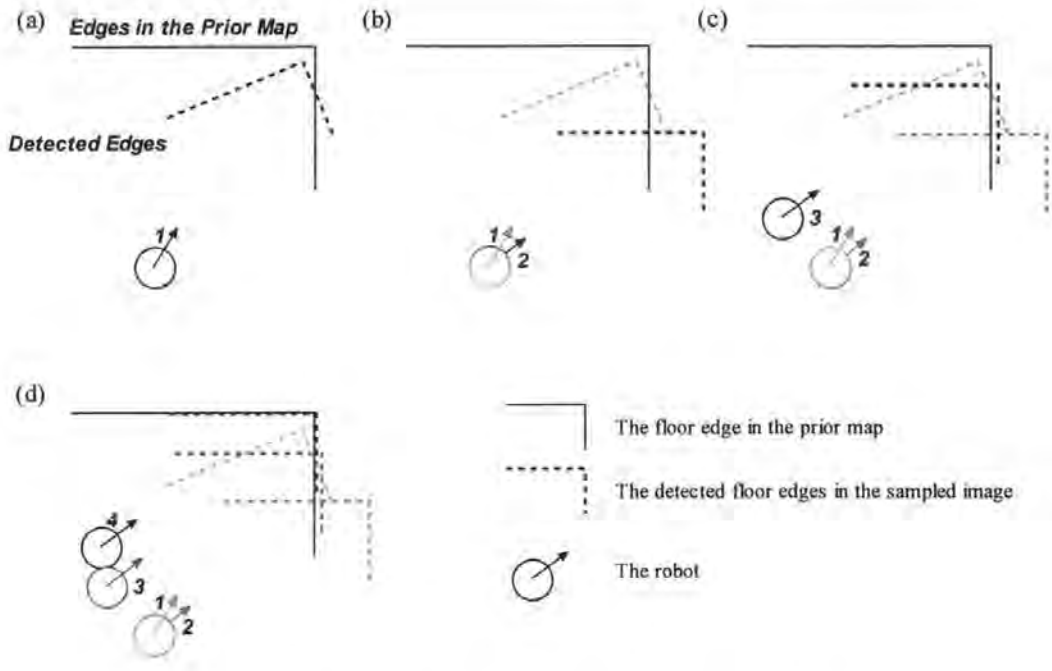


Figure 4.25: Illustration of the vision-based self-localization process. (a) Comparing the floor edges (horizontal and vertical) detected in the image with floor edges in the prior map to obtain the angular and position deviations between the detected edges and their nearest neighbour edges in the prior map. (b) Using the mean angular deviation to recalibrates the robot's orientation, and (c) the mean position deviation to recalibrate the robot's position. The whole process illustrates at a, b and c is repeated once to provide a better estimation of the robot's pose as illustrate in (d). The numbers 1, 2 and 3 in the above figures indicate the sequence of robot positions during the steps of the vision-based self-localization process while the number 4 indicates the final robot's pose.

The matching process is performed by matching the detected floor edges with the floor edges in the prior map. For each detected edge, a search is performed in its surrounding about 6cm in each direction on the prior map, to find a nearest neighbour with a similar orientation. During this process, the angular deviations of all the neighbours in that surrounding that are less than 30 degree are recorded and averaged, but only the nearest neighbour which is the closest to the detected edge is selected as the nearest neighbour and its coordinate difference is recorded for positional recalibration.

Once the search process for all the detected edges is completed, the mean angle deviation of all the potential neighbours for the detected edges is used to calibrate the robot's orientation. The (x, y) coordinate deviations for all the nearest neighbours are then used to

determine the mean (x, y) coordinate deviation to recalibrate the robot's coordinates. This completes the robot self-localization process as illustrate in figure 4.25.

Note that the vision-based self-localization sub-routine is implemented in this research by using the waypoints information to narrow down the number of possible pattern matching edges in the prior map hence reducing the aliasing problem. The proposed approach was to use 3 points along the planned path, halfway between the waypoints, and one point at the first waypoint (in case the robot has not moved). Chapter 5 describes how waypoints are defined. The self-localization algorithm is executed for each of the 4 points, and the pose with the most matches between the detected edges and the edges on the prior map then represent the best estimate of the robot's pose at the time the image is captured. Note that this method relies on the shaft-encoders to perform reliably as it assumes that the robot has followed the path within a margin of ±6cm.

Before the vision-based self-localization procedure is implemented in the navigation system, an experiment was conducted to determine its performances. This is described in section 4.5.

4.5 Self-localization Tests

4.5.1 Results

During the experiments, the robot was physically positioned in its environment at a fixed position and orientation with its video camera looking forward. The idea was to change the assumed initial position and orientation of the robot and determine if the vision-based self-localization procedure were capable of correcting this error based on the image obtained from the video camera. The experiment is divided into two parts, with the first part dealing with position shifts and the second part dealing with orientation changes.

In the first series of experiments, the robot was physically positioned at (45cm, 16cm) with an orientation of *0* degree. The initial believe of the robot's position was changed by two centimetre incrementally in either the x or y direction. Four tests were conducted where each test was performed for a specific direction of the displaced assumed position (i.e. forward, backward, left and right). For each direction, the initial believe of the robot's position was shifted by up to 8 centimeters (figure 4.26a). For each two centimetre increment, 10 trials were performed.

In the second series of experiments, again the robot was physically positioned at (45cm, 16cm) with orientation of *0* degree. The initial belief of the robot's position was not changed but instead its orientation was changed with successive increments of 1, 1, 2, 2, 2, 2, 5 and 5 degree in a clockwise and anticlockwise direction (figure 4.26b). Two tests were conducted where each test was performed for a specific direction (i.e. clockwise or anti clockwise). For each direction, at every angular increment 10 trials were performed. Figure 4.2.7 shows, as an example, the results for each of the 10 trials for shifts of 2 cm in four directions. Figure 4.28 shows the standard deviations of the position errors, as a summary of all the measurement results.

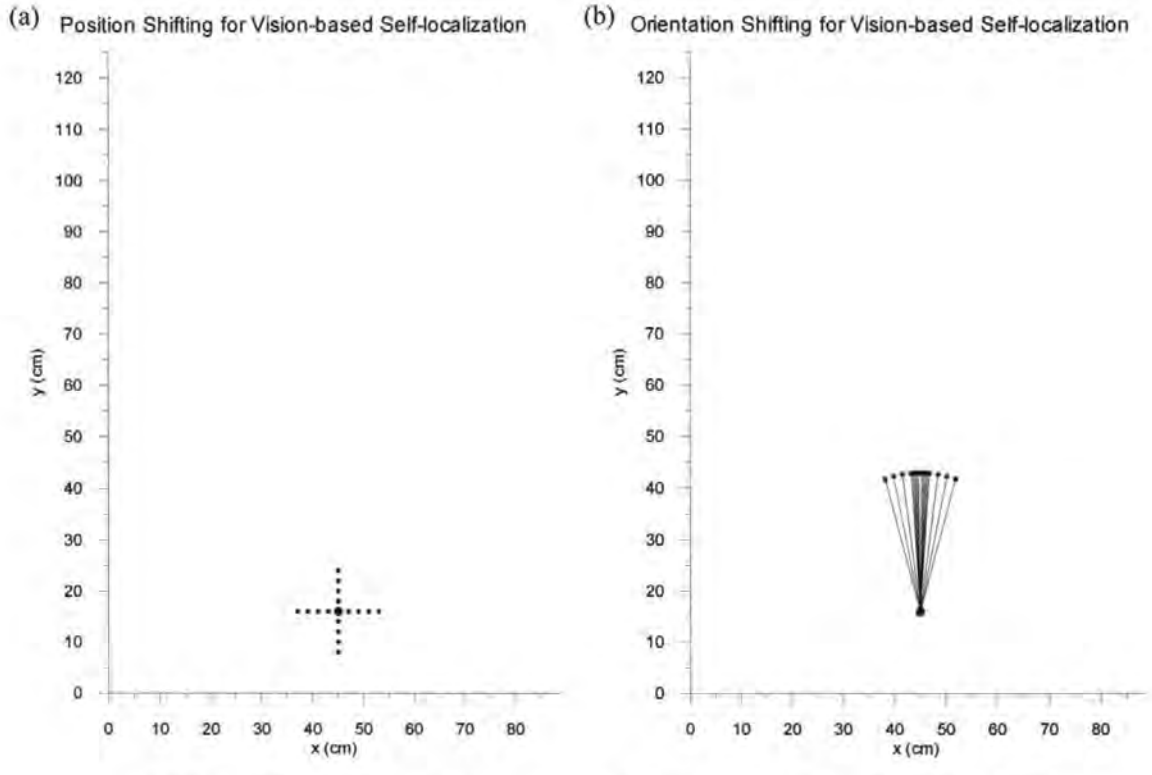


Figure 4.26: The positions and orientations used in the vision-based self-localization test. (a) The positions used. (b) The orientations used.

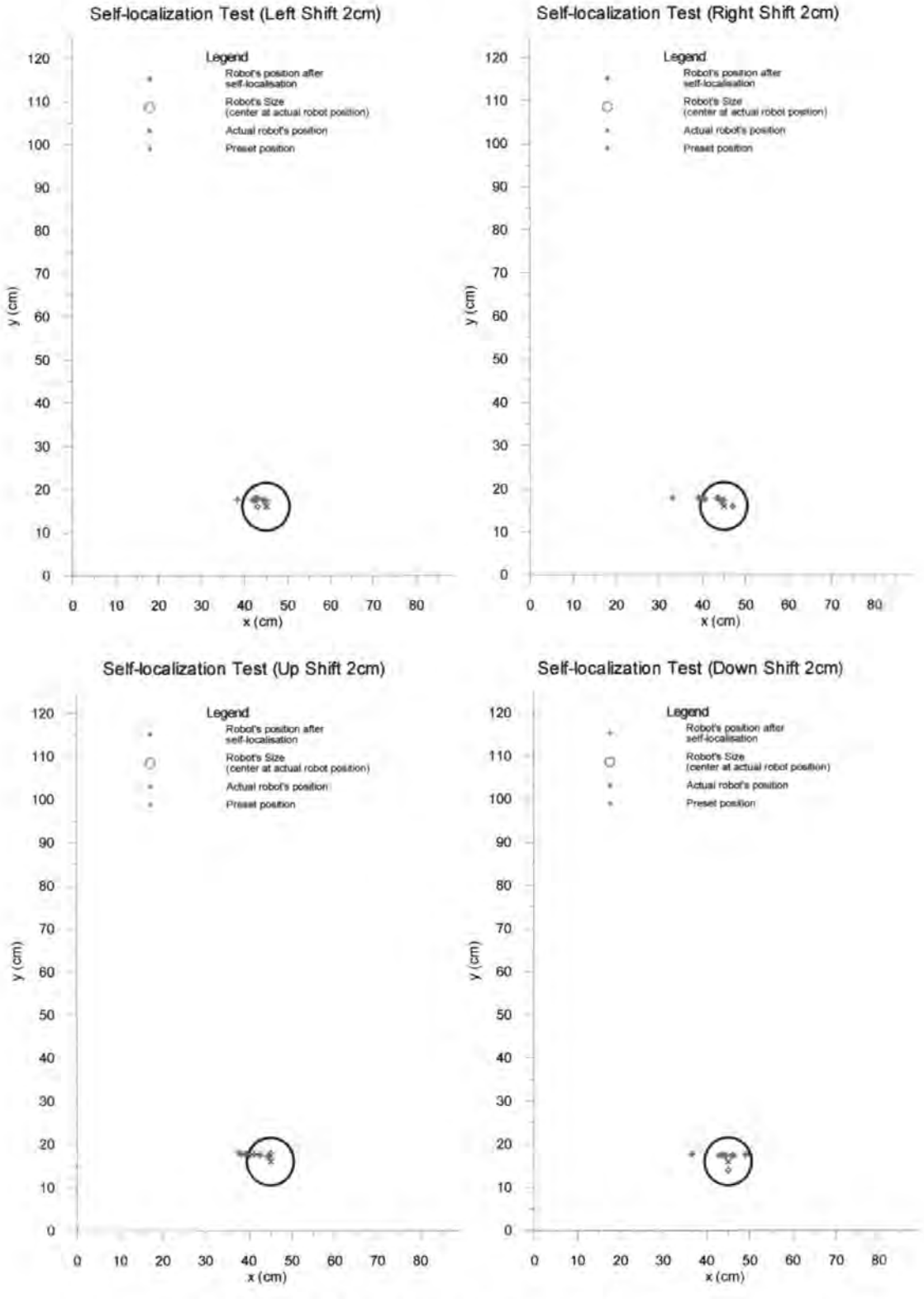


Figure 4.27: Examples of vision-based self-localization tests plots. The + symbols represent the result of the self-localization process. The **x** symbol is the actual position. The ♦ symbol is the initial position assumed by the self-localization algorithm.

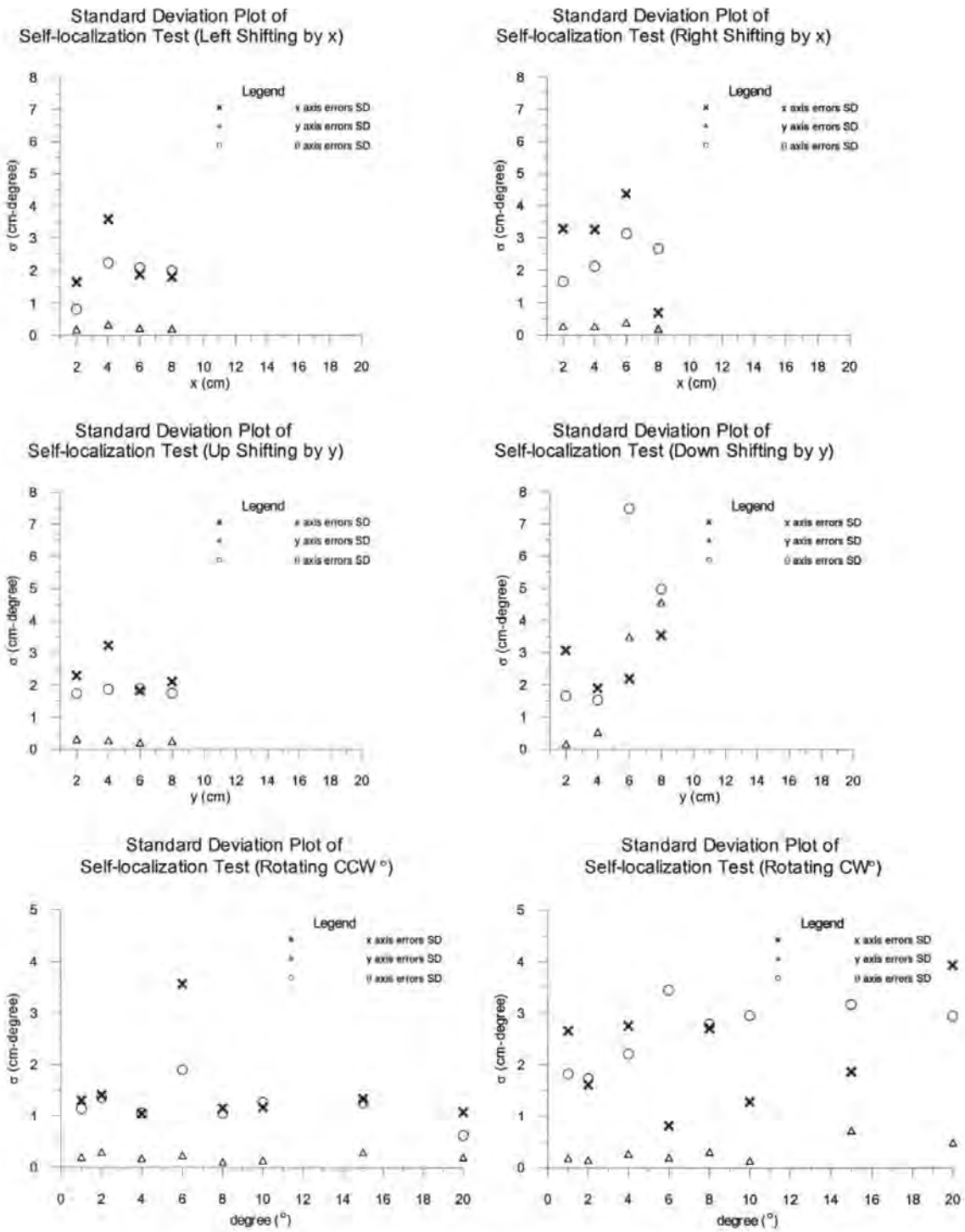


Figure 4.28: Standard deviation of vision-based self-localization errors. Each value is obtained from 10 measurements.

4.5.2 Discussion of the Vision-based Self-localization Results

The results show that self-localization along the y dimension is much more accurate than that along the x dimension or the orientation. The reason for that behaviour will be discussed in this section.

Figure 4.29 is an example of the image used in the vision-based self-localization process taken at coordinate (45, 16). Note that although the camera is looking at the same location throughout the test series with the same illumination conditions, the number of edges detected might be slightly different due to pixel noise that effects the automatic thresholding process.

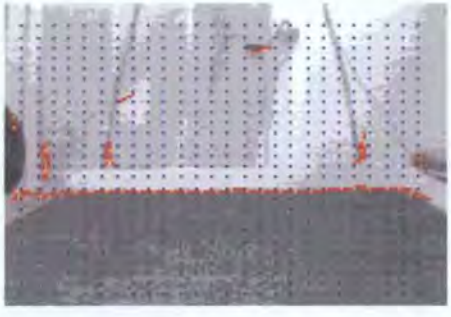
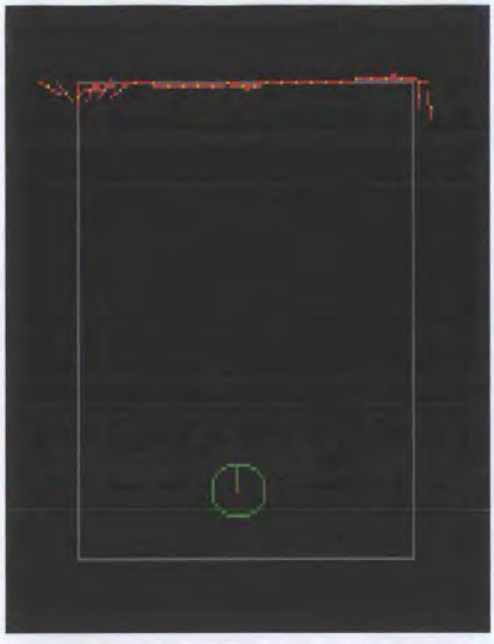


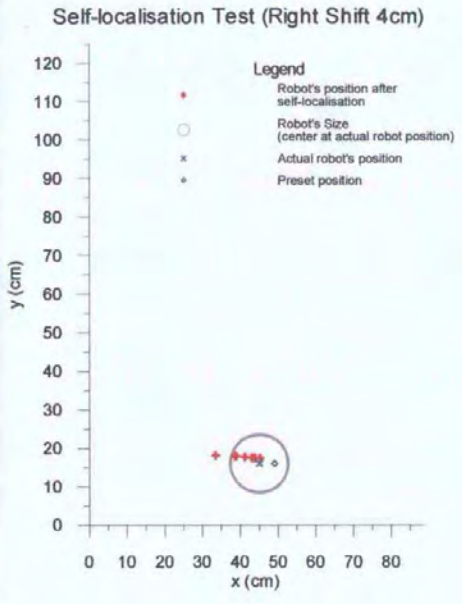
Figure 4.29: Example of the image taken at coordinate (45, 16). Illustrating the result of edges detection, showing the smaller number of visual cues on the sides of the robot's environment.

Figure 4.30 shows a test result where the initial belief of the robot position is shifted by a lateral offset. The picture on the left shows where the remote brain initially thinks the robot is, and where the edges detected in figure 4.29 should be relative to the robot's position. The picture on the right shows the robot's positions indicates by + symbols after 10 runs of vision-based self-localization. With an accurate vision-based self-localization process, the + symbols should end up at coordinates (45, 16), where the actual robot is located (symbol **x**).

Figure 4.30a shows that the vision-based self-localization performs quite well in the condition where there are plenty of visual cues (i.e. the detected edges). This is illustrates in this figure where the y coordinates from the calibration results does not vary as much as the x coordinate, since there are plenty of detected horizontal edges to be matched. Figure 4.30b also illustrates this effect, showing that no calibration on the x coordinates has taken place since there are no detected edges that have similar orientations to the left and right edges of the environment. Therefore the + symbol is staying close to the ♦ symbol.

(a)





(b)

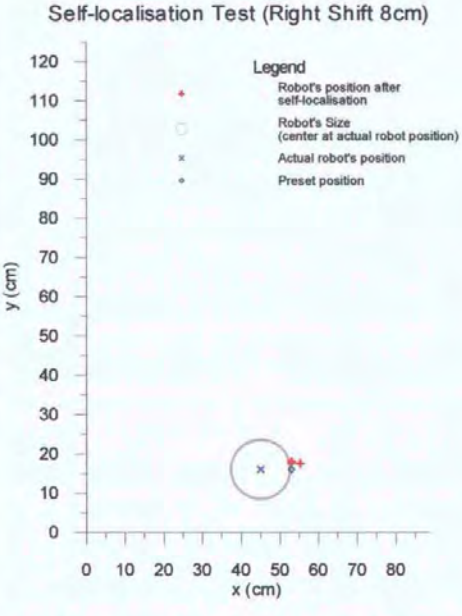
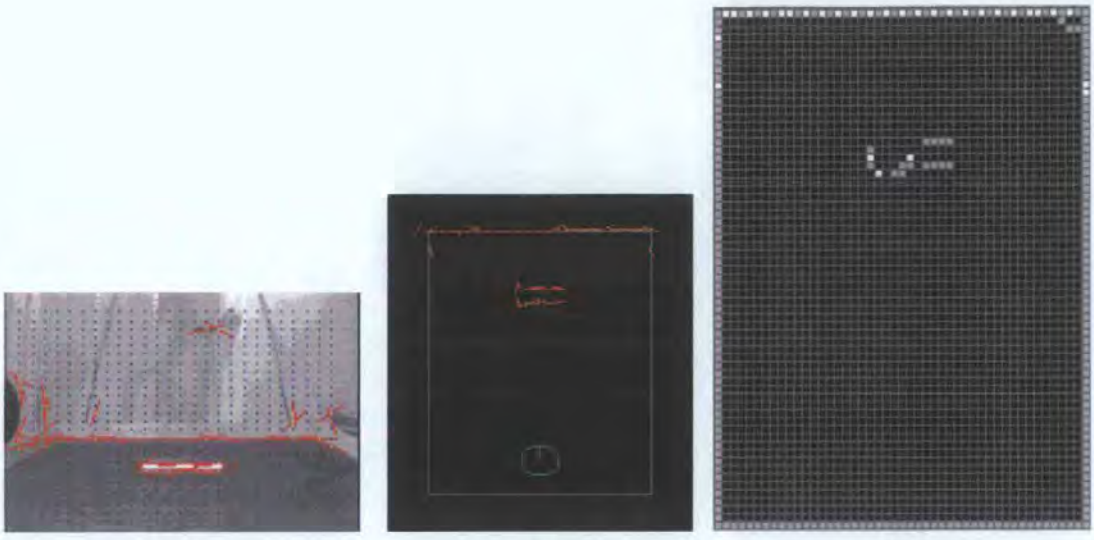


Figure 4.30: Examples of the vision-based self-localization tests results. (a) Position shifted by 4 cm to the right. Note that there are not many detected edges with similar orientation to the side walls therefore the robot's x coordinate varies at each trial. (b) Position shifted by 8 cm to the right. Note that the results of the calibration process are located closed to the initial position (♦) instead of the actual position (**x**). Since there is no visual information (edges) that have the same orientation to both sides of the wall, this prevents recalibration from taking place for the robot's x coordinate.

4.6 Obstacle Detection and Registration

Once the vision-based self-localization process finishes recalibrating the robot's coordinates in the prior map, all the detected edges that are within the room are placed into the map. For the detected floor and walls edges with a nearest neighbour, they are assigned to their nearest neighbour, while the rest of the detected edges with no nearest neighbours are assumed to be obstacles and are pasted into the prior map. The data structure used for the prior map is known as the neuro-resistive grid which has a spatial memory layer. The spatial memory layer is used to store information such as the robot's position, the goal's position and the detected obstacle positions. The neuro-resistive grid uses the spatial memory layers to calculate its potential field distribution that is used for path planning. Details of the neuro-resistive grid are described in chapter 5.

The robot's prior map (i.e. the spatial memory layer of the neural-resistive grid) is updated throughout the navigation process based on the latest information decoded from the images obtained through the robot's video camera. This information includes the latest position and orientation of the robot and the position of detected obstacles within the robot's environment. The updating process is illustrated in figure 4.31 where a) show the edges detected on the images, b) the position of the detected edges after self-localization and c) the detected edges and obstacle registered in the spatial memory layer of the neuro-resistive grid.



(a) (b) (c)

Figure 4.31: The process of registering the detected obstacle into the spatial memory layer of the neural-resistive grid. (a) The results of the edges detection process on the image obtained from the robot's video camera, (b) After the vision-based self-location process, the detected edges shows a good match with the prior map. (c) The detected obstacle is mapped into the spatial memory of the neural-resistive grid.

4.7 Discussion

The vision-based obstacles detection, self-localization and map updating processes used in this project have been described and tested. The results from the test of these procedures were shown.

The fish-eye lens calibration procedure shows a simple way of calibrating images that suffer from barrel distortion. The result from the calibration process shows its reliability and robustness.

The automatic thresholding method is successfully used for determining the threshold values for the floor and walls; this is useful as it helps the vision system to become less light sensitive.

The floor edges specific filters currently work at discrete locations in the image. The advantage is that it reduces the computational load. The disadvantage is the difficulty of

determining accurately the position of the detected edges. The proposed refined method solves this problem.

Vision-based self-localization is an important procedure in this project. It provides feedback for the robot's remote brain and makes other procedures such as path planning possible. This is a simple approach that produces acceptable results. The accuracy problems of this procedure are mostly caused by the positioning problem from the floor edges specific filters. The vision-based self-localization procedure will be more robust once the positioning problem is solved. The obstacle detection and registration procedure currently wasn't able to distinguish between real and phantom obstacles. Therefore all the detected edges were currently being registered into the neuro-resistive grid which is then used for path planning. The phantom obstacles problem will be addressed in future work.

Chapter 5

Path Planning and Encoding

Path planning is a basic function of most mobile robot or autonomous vehicle control systems. It involves generating a sequence of commands that will be used to navigate the mobile vehicle from its current position toward its final position/goal without colliding with obstacles. To achieve this, a map with data structures that suits the chosen method of path planning is needed. The data structure has to be able to store the information about the state of the mapped areas and enable movements from any element in the structure to the elements which represent adjacent areas in space. In addition, a data structure for storing paths that complements the map data structure, and efficient algorithms for locating the robot, path searching and navigation are required. Easy integration of sensory data for map construction, adaptation and extension is also a must. Based on these requirements, the neural-resistive grid method (Bugmann, Taylor and Denham, 1994; Althöfer and Bugmann, 1995) is chosen as the ideal data structure to be used in this project.

This chapter is divided into two sections, the path planning through the neural-resistive grid and the path encoding and decoding through normalised radial basis functions (NRBF). Section 5.1 discusses path planning using the neural-resistive grid. This section begins with an introduction of the theory behind the neural-resistive grid

(section 5.1.1), followed by a description of the representation of the robot and the obstacle within the neural-resistive grid, and the illustration of obstacle free path planning based on the gradient distribution in the neural-resistive grid (section 5.1.2).

Section 5.2 describes how the obstacle free path is represented by a waypoint data structure and is used in the robot navigation process. For this, the NRBF net will be described in section 5.2.1 followed by showing how it facilitates the path encoding in section 5.2.2 and the path decoding in section 5.2.3.

5.1 Path Planning through the Neural-resistive Grid

5.1.1 Neural-resistive Grid

The route-finding neural net proposed by Bugmann, Taylor and Denham (1994) was used in this project for environment mapping and path planning. The route-finding neural net is a neural implementation of a resistive grid; it consists of two layers, a neuro-resistive grid and a spatial memory layer. In the neuro-resistive grid, every node is connected to its $2N$ neighbours. N is the dimension of the represented state space (N=2 in our case). Each node is also directly connected to the node corresponding to the same spatial location in the spatial memory layer as shows in figure 5.1.

The neural-resistive grid holds the ideal data structure characteristic for environment mapping and path planning as the spatial memory is able to store goal and obstacles information about the mapped area which enables easy integration of sensory data with a simple algorithm for map construction, adaptation and extension while the neuro-resistive grid calculates the potential distribution over the mapped area based on the information encoded in the spatial memory. The neural-resistive grid is updated every image processing cycle as new sensory data become available.

The Concept

The potential distribution is calculated based on the law of physics where electric current flows from a node of a higher potential toward a node of a lower potential. Here, the node corresponding to the goal state is set with a highest potential value (i.e. one) while nodes correspond to obstacles, or forbidden states, are set to a low potential value (i.e. zero). Therefore we have currents flowing from the goal, through the grid and towards the obstacle nodes. At any point in the grid, the direction opposite to that of the current flow indicates a path to the goal.

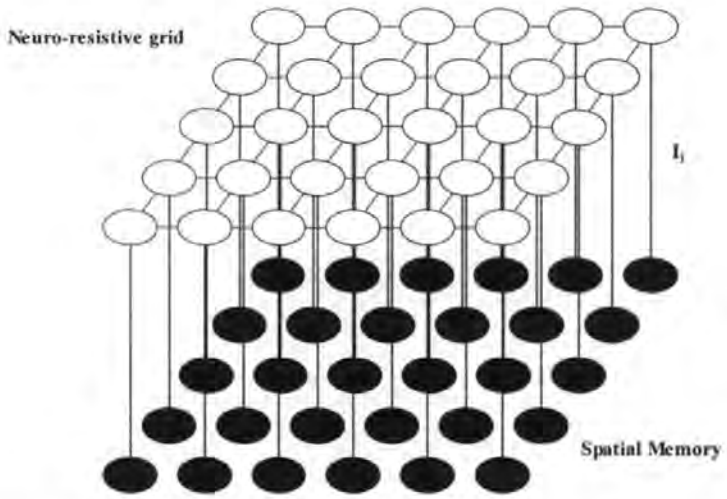


Figure 5.1: The neural-resistive grid planner is composed of the neuro-resistive grid layer and the spatial memory layer.

The Theory

In the implementation of this concept, the neuro-resistive grid receives inputs from its N_i neighbours and one of its corresponding neuron in the spatial memory layer. Each output or "potential" y_i of the neuron i is calculated as follows

$$y_i = Tf\left(\sum_{j \in N_i} w_{ij} y_j + I_i\right) \tag{5.1}$$

where w_{ij} is the weight given to the input from neuron j to neuron i; y_j is the output of neuron j; I_i is an external input used to constrain the value of y_j, and Tf is the transfer

function of the neuron i. The linear saturating function illustrated below (figure 5.2) is used as the transfer function Tf.

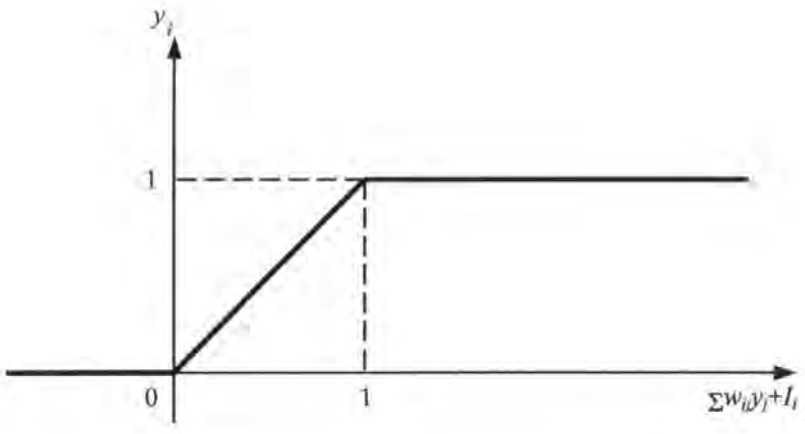


Figure 5.2: Transfer function of the neurons representing nodes of the neuro-resistive grid.

For a two-dimensional case, the weight w_{ij} is set to $1/N_i$ (=0.25), which makes y_i the average potential of the N_i neighbours. As nodes at the edge of a two-dimensional grid have only 3 neighbours (N_i=3) and those at the corners only 2 neighbours (N_i=2), their input weight must be set to w_{ij}=0.333 and w_{ij}=0.5 respectively. The saturation of the transfer function for inputs larger than 1 or smaller than 0 only happens for nodes corresponding to target or obstacles respectively due to external inputs from the spatial memory. These cause the goal neuron to have a potential y_j=1 as the external input is set to I_i=1 and the neurons corresponding to obstacles to have a potential y_j=0 as their external inputs are set to I_i=-1. The external inputs for nodes that are neither the target nor obstacles are set to I_i=0. Therefore, these nodes determine their potential freely, according to the potentials of their neighbours. Before an equilibrium distribution of the potentials is achieved, all the neurons in the network must be updated several times. Theoretically, an infinite number of updating cycles is needed but in practice only a few tens of iterations are needed to achieve a correct direction of potential gradients. The minimal number of iterations depends on the distance between the current position of the robot and its target while taking account the complexity of the maze. Note that the gradients distribution does

not need to be recalculated each time provided that there is no new obstacle, and the obstacles and target remain static.

The Setup

To model the robot's environment, a neural-resistive grid with 47×65 nodes was built. The walls in the robot's environment were pre-programmed into the outer nodes of the spatial memory of the neural-resistive grid. In the neural-resistive grid, the spatial memory is used as a prior map for map updating while the neuro-resistive grid is used for path planning.

The actual size of the robot's navigation area is $89\times125cm^2$, which is represented by 45x63 nodes in the neural-resistive grid. Thus, each node in the neural-resistive grid covers a $2\times2cm^2$ area of the robot's environment.

5.1.2 Representation of Robot and Obstacles in the Neuro-resistive Grid

Representation of robot and obstacles in the resistive grid plays an important role in producing obstacle free paths and assuring clear navigation for the robot. In the neural-resistive grid method, the robot is modelled by a point of the size of a node while obstacles and walls are expanded by the radius of the robot (figure 5.3) to make sure that the path produced will avoid a collision with an obstacle and that the robot will not attempt to go through any corridor that is too narrow for it.

The expansion is achieved by using divergent connections (one-to-many) from the spatial memory to the neuro-resistive grid. Therefore equation 5.1 becomes

$$y_i = Tf\left(\sum_{j\in N_i} w_{ij}y_j + \sum_{j\in M_i} I_j\right) \tag{5.2}$$

where M_i represents the number of divergent connections from node i of the spatial memory to the neuro-resistive grid. This is a novel design, extending the functionality of the original neural-resistive grid in figure 5.1.

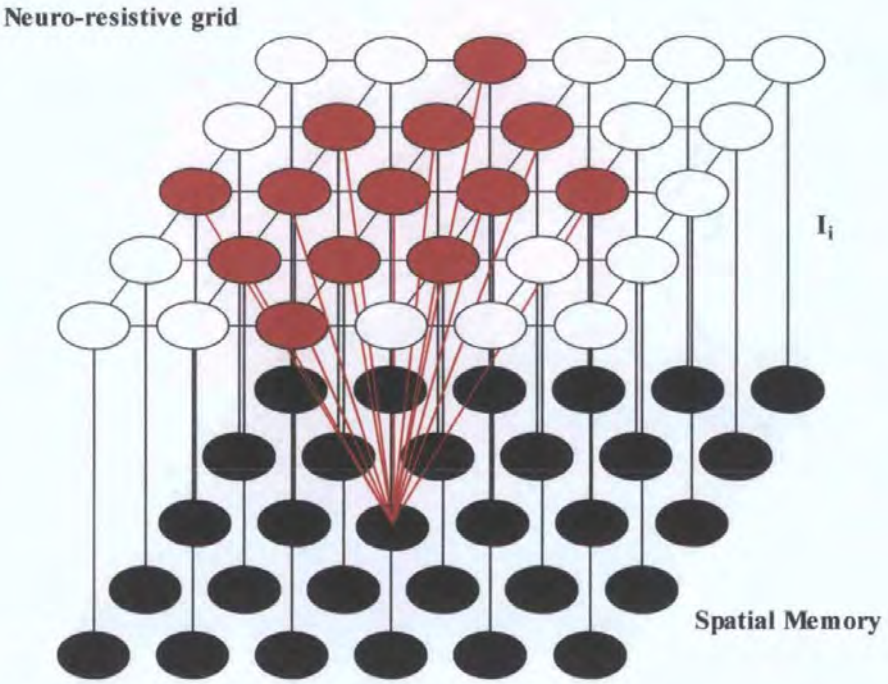


Figure 5.3: Modified neural-resistive grid with one-to-many connections from the spatial memory layer to the resistive grid layer. The radius of the connectivity is equal to the radius of the robot.

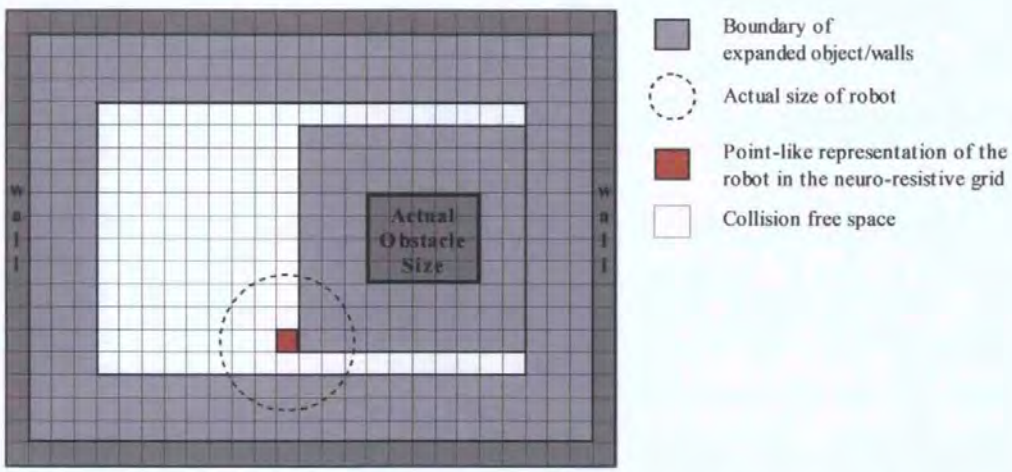


Figure 5.4: Representation of walls, obstacle and collision free space within the neural-resistive grid. The robot is represented by a node in the grid while obstacle and walls are expanded by the radius of the robot to ensure that collision free paths are planned.

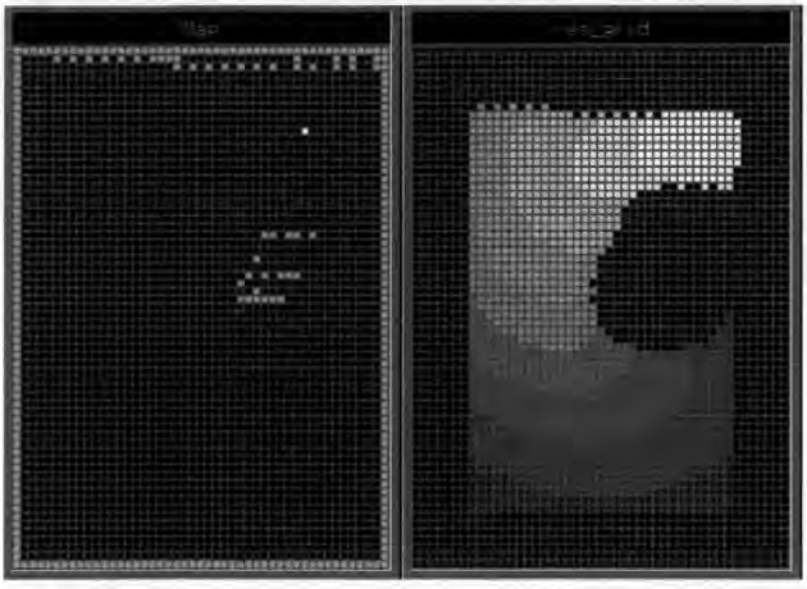


Figure 5.5: The neural-resistive grid representation. The spatial memory (map) shown on the left represents free space in black colour, while occupied areas (i.e. the pre-programmed walls and the detected walls and obstacles) in grey colour. The white node in the top right quadrant of the map represents the goal position. The neuro-resistive grid on the right uses the spatial memory to produces the potential distribution of the free space in the robot's environment. The free space is represented by a gradient of grey levels while the forbidden space (i.e. walls and obstacle after the expansion process) is represented by black-coloured nodes.

5.1.3 Path Planning through Gradient Climbing in the Neuro-resistive Grid

This section describes the generation of waypoints that define a collision free path based on the potential distribution in the resistive grid. The waypoints which are along the collision free path are later to be sent to the robot which uses them to produce steering controls.

The initial aim of waypoints generation is to search for a collision free path from the robot's current position to the goal. This is done by searching through the neuro-resistive grid, for a series of highest potential neighbour nodes from the robot's location toward the goal. The algorithm begins by searching through the nearest neighbours of the node where the robot is located for a node with the highest potential, then move to this node and continues the search. Every fifth highest potential node found (i.e. indicate as green in the resistive grid shows in figure 5.6) defines a waypoint. The

searching process is repeated until 8 waypoints are found (there is no need for calculating more than 8 waypoints at a time, as explained in chapter 6).

Figure 5.6: Waypoints representation of the path within the neuro-resistive grid. The neuro-resistive grid shows the location of the robot in red while the waypoints that form a path from the robot location toward the goal in green.

5.2 Path Encoding and Decoding through NRBF Nets

This section describes the NRBF path encoder that is used in the robot. The function of the NRBF path encoder is to continuously produce a target point for the robot to follow. The target point is a close point on the obstacle free path ahead of the robot. The purpose of the target point is to attract the robot towards and along the obstacle free path until the robot reaches the goal.

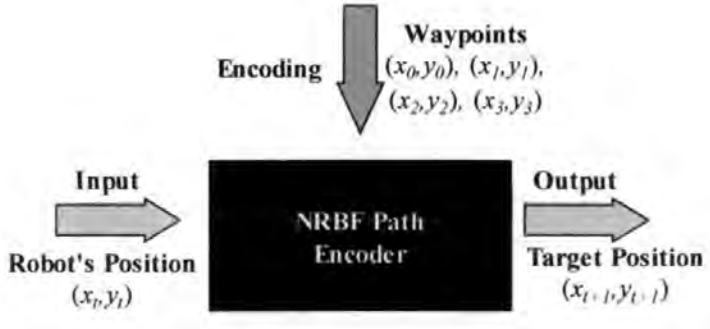


Figure 5.7: The NRBF path encoder. The NRBF path encoder takes the robot's position and produces a target position for the robot controller to steer the robot toward it. The target position is a position along the encoded path.

5.2.1 The Normalised Radial Basis Function (NRBF) Net

Standard Radial Basis Function (RBF) nets comprise a hidden layer of RBF nodes and an output layer with linear nodes (Broomhead, 1988; Brown, 1994). The function of these nets is given by:

$$y_i(\vec{x}) = \sum_{j=1}^{n} w_{i,j}\phi(\vec{x} - \vec{x}_j) \tag{5.3}$$

where y_i is the activity of the output node i, $\phi(\vec{x} - \vec{x}_j)$ is the activity of the hidden node j, with a RBF function centred on the vector $\vec{x}_j$, and $\vec{x}$ is the actual input vector and w_{ij} are the weights from the RBF nodes in the hidden layer to the linear output node (Figure 5.8). Such a net is a universal function approximator (Powell, 1987).

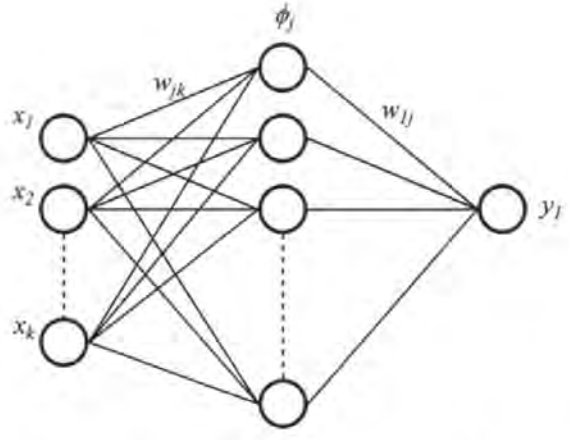


Figure 5.8: Network architecture for standard RBF nets and Normalized RBF nets.

The function $\phi(\vec{x} - \vec{x}_j)$ of a hidden node j is usually the Gaussian Radial Basis Function:

$$\phi(\vec{x} - \vec{x}_j) = \exp\left(-\frac{\sum_{k=1}^{K}(x_k - w_{jk})^2}{2\sigma^2}\right) \tag{5.4}$$

where σ is the width of the Gaussian and K is the dimension of the input space. The "weights" w_{jk} (shown in figure 5.8) between node k in the input layer and node j in the

hidden layer do not act multiplicatively as in other neuron models, but define the input vector $\vec{x}_j = (w_{j1}, \ldots, w_{jk})$ eliciting the maximum response of node j ($\vec{x}_j$ is the "centre of the receptive field").

Normalised RBF nets have a functional form very similar to the standard one (equation 5.3), with the difference of a normalisation by the total activity in the hidden layer:

$$y_1(\vec{x}) = \frac{\sum_j w_{1j}\phi(\vec{x} - \vec{x}_j)}{\sum_j \phi(\vec{x} - \vec{x}_j)} \tag{5.5}$$

As a result, the output activity becomes an activity-weighted average of the input weights in which the weights from the most active inputs contribute most to the value of the output activity. For instance, in the extreme case where only one of the hidden nodes is active, then the output of the net becomes equal to the *weight* corresponding to that hidden node, whatever its actual activity. Thus RBF nodes in the hidden layer are used here as case indicators rather than as basis functions proper.

Figure 5.9 shows that each hidden node in a Normalized RBF net takes over a portion of the input space over which it determines the output of the net. Due to this property, outputs of the normalized RBF net are always a point on the path, even if the current position is not exactly a waypoint. In contrast, the standard RBF net produces outputs out of the path for input positions that are not exactly on a waypoint.

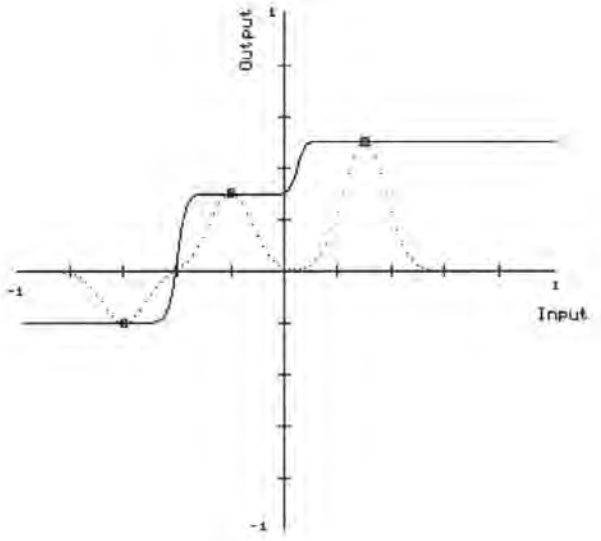


Figure 5.9: Comparison between standard RBF nets and Normalized RBF nets with three hidden nodes on an example of a 1-Dimensional path. The path has 4 waypoints: x = -0.6, -0.2, 0.3, 0.5. The path can be represented as a mapping {-0.6 -> -0.2; -0.2 -> 0.3; 0.3 -> 0.5}. Dotted line: function of a standard RBF net approximating the mapping. Full line: Function of a Normalized RBF net.

A similar normalisation principle is used in the "centre of gravity defuzzification method (Brown and Harris, 1994, pp. 388-404). Our approach is a special case of the approach proposed by (Shao, Kee and Jones, 1993) for selecting linear functions $L_{ij}(x)$ (instead of the constant weights w_{ij} used here). In (Rao and Fuentes, 1996) equation 5.5 was used to compute normalised motor output vectors in robots. Normalised RBF nets have also been used for path encoding in an autonomous wheelchair (Koay, Bugmann, Barlow, Phillips and Rodney, 1998) and show very good properties in pattern classification applications (Bugmann, 1998).

5.2.2 Path Encoding

Encoding a path in a 2-dimensional space is done with an NRBF net with two input nodes and two output nodes, and one hidden node per waypoint. The centre of the receptive field of each hidden node is set to the position (x_n,y_n) of one waypoint (equation 5.6) and its output weights are set to the position (x_{n+1},y_{n+1}) of the next waypoint (equation 5.7).

$$w_{j1} = x_n; \qquad w_{j2} = y_n \tag{5.6}$$

$$w_{1j} = x_{n+1}; \qquad w_{2j} = y_{n+1} \tag{5.7}$$

Therefore when the robot reaches the target (x_n,y_n), this activates hidden node j and its output weights (w_{1j},w_{2j}) become the new target (x_{n+1}, y_{n+1}), which pulls the robot along the path. To enable the robot to stop its motion when it reaches the final waypoint, the input and output weight of the final hidden node are set to the final waypoint, hence the target will keep pointing at the same point and the robot will stop. The targets change when new waypoints sent by the remote brain are encoded into the NRBF path encoder.

5.2.3 Path Decoding

The NRBF path encoder is a function that provides a target position (x_t,y_t) for the robot based on the robot's current position (x_c,y_c) as input (i.e. equation 5.8 and equation 5.9). The target position is usually a point along the demanded path encoded in the NRBF path encoder if the robot is in a position close to the path. If the robot is somewhere outside the path, the target position will be a point nearer to the demanded path.

$$x_t = \frac{\sum_j w_{1j}\phi(\vec{x} - \vec{x}_j)}{\sum_j \phi(\vec{x} - \vec{x}_j)} \tag{5.8}$$

$$y_t = \frac{\sum_j w_{2j}\phi(\vec{x} - \vec{x}_j)}{\sum_j \phi(\vec{x} - \vec{x}_j)} \tag{5.9}$$

where

$$\vec{x} = (x_c, y_c)$$

Note that the robot will not attempt to reach exactly each intermediate waypoints, because when it reaches the neighbourhood of a waypoint, it is directed towards the next waypoint. Thus, for a more precise path following, waypoints must be closely spaced. Figure 5.10 illustrates a case where the spacing between waypoints is much too large.

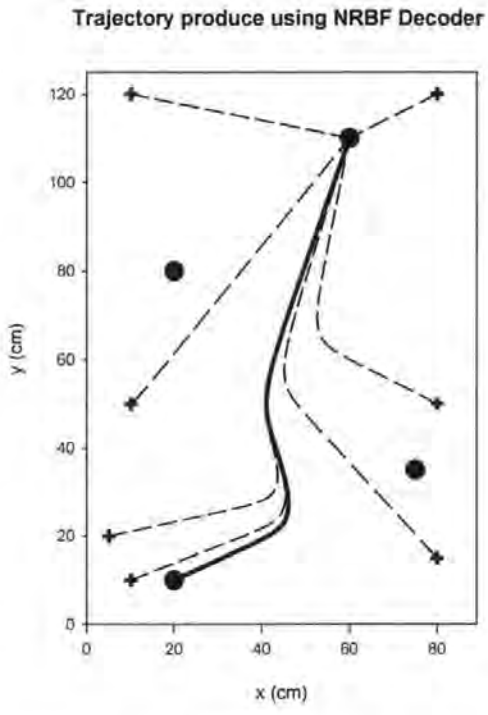


Figure 5.10: A simulation of the NRBF path encoder attractive field. The four dots represent the waypoints while the solid line shows the path from the initial waypoint to the final waypoint. The 7 crosses represent various starting position and the dotted lines represent their paths.

Another point is that σ in equation 5.4 must be of the order of the distance between waypoints, so that only one hidden node at a time is activated and defines the next waypoint. For too large values of σ the produced target becomes a combination of waypoints and the path is smoothed out.

However, the NRBF path encoder has the advantage of being able to produce a target point that will lead the robot towards the demanded path from whatever starting point as illustrated in figure 5.10. This is particularly useful when the robot has left the desired path by error.

Chapter 6

Motion Control with Intermittent Delayed Measurements

This chapter discusses motion control with intermittent delayed measurements in the system (the remote brain and the robot). A delayed measurement is defined as a measurement which is delayed by $n\tau$, where τ is the controller's cycle time and n is the number of cycles between data acquisition and data availability. Delays in measurements are usually introduced by the complexity of processing sensory data. Applications such as vision-based mobile robots are often faced with delayed measurements from visual sensing. Delayed measurements used to cause robots (i.e. robotics system) to exhibit a stop-and-go motion (Moravec, 1983). For example, a mobile robot that relies on vision for its navigation process has to wait for the visual sensory data to become available before the navigation process can be executed. Delayed measurements are due to processes such as image digitization, image processing, self-localisation, path planning and data transfer. This is not a problem that can be solved with a faster or more powerful machine, as not all of these processes depend on the computation speed. Furthermore, computation time also tends to increase with more intelligent and complex algorithms (Bak, Larsen, Norgaard, Andersen, Poulsen and Ravn, 1998). Apart from that, not all time delays are caused by the

controller, for example, in a rolling mill process where the time delay lies between the issuing of a control and its result's feedback (Smith, 1959).

Section 6.1 describes the vision-based mobile robotic system used and the time delay problem that exists in the system. This section is divided into two subsections; section 6.1.1 discusses the system in details, its time delays and the cause of the stop-and-go motion, while a solution to the problem is presented in section 6.1.2.

Section 6.2 deals with the proposed solution to the stop-and-go motion problem which was discussed in section 6.1.2. This includes the use of receding horizon control (in section 6.2.1) and the adaptation of the retroactive updating scheme in the Smith Predictor to the case of intermittent delayed measurement (section 6.2.2).

Section 6.3 deals with the implementation of the Smith Predictor in which a robot model is built (section 6.3.1) followed by the derivation of a set of equations for tracking the robot pose (section 6.3.2) using the distances travelled by the robot's wheels (determined either by the model or direct readings of the shaft encoders).

Section 6.4 describes the robot's on-board path control followed by test results using the NRBF path encoder.

Section 6.5 describes and discusses the specifically designed coordinate recalibration algorithm for mobile robotic systems that incorporates intermittent delayed measurements through retroactive updating.

6.1 The Time Delays Problem

In control, a system always consists of several components that act together and perform certain functions. Each component requires an amount of time to complete its task. The amount of time required depends on several factors such as the complexity of the task and the speed of the hardware involved. This amount of time (i.e. time delay) often poses a serious threat to the performances of a real-time system.

The vision-based navigation system used in this research also suffers from time delays problem. As a result, the system exhibits a stop-and-go motion. This is unacceptable especially for a real-world system such as an autonomous wheelchair. The aim here is to analyse the time delays within the system and to propose a solution that will solve the stop-and-go motion problem.

Section 6.1.1 looks at the vision-based navigation system, its control structure and timing diagram to investigate the relationship between the time delays and the stop-and-go motion. Note that the system was designed with the use of sequential control method.

Section 6.1.2 proposes a solution to deal with the time delays and overcome the stop-and-go motion problem through a concurrent control method.

6.1.1 Sequential Control "Compute then Move"

The aim of the vision-based navigation system used in this research is to navigate around obstacles towards the goal. The vision-based navigation system flow diagram in figure 6.1 shows its components and their relationship within the system. The components are grouped into two categories or sub-systems, the first group is known as the remote brain. As its name implies, the remote brain deals with high level tasks which are

responsible for the sensing, thinking and planning processes; the second group is the robotic system which deals with low level tasks such as controlling the robot's motion.

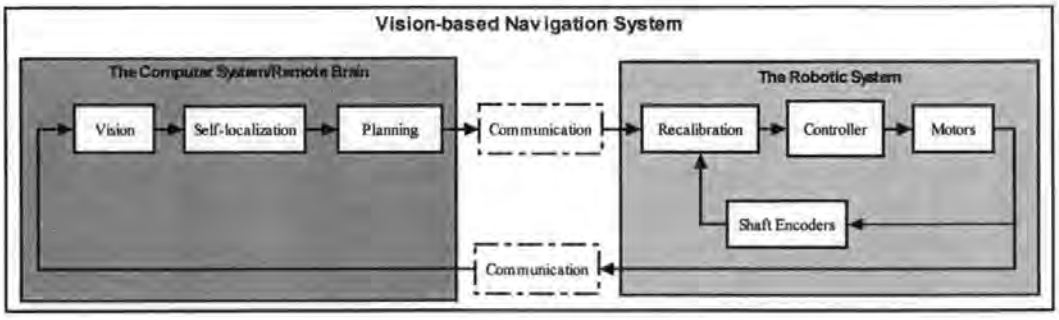


Figure 6.1: Vision-based Navigation System Sequential Control Flow Diagram.

Figure 6.1 also demonstrates the relationship between the remote brain and the robotic system. The remote brain is responsible for determining the robot pose through the image obtained from the robot's video camera, and then plan an obstacle free path from the robot pose to the goal point. This path and an activation signal are transmitted to the robotic system which is responsible for the robot's navigation processes. The remote brain then switches into sleep mode while waiting for the robotic system to finish its navigation to the target point. The target point in this case is a point along the path from the robot's initial position to the robot's goal point. Once the robot has reached the target point, the robotic system stop the robot's motion, and sends an activation signal to reactivate the remote brain's sensing, thinking and planning process. The whole program cycle is then repeated until the robot reaches the goal point. Figure 6.2 shows the sequential control vision-based navigation system tasks scheduling diagram which demonstrates the sequence of tasks being executed during a navigation process. This diagram also illustrates the main system program cycle and the relationship between the remote brain and the robotic system within it.

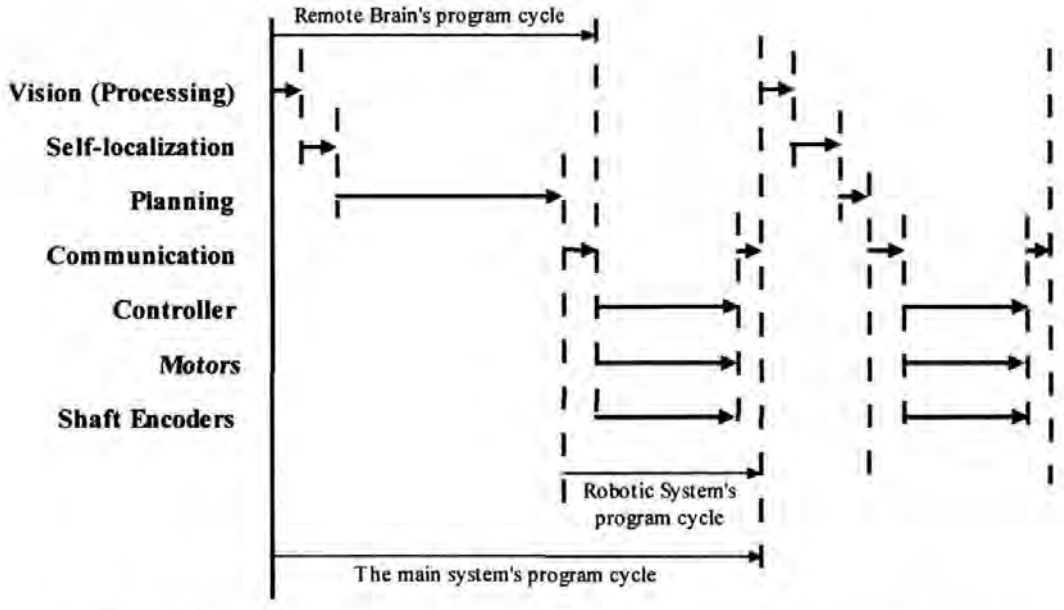


Figure 6.2: Tasks scheduling diagram for the sequential control vision-based navigation system. This diagram show how each tasks is executed and illustrates the sequential control where the main system begins its program cycle by executing the high level tasks (i.e. Vision, Self-localisation and Planning modules) followed by the communication tasks and concludes with the low level tasks (i.e. Controller, Motors and Shaft-encoder modules). Note that the scale in this diagram represents only an approximation of the actual delays.

This control method is known as the sequential control method since both program cycles within the main system work in a sequential manner. Here, the robotic system has to wait for the remote brain to finish executing the main system's high level tasks before it can execute the main system's low level tasks. The duration for executing the main system's high level tasks varies as it depends on the complexity of the captured image. In average, the vision processing task requires about 0.55 seconds of execution time. The self-localization task requires about 2 seconds of execution time. The planning process requires about 25 seconds of execution time initially (to perform 80 updating cycles per program cycle on the neuro-resistive grid) to achieve a correct direction of potential gradients (as described in section 5.1.1), then about 0.68 seconds (for performing 1 updating cycle and selecting new waypoints) at each program cycle.

Due to the complexity and time consuming process within the high level tasks, time delays are created, causing the execution of low level tasks to be delayed. As a

consequence, the robot has to remain in a static state until the high level tasks program cycle finishes. The same applies to the remote brain which has to wait for the robotic system to finish executing the low level tasks before it can begin executing the high level tasks. The average execution time for the main system's low level tasks (robot motion) is about 10 seconds, and this execution time varies with the complexity of the path. Thus, the result of time delays within a sequential control method cause the robot to exhibit a stop-and-go motion.

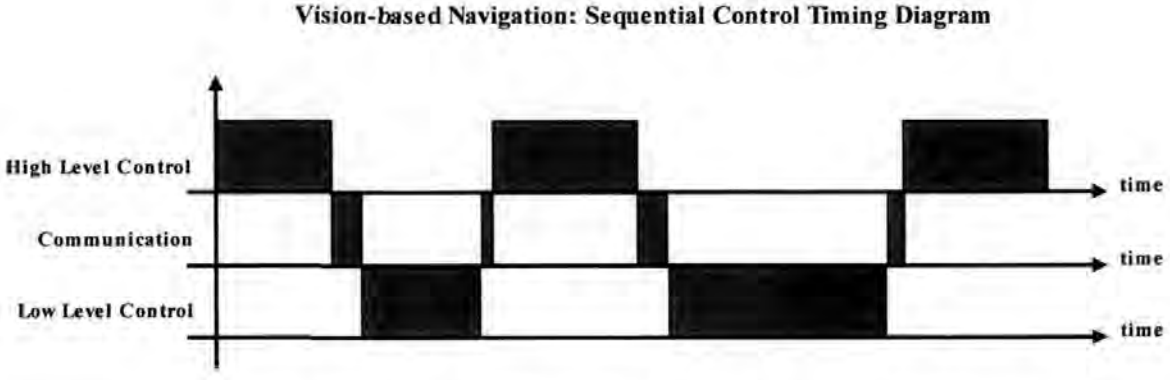


Figure 6.3: Vision-based Navigation System Sequential Control timing diagram. It demonstrates the tasks and their timing perform in the main system (which consists of the remote brain and the robotics system). The timing diagram is divided into three rows, the first row from the top represent the main system's high level tasks which is a collection of tasks executed by the remote brain, the second row represents the communication task involving both the remote brain and the robotic system, and the third row represent the main system's low level tasks which is a collection of tasks executed by the robotic system. Note that the scale in this diagram represents only an approximation of the actual delays.

Figure 6.3 illustrates the timing diagram of the main system. The timing diagram is divided into three rows, with the top row representing the main system high level tasks (which are handled by the remote brain), the second row represents the communication tasks between the remote brain and the robotic system (the communication process requires about 0.10 seconds of execution time for sending waypoints data to the robotic system while the communication process for reactivating the remote brain needs about 0.015 seconds), and the third or the bottom row represent the low level tasks (which are handled by the robotic system).

Many would think that the stop-and-go motion problem can be solved by using a faster machine, but this is not entirely true. As shown in figure 6.2, the amount of time the robot spends in a static state is equal to the amount of time required to execute the high level tasks and the communication tasks. Therefore, reducing the time required to execute the main system's high level tasks with a faster machine will only reduce the time during which the robot is in a static state, but does not eliminate the static state. This is because the vision-based navigation system involves wireless communication tasks (between its two sub-systems, the remote brain and the robotic system) whose execution time is not directly influenced by a faster machine.

Therefore it is clear that the stop-and-go motion problem cannot be solved by using a faster machine. Section 6.1.2 discusses the way of solving this problem using a concurrent control method.

6.1.2 Concurrent Control "Compute while Moving"

The stop-and-go motion is the result of the delays in the systems and the sequential conception of the control architecture.

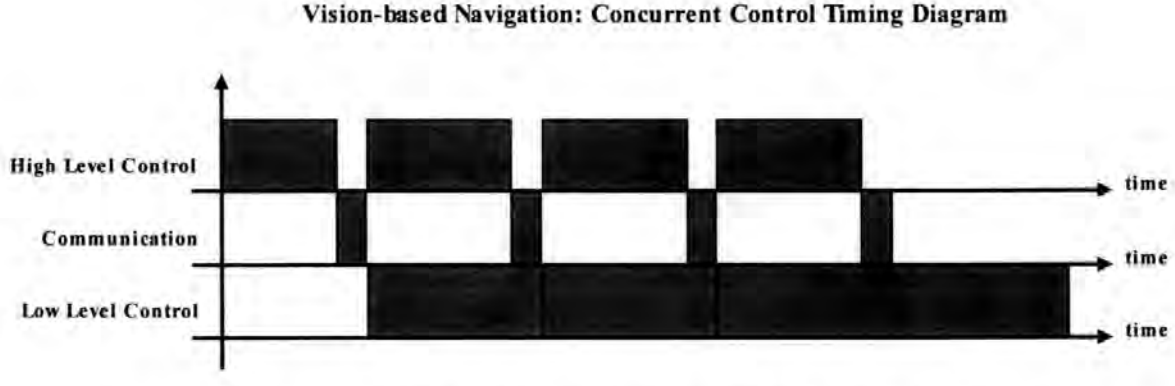


Figure 6.4: Timing diagram of a Vision-based Navigation System with Concurrent Control. Note that the scale in this diagram represents only an approximation of the actual delays.

To overcome the stop-and-go motion problem, the proposed solution is to run both the main system's high level tasks and low level tasks concurrently. This is done by executing the main system high level tasks to determine a new target point while the robot is still moving toward its current target point (low level task), and transmits the newly determined target point to the robotic system before the robot reaches it current target point. By doing so, the robot will move continuously form one target point to the next until it reaches the goal. The timing diagram of such system is illustrated in figure 6.4. From this diagram, it is clear that by executing the main system's high level tasks concurrently with the main system's low level tasks, it is possible to keep executing the low level tasks continuously from the robot's initial position to its goal point, therefore eliminating the robot's static state and overcoming the stop-and-go motion problem.

The tasks scheduling diagram for the concurrent control vision-based navigation system is shown in figure 6.5. This diagram illustrates the new program cycles for both the remote brain and the robotics system. It is important to note here that planning has to be done in advance, before the robot reaches the position for which the plan is relevant.

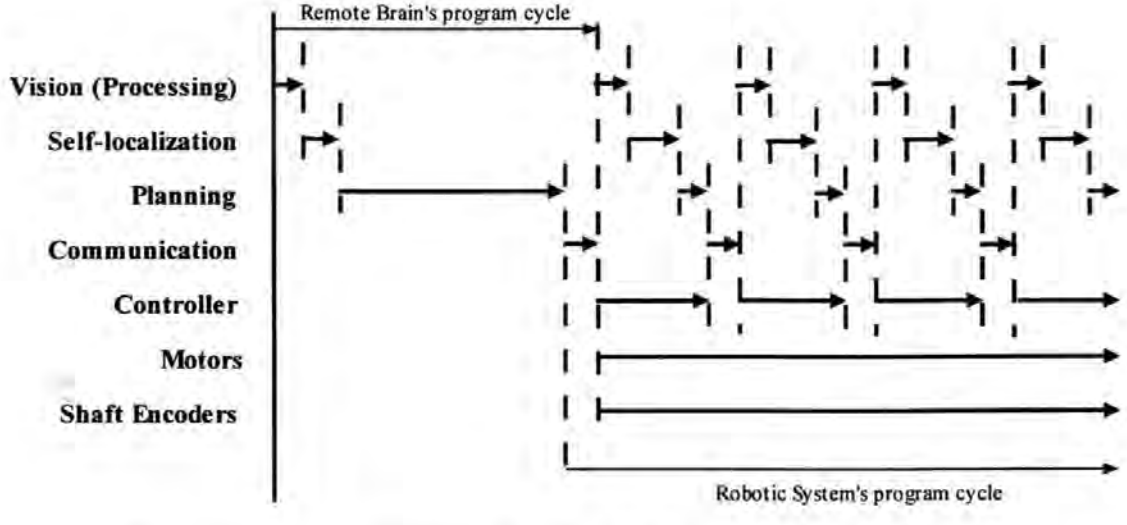


Figure 6.5: Tasks scheduling diagram for the concurrent control vision-based navigation system. Note that the scale in this diagram represents only an approximation of the actual delays.

Figure 6.6 shows the process flow diagram of a stop-and-go motion free system. This flow diagram is very similar to the one shown in figure 6.1 except that the remote brain does not need an activation message from the robotics system, as both the remote brain and the robotic system have independent program cycles as illustrate in figure 6.5.

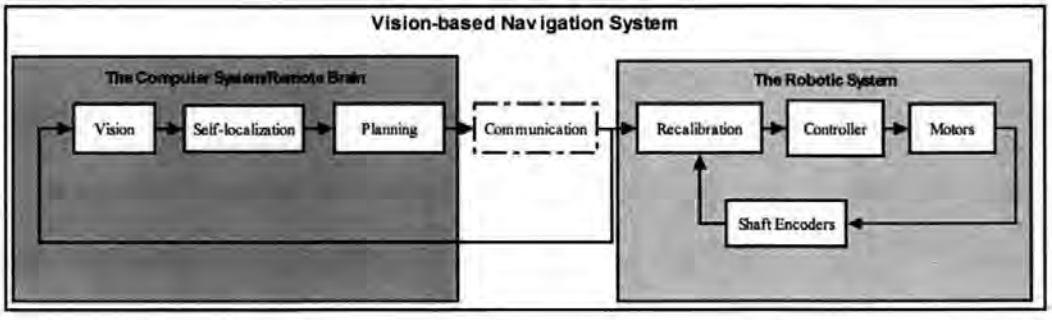


Figure 6.6: Concurrent control process flow diagram. The difference between this flow diagram and the one shown in figure 6.1 is the communication process. The flow diagram here has two independent program cycles while in figure 6.1, the remote brain and the robotic system work dependently through the communication module.

6.2 Proposed Implementation of Concurrent Control

This section proposes a strategy for implementing concurrent control into the system. The aim is to allow both the remote brain and the robot to function concurrently as illustrated in figure 6.4. The relationship between the remote brain and the robotic system are such that the robotic system relies on the remote brain for the robot's navigation process, therefore it is important that this is taken into consideration when implementing concurrent control.

Concurrent control requires concurrent sensory processing and planning while the robot is moving. This poses a problem for generating a meaningful motion from delayed information, during the delay between measurements. For example, let us assume the remote brain takes $n\tau$ seconds to complete the high level tasks. If an image is taken for processing at time t_0, while processing the high level tasks, the robot continues to move.

The robot will only receive the feedback (i.e. robot's pose at time t_0) and path information valid for time t_0 at time $t_0 + n\tau$, when the robot is already in a new position.

The delayed path information problem is solved here using a receding horizon control method (section 6.2.1) which overcomes the time gap between measurements by planning paths that are valid over a certain time range for the robot to follow while new plans are elaborated. When the new path becomes available, it can easily be integrated into the robot's navigation process and results in a smooth navigation motion (i.e. using the NRBF net as discussed in section 5.2).

A modified Smith Predictor is used here to control the robot during the delays between visual feedbacks (section 6.2.2). The Smith Predictor was originally designed to deal with continuous but delayed feedbacks. The modification proposed here allows it to handle intermittent delayed feedbacks such as the ones caused by image processing. We have explored two possible sources of the fast feedback component of the Smith Predictor. The first is the standard use of a dynamical model of the robot (section 6.3.1). The second is the more direct use of tracking information from shaft encoders (section 6.3.2)

The use of the modified Smith Predictor in conjunction with the NRBF path encoder for on-board path control is described in section 6.24.

In the modified Smith Predictor, when the delayed feedback (i.e. robot's position at time t_0) from the remote brain becomes available at time $t_0+n\tau$, it is used to improve the estimation of the robot's current position (based on the assumed position at time t_0 and the integration of displacements estimated from the fast feedback component) by retroactively updating the position assumed for time t_0, this coordinate recalibration process is described in section 6.25.

6.2.1 Receding Horizon Control Strategy

The receding horizon approach is used here to fill the time gap between measurements and provide the robot with a path to follow while new plans are elaborated. In the receding horizon approach, the remote brain produces an obstacle free path at each remote brain program cycle. Every obstacle free path is based on the image obtained at the beginning of the program cycle. Thus the path remains valid from the time it was created until the end of the horizon. Here, the end of the horizon is defined as the goal. Therefore the path generated remains valid until the robot drifts away due to accumulated odometry error or due to the inaccuracy of the control actions (e.g. unbalanced responses of the motors). This is overcome when the robot receives the delayed visual information feedback and the newest path from the remote brain.

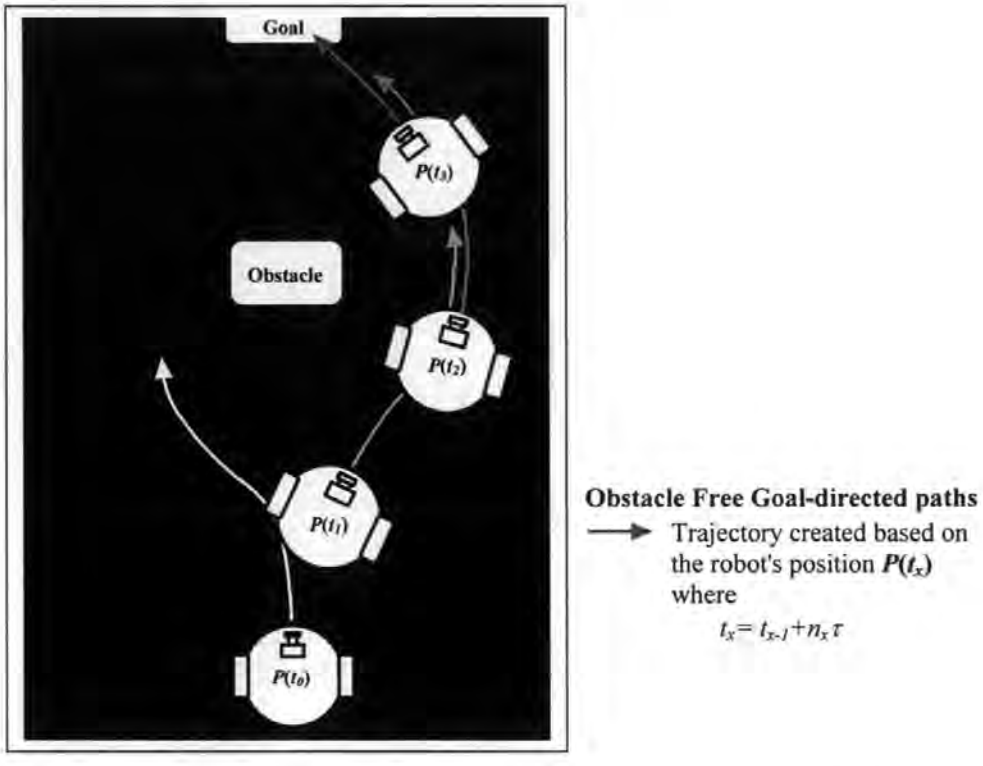


Figure 6.7: The concept of receding horizon control strategy. Using the Receding Horizon Strategy, the robot's remote brain is regularly searching for new paths toward the goal based on the robot latest coordinate (obtained through vision). Each line in figure here represents an obstacle free path $P(t_x)$ based on the image captured at time t_x.

Figure 6.7 illustrates the concept of receding horizon control strategy. Based on the receding horizon control strategy, an obstacle-free path is planned from the position where the sensory reading is made e.g. $P(t_0)$, to the goal. Theoretically, this path should enable the robot's controller to steer the robot towards the goal based on its odometric feedback. But in reality, there are many factors such as the accumulated odometry error and the inaccuracy of control actions that can cause the robot to drift away from its path. For example, the robot captures an image at position $P(t_0)$ at time t_0 and plans an obstacle-free path from position $P(t_0)$ to the goal. Let's assume that the robot then attempts to navigate along this path but drifts to the right and ends up at position $P(t_1)$. The path created at position $P(t_0)$ becomes invalid thus a new path bases on position $P(t_1)$ is needed. Therefore it is necessary for the remote brain to provide delayed visual information feedback and obstacle-free paths as often as possible (i.e. at every remote brain program cycle) to minimise the accumulated odometry errors and keep the robot on track. As illustrates in the figure 6.7, a new obstacle-free path is constantly created at each program cycle as the robot moves towards the goal. Thus, by continuously providing the robot with the latest visual feedback and a new obstacle-free path, the robot position can be continuously corrected, therefore minimizing the accumulated odometry errors and enabling the robot to reach its goal.

The obstacle-free path is encoded as 8 waypoints with a distance of 10cm between each waypoint. Figure 6.8 shows the process and implementation of the receding horizon control strategy using waypoints. As illustrated, the remote brain captures an image at time t_0 for high level processing. At the end of the high level processing at time t_1, the obstacle-free path produced is sent to the robot. Let's call this path ${}^{t0}T_{t1}$, indicating that the obstacle free path is created based on the image captured at time t_0 and becomes available at time t_1. This path ${}^{t0}T_{t1}$ is then sent to the robot in term of waypoints while the remote brain captures a new image to produce the path ${}^{t1}T_{t2}$. Since the robot has a limited memory

buffer, only four waypoints are sent. It is important to note that the waypoints sent must be able to support the robot's navigation process until the next obstacle-free path becomes available.

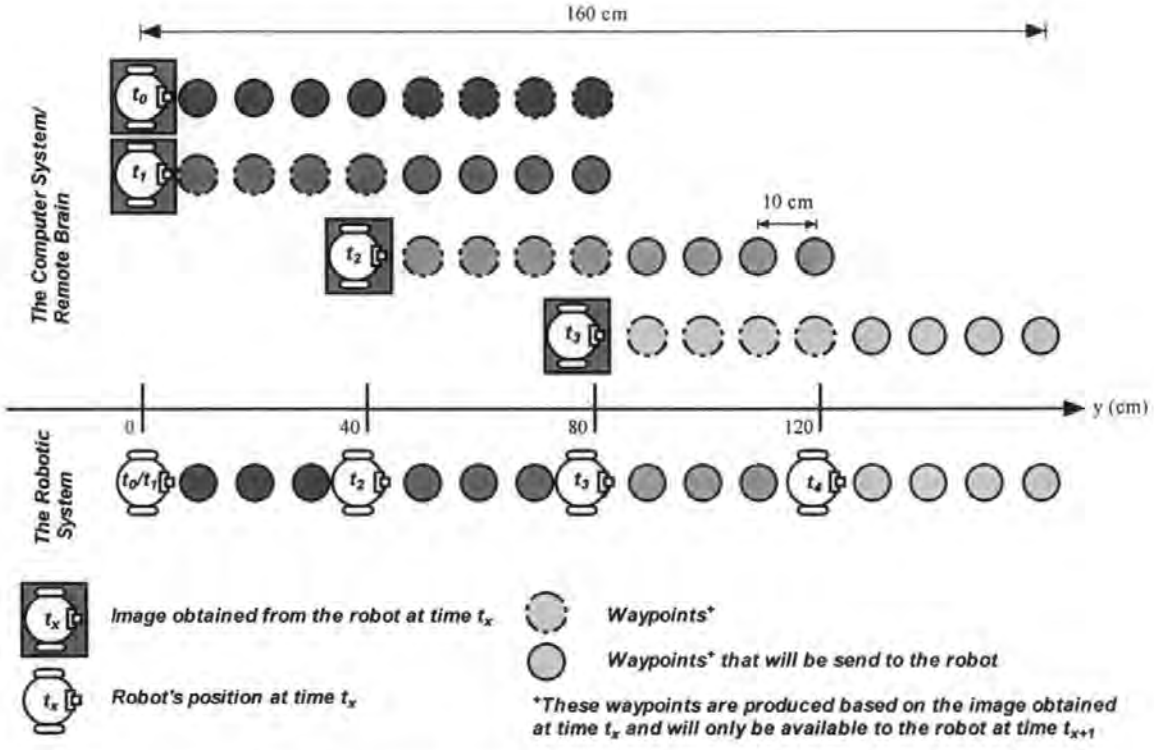


Figure 6.8: This figure illustrates the advantage of applying the receding horizon control strategy using the waypoints method.

Initially the robot is in a static state from t_0 to t_1, therefore the first four waypoints form the path ${}^{t0}T_{t1}$ are sent to the robot at t_1. Note that at time t_2, the last four waypoints of path ${}^{t1}T_{t2}$ are sent to the robot instead of the first four. This is because the robot only started to move at time t_1. Therefore the image captured at time t_1 is the same as the one captured at time t_0, and it is expected that the robot arrives at the fourth waypoints of path ${}^{t0}T_{t1}$ at time t_2.

As illustrated in figure 6.8, as the robot moves along the path from time t_2 onward, only the last four waypoints (i.e. waypoint 5, 6, 7 and 8) of a path are sent to the robot. This is due to the fact that the robot is in motion during the processing of high level tasks. When the path becomes available (i.e. with eight waypoints), the robot has already reached

the 4th waypoint. This procedure is essentially based on the observation that the remote brain needs $n\tau$ seconds for processing the high level tasks and that in that time the robot covers a distance of 30 to 40 cm. $n\tau$ refers to a number of low level control cycles, but is not a constant, as it depends on the image processing and planning complexity. It is of the order of 1 sec in our control system.

Planning of new paths stops if the second half of the waypoints sent to the robot are at the same location as the goal (i.e. the 7th and 8th waypoint's coordinates are the same as the goal's coordinate).

6.2.2 Modified Smith Predictor for Intermittent Delayed Feedback

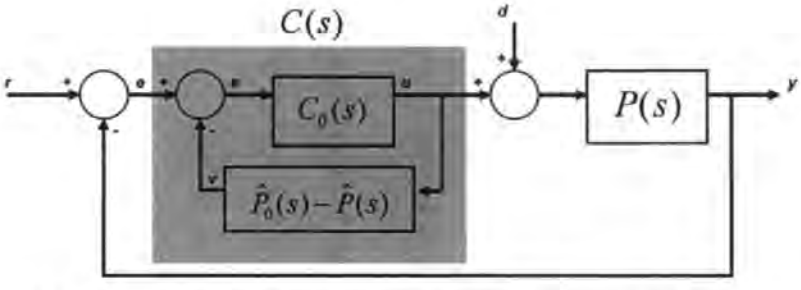


Figure 6.9: Classical diagram of a control system incorporating a Smith Predictor where r is the reference signal, $P(s)$ is the transfer function of the process with large dead-time, $\hat{P}(s)$ and $\hat{P}_0(s)$ are the process models with and without dead time respectively, the shaded area $C(s)$ is the Smith Predictor or Dead-time Compensators (DTC), $C_0(s)$ is the primary controller and d represents external disturbances.

The Smith Predictor is well known as an effective Dead-time Compensator (DTC) for a stable process with a large dead time (Smith, 1959). The classical configuration of a Smith Predictor is shows in figure 6.9. The presence of a large dead-time (i.e. $n\tau$) in the process $P(s)$ causes the feedback of $y(t)$ to be delayed and usually slows down the controls and causes the system to have a sluggish response or overcorrection associated with conventional controllers. The aim of the Smith Predictor is to improve this closed-loop performance. This is done by introducing a minor feedback loop around the primary

controller to produce $v(t)$, which is an estimation of the variation of $y(t)$ during the last $n\tau$ units of time. This variation $v(t)$ added to the delayed measurement constitute an estimate of the current value of $y(t)$. This is subtracted form the requested value r to produce the error e' that is fed into the controller. This eliminates the sluggish responses or overcorrection associate with conventional controllers (Levine, 1996).

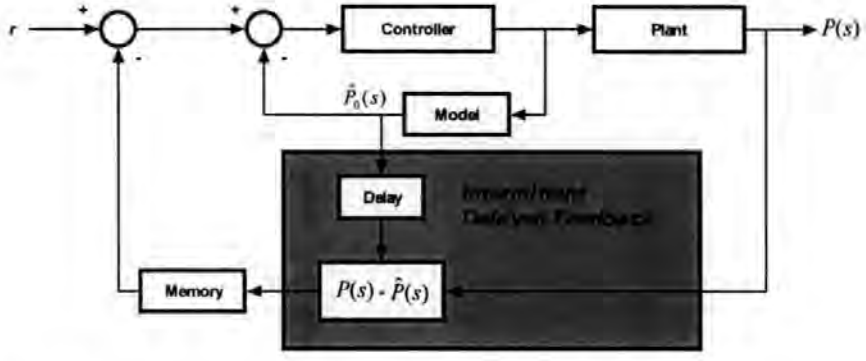


Figure 6.10: Modified Smith Predictor for Intermittent Delayed Feedback

The initial idea was to build a Smith Predictor into the vision-based navigation system to overcome the intermittent delayed feedback. This was not possible since the Smith predictor is developed for dealing with dead-time problems common to industrial process where feedback from the processes is continuous. Therefore, we proposed the modification of the Smith predictor for dealing with intermittent delayed feedback shown in figure 6.10. In the conventional Smith predictor a delayed copy $\hat{P}(s)$ of the model's output $\hat{P}_0(s)$ is in effect compared in each time step with the actual delayed measurement $P(s)$. The difference between the two provides a new correction factor at each time step. This is a form of retroactive updating where the error made at $n\tau$ time steps in the past is corrected in each time step τ.

In the modified Smith predictor, the delayed copy of the model's output can be compared with the actual measurement only every $n\tau$ time steps. Thus the same correction

factor has to be used until the next measurement becomes available. Retroactive updating is here of an intermittent nature, very similar to that proposed by other authors in a context different from that of the Smith Predictor (see discussion 6.5).

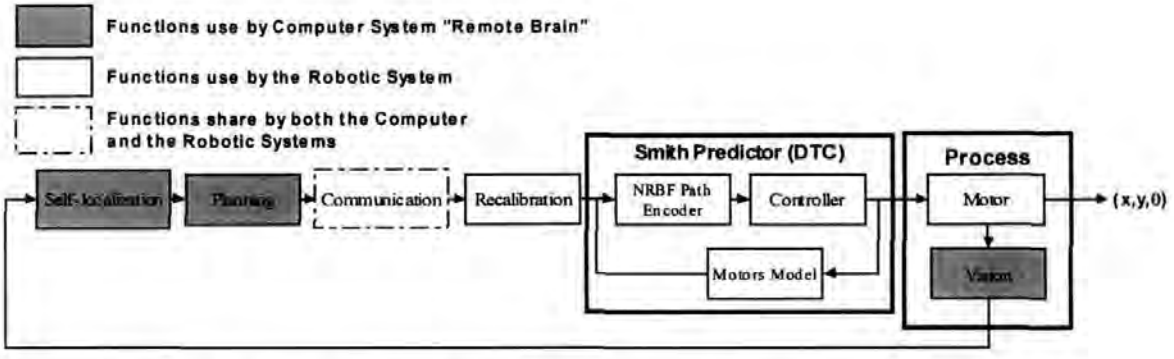


Figure 6.11: Operation sequence in the proposed system that uses a modified Smith Predictor.

The block diagram shows in figure 6.11 illustrates the proposed system with a modified Smith predictor. The initial approach was to use a dynamical model of the motors rather than the shaft encoder feedback because the original shaft encoders that came with the robot were inaccurate and often gave false readings, and the original Smith Predictor design suggests the use of a model to give feedback to the controller. A simplified version of the motor model was built; details on how this model was built are discussed in section 6.3.1. Note that the final system design is shown in figure 6.12, in this design we used the shaft encoders feedback rather than the model feedback because of two main reasons. The first was that we had developed more reliable shaft encoders. The second reason was that the model we created was difficult to use, as it did not take into account variations of the battery voltage level. This caused a mismatch between the model and the actual motors when the battery voltage level dropped over time.

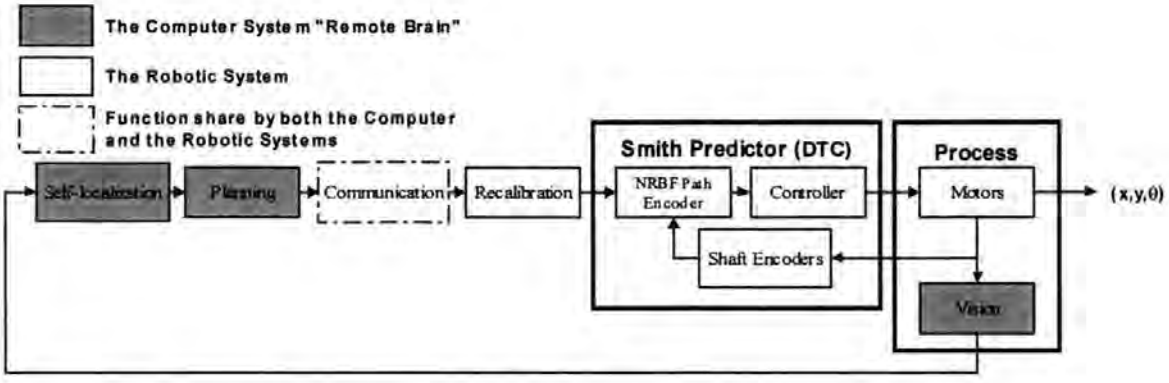


Figure 6.12: Operation Sequence in the final system inspired by Smith Predictor.

Figure 6.12 shows the final design of the system that was implemented. The modified Smith predictor here is responsible for keeping the robot along the obstacle free path with the help of the shaft encoders feedback. The shaft encoders are used to replace the motor model and provide a more accurate estimation of the distance travelled by the robot. These distance data are then used to estimate the robot position within its environment using the on-board motion tracking module which will be described in section 6.3.2.

6.3 Fast Feedback Loop in the Smith Predictor

6.3.1 Building a Dynamical Model of the Robot

Building the motor or robot model is done here by first collecting data of the robot that exhibit its dynamics and behaviour during motion and speed changes. The second step is to derive the robot model and determine the coefficients that will give the model a dynamics and behaviour similar to that of the actual robot.

Due to the spur gear type used by the motors, the inertia of the robot can be incorporated in a motor model. Hence we will refer to "motor model" for what is actually a model of the motors and the robot. All data are obtained from the robot moving in straight line.

6.3.1.1 Collecting Modelling Data

10 sets of data were obtained by running the robot in straight line in its environment where the speed was stepwise increased in each run from 0 to 50%, then from 50% to 75% and from 75% to 100%. These values represent the percentage of the full speed at which the motors can operate. Each dataset consists of 100 time interval values. The data recorded were the times (ms) spend between each tick of the right wheel shaft encoder. Knowing the distances travelled by the wheel between two ticks allows the average velocities (cm/ms) to be determined for each time interval. Figure 6.13 shows that the first 35 ticks cover the motors speed setting of 50, the next 35 ticks cover the speed setting of 75 and the last 30 ticks cover the speed setting of 100. A tick is an optically detected either white to black or black to white transition in the encoder pattern shown in figure 6.13.

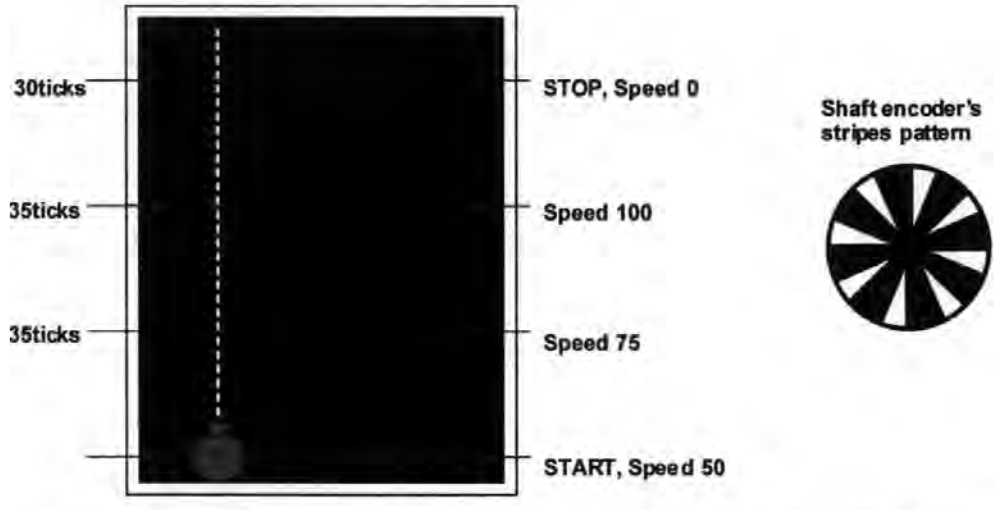


Figure 6.13: Illustration of the data collection protocol and an example of the encoder's strip pattern (i.e. black strip absorb light and white strip reflect light) glued to the wheel. A tick is a transition from black to white or from white to black detected by an Infra Red emitter/receiver when the wheel turns.

Once the data of the time intervals between ticks are collected, equation 6.1 is used to determine the instantaneous velocity between each tick.

$$v_{tick} = \frac{d_{tick}}{t_{tick}} \tag{6.1}$$

where v_{tick}, d_{tick} and t_{tick} represent the velocity, the distance travelled by the wheel and the time interval between each tick respectively. Since the distance travelled between each tick is constant and known, and the time between each tick is obtained from sensory measurements, the velocity between each tick can be determined easily. The results are plotted on the graph shown in figure 6.14.

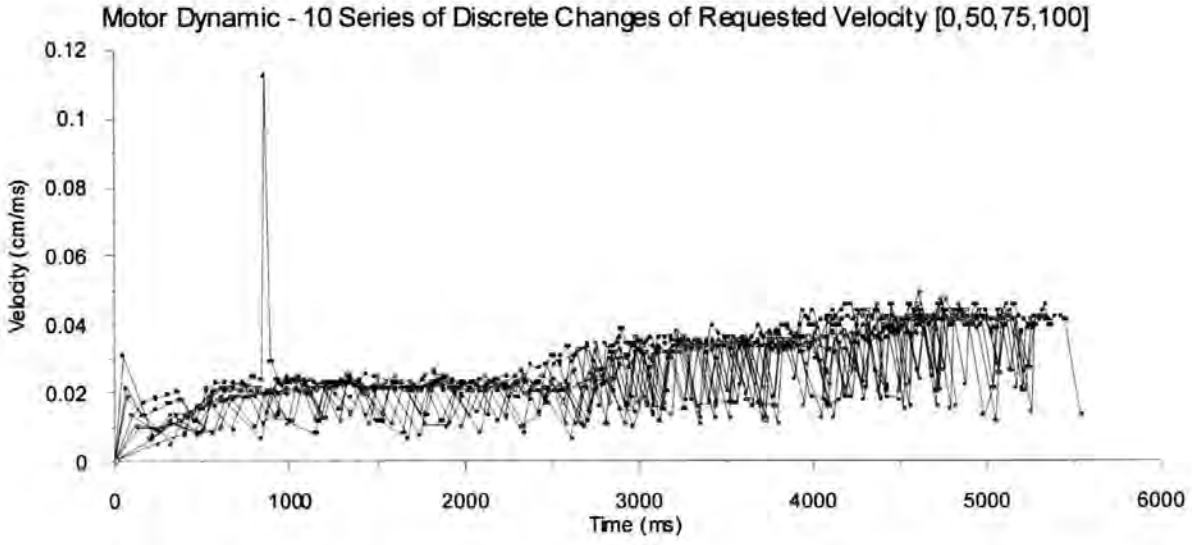


Figure 6.14: The motor dynamic plot for 10 runs. The downward jumps in instantaneous speed are due to missed shaft encoder ticks.

The data on figure 6.14 are quite noisy due to the unreliable detection of ticks by the optical shaft encoder. Most of the errors are due to a tick being missed. This apparently doubles the time interval between ticks and reduces the calculated velocity. This can be detected and corrected by using a simple algorithm (not described here). Some of the errors are due to non-existent transitions being detected between two ticks. This causes the single high-velocity peak in figure 6.14. This was removed from the data used to fit the model parameters. The data after correction can be seen in figure 6.15. The unreliability of the shaft encoder also caused a variability of the time at which the step changes in requested speed where applied. These time variations were manually corrected before the fitting process in 6.3.1.3

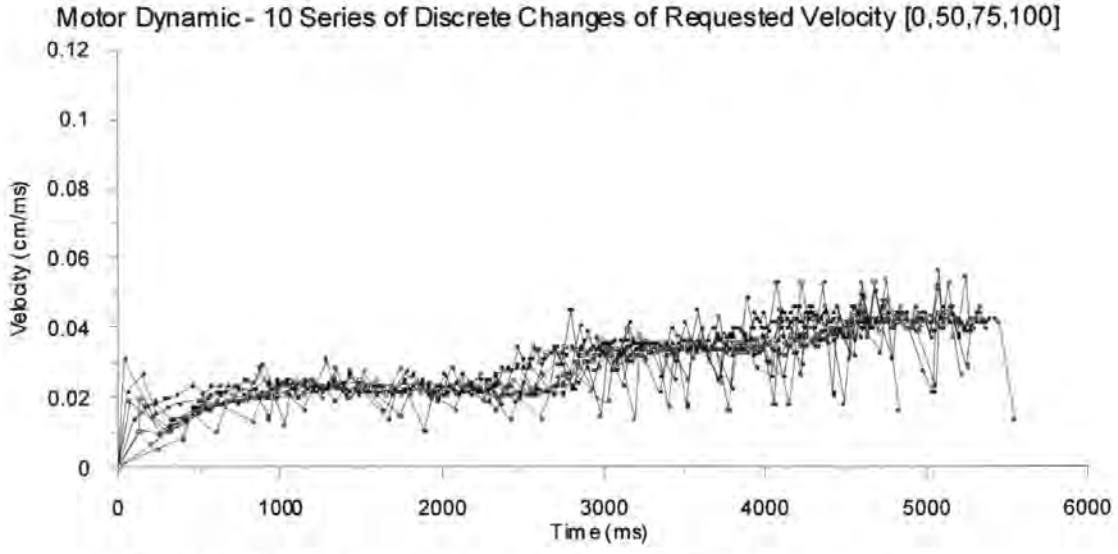


Figure 6.15: The motor dynamic plot for 10 runs after data correction.

6.3.1.2 Derivation of the Rug Warrior's Motion Model

According to Newton's Second Law, the sum of the external forces F on any object or collection of objects equals the product of total mass m and the acceleration a of the centre of mass.

$$\sum F = ma \tag{6.2}$$

In this case the only relevant forces on a flat plane are the traction F_T exerted by the motor and the frictional losses μF_N, where μ is the coefficient of friction and F_N is the normal force. Therefore the equation of motion can be defined as

$$ma = F_T - \mu F_N \tag{6.3}$$

Let v be the velocity of the robot, then equation (6.3) can be rewritten as

$$m\frac{dv}{dt} = F_T - \mu F_N \tag{6.4}$$

For a geared electric motor, the traction force is defines as

$$F_T = \frac{\Gamma G}{r_w} \tag{6.5}$$

where, Γ is the motor's torque, G is the gear ratio and r_w is the radius of the wheel. Applying this equation to equation (6.4) and obtain

$$m\frac{dv}{dt} = \frac{\Gamma G}{r_w} - \mu F_N \tag{6.6}$$

For a permanent-magnet dc motor where the magnetic field flux Φ_f is constant, the applied voltage V is related to the armature current I_a and the induced back-emf voltage E_a by (Mohan, Undeland and Robbins,1995, pp. 377-381, and Jones and Flynn 1993):

$$V = I_a R_a + E_a \tag{6.7}$$

The induced back-emf E_a increases proportionally with the angular velocity of the armature ω_m and the back-emf constant of the motor k_E:

$$E_a = k_E \omega_m \tag{6.8}$$

where

$$k_E = k_e \Phi_f \tag{6.9}$$

k_e being the voltage constant of the motor.

The torque Γ increases linearly with the armature current I_a and the torque constant of the motor k_T:

$$\Gamma = k_T I_a \tag{6.10}$$

where the torque constant k_T is proportional to the magnetic flux.

$$k_T = k_t \Phi_f \tag{6.11}$$

Therefore the applied voltage V can be rewritten as

$$V = \frac{\Gamma R_a}{k_T} + k_E \omega \tag{6.12}$$

from which the torque Γ becomes

$$\Gamma = V\frac{k_T}{R_a} - \omega\frac{k_E k_T}{R_a} \tag{6.13}$$

In a steady state, the electrical power P_e of the motor is equal to the mechanical power P_m of the motor:

$$P_e = P_m \tag{6.14}$$

where

$$P_e = E_a I_a = k_E \omega_m I_a \tag{6.15}$$

and

$$P_m = \omega_m \Gamma = \omega_m k_T I_a \tag{6.16}$$

therefore

$$k_E = k_T \tag{6.17}$$

By defining

$$k = k_E = k_T$$

equation 6.13 becomes

$$\Gamma = V\frac{k}{R_a} - \omega_m\frac{k^2}{R_a} \tag{6.18}$$

The angular velocity ω_m of the motor is related to the displacement velocity v of the wheel of the robot by the gear ratio

$$\omega_m = \frac{v}{r_w}G \tag{6.19}$$

Therefore equation (6.18) can be rewrite as

$$\Gamma = V\frac{k}{R_a} - v\frac{k^2}{R_a r_w}G \tag{6.20}$$

Applying the torque equation (6.20) to equation (6.6)

$$m\frac{dv}{dt} = V\frac{kG}{R_a r_w} - v\frac{k^2G^2}{R_a r_w^2} - \mu F_N \tag{6.21}$$

and defining C_1 and C_2 as followed

$$C_1 = \frac{kG}{R_a r_w}$$

$$C_2 = \frac{k^2 G^2}{R_a r_w^2}$$

one obtains

$$\frac{dv}{dt} = V\frac{C_1}{m} - v\frac{C_2}{m} - \frac{\mu F_N}{m} \tag{6.22}$$

By defining P and Q as followed

$$P = \frac{C_2}{m} \tag{6.23}$$

$$Q = V\frac{C_1}{m} - \frac{\mu F_N}{m} \tag{6.24}$$

and equation (6.22) becomes

$$\frac{dv}{dt} = Q - Pv \tag{6.25}$$

$$\frac{dv}{dt} + Pv = Q \tag{6.26}$$

this is a Linear Differential Equation of 1st order with solution (see Appendix A).

$$v(t) = \frac{Q}{P}(1 - e^{-P[t-t_0]}) + Ce^{-P[t-t_0]} \tag{6.27}$$

Defining α as

$$\alpha = \frac{Q}{P} \tag{6.28}$$

equation 6.27 can be rewritten as

$$v(t) = \alpha(1 - e^{-P[t-t_0]}) + Ce^{-P[t-t_0]} \tag{6.29}$$

where α is the maximum steady-state velocity achieved by the robot for a given applied voltage. The steady-state is reached with a time constant $\frac{1}{P}$. As P is a constant, α is expected to increase linearly with the voltage V (see equation 6.24). However, there is a

threshold voltage due to the friction force. C is the initial velocity at time t_0 (see figure 6.16).

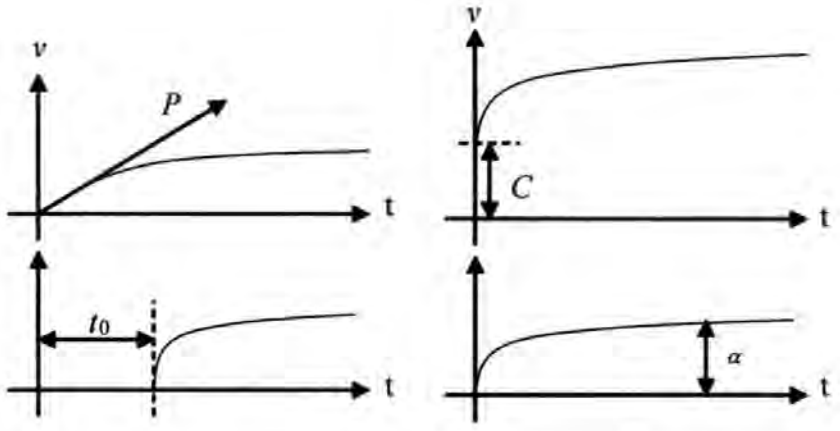


Figure 6.16: This figure illustrates the relation between each of the motor coefficients and their effect on the model output.

6.3.1.3 Parameters Fitting

Before the curve fitting process, the experimental data are divided into three sets based on their speed setting; the first set covers the data for speed setting 50, the second set covers the data for speed setting 75 and the third set covers for speed setting 100. Each set of the data is then curve-fitted separately using equation 6.29 to obtain their coefficient α, P and C. The coefficients of each set of the experimental data are show in Table 6.1.

Speed Command	α	P	C
0-50	0.022515	0.003872	0
50-75	0.034307	0.003872	0.022515
75-100	0.042882	0.003872	0.034307

Table 6.1: The coefficients for each of the command speed obtained through curve fitting.

Figure 6.17 shows the model's output and the modelling data for speeds setting of 50, 75 and 100 in solid line and dots respectively. The model's coefficients were those shown in table 6.1.

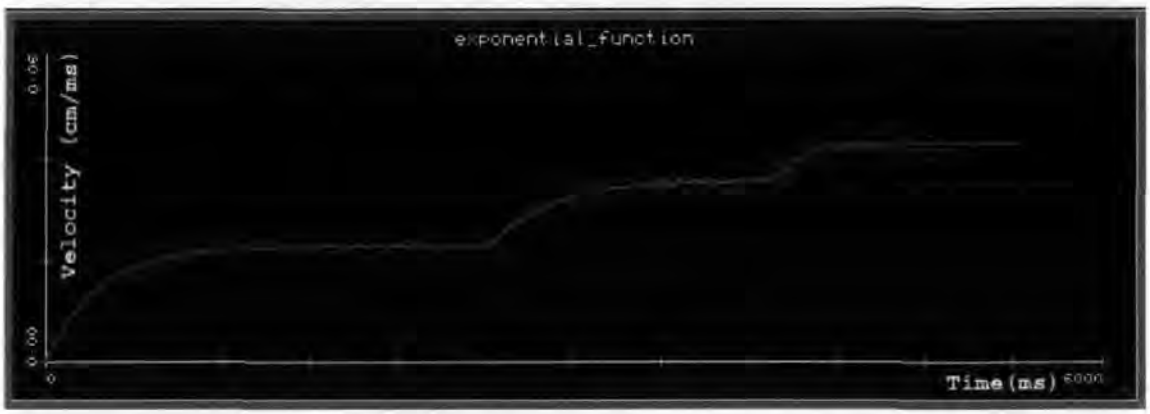


Figure 6.17: This figure shows the plot of data collected from the motor (dots) and the motor model (solid line) using the coefficients obtained through curve fitting (see Table 6.1).

The coefficients shown in table 6.1 are only applicable for speed settings of 50, 75 and 100. To obtain the values of the coefficient α for intermediate values of the set speed, the following interpolation functions are used:

$$\alpha(\delta) = -0.0000000957941 82\delta^3 + 0.000014526292669^2 - 0.0000627823917 99\delta - 0.000719846172773 \quad (6.30)$$

Figure 6.18 shows the plot of the α (for the set speeds of 50, 75 and 100), and the function $\alpha(\delta)$ for the set speeds δ=1…100.

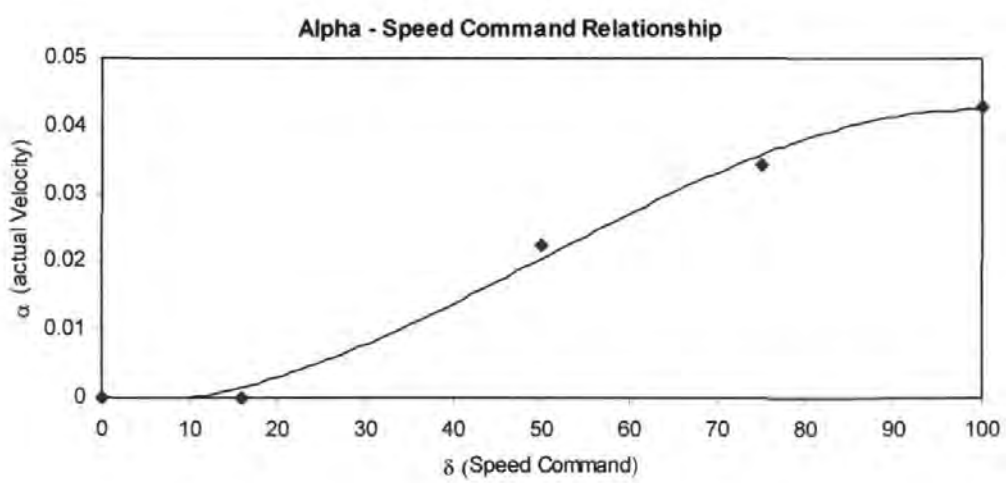


Figure 6.18: The actual maximal velocity as a function of the speed commands.

Then equation 6.29 can then be rewritten as:

$$v(t) = \alpha(\delta)(1 - e^{-P[t-t_0]}) + v_{t_0} e^{-P[t-t_0]} \tag{6.31}$$

where

$v(t)$ is the velocity at time t.

v_{t0} is the velocity at time t_0.

$\alpha(\delta)$ is the maximal velocity for the set speed δ.

P is the zero speed acceleration.

Note that equations 6.30 and 6.31 are used to determine the velocity of each wheel in each time step of the fast control loop.

The distance travelled by each wheel in each time step within the fast control loop, is obtained by integrating the equation 6.31:

$$\begin{aligned} s(t) &= \int_{t_0}^{t} v(t')dt' = \int_{t_0}^{t} [\alpha(\delta) - e^{-Pt'}(\alpha(\delta) - v_0)]dt' \\ &= \alpha \int_{t_0}^{t} 1dt' - (\alpha - v_0) \int_{t_0}^{t} e^{-Pt'} dt' \\ &= \alpha(t - t_0) - (\alpha - v_0) \frac{e^{-Pt} - e^{-Pt_0}}{-P} \end{aligned} \tag{6.32}$$

where S is the distance travelled by the wheel from the time when the initial velocity at time t_0 was v_0. Details of how displacement information obtained from the model is used to track the robot's path are given in the next section. The model provided accurate self-tracking information as long as the level of its batteries remained stable (figure 6.19). However, when batteries where allowed to discharge the behaviour became unreliable.

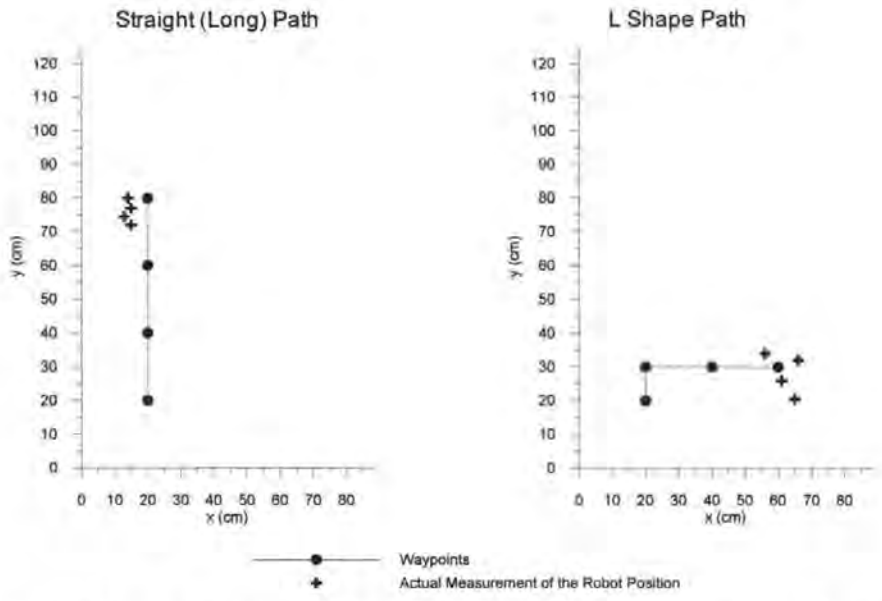


Figure 6.19: The test results of the robot following two prescribed paths based on the motors model as feedback. Note that the robot starting point is at the first waypoint (20,20) and that the reliability of the model is judged by how close the actual measurement of the robot position is to the target point (final waypoint).

6.3.2 Odometric Motion Tracking

Another way to determine the distance travelled by the robot is to use direct readings from the shaft encoders. However, these need to be more reliable than the ones provided in the Rug Warrior kit. For that reason new rigid encoder disks were built and attached to the wheels in place of the original adhesive foils. Self-tracking can then be performed by integrating the information obtained from the robot's shaft encoders (i.e. the distances travelled by each wheel). This section will discuss the self-tracking formula for a straight forward motion (section 6.3.2.1) and a curved motion (section 6.3.2.2). These formulas are then used to determine the robot's position and orientation based on the shaft encoders information.

6.3.2.1 The Robot in Straight Motion

During forward motion, the robot is programmed to move forwards in a straight line with both shaft encoders expected to show the same counter values (right wheel shaft encoder counter value C_R is expected to be equal to the left wheel shaft encoder counter

value C_L). The distances (in centimetre) travelled by the robot left wheel d_L and right wheel d_R were each obtained by multiplying their counter value (i.e. the number of ticks) with the distance per tick factor f_{dpt} as shown in equation 6.33 and 6.34.

$$d_L = C_L \times f_{dpt} \tag{6.33}$$

$$d_R = C_R \times f_{dpt} \tag{6.34}$$

6.3.2.2 The Robot in a Curved Motion

This section starts by considering the case of a leftward curve, and then adapts the equations for a rightward curve.

During leftward motion, the robot is programmed to move towards its left with both shaft encoder counters expecting to have different values. The distance travelled by the robot right wheel d_R should be greater than the distance travelled by the robot left wheel d_L as shows in figure 6.20. The actual distance travelled by the centre of the robot is shown in figure 6.20 as curve d_m and can be represented by Δx_{TR} and Δy_{TR} as the actual distances travelled in the x and y direction respectively. Δx_{TR} and Δy_{TR} are obtained using equation 6.44 and 6.45. In order to solve Δx_{TR} and Δy_{TR}, we first determine the change in the robot's orientation $\Delta\theta_R$ using equation 6.41 and the distance R_m from the robot's origin to the *ICC* (Instantaneous Centre of Curvature) of the robot at the start of the curve motion using equation 6.43. Below is shown how these equations were derived for the conceptual diagram shown in figure 6.20.

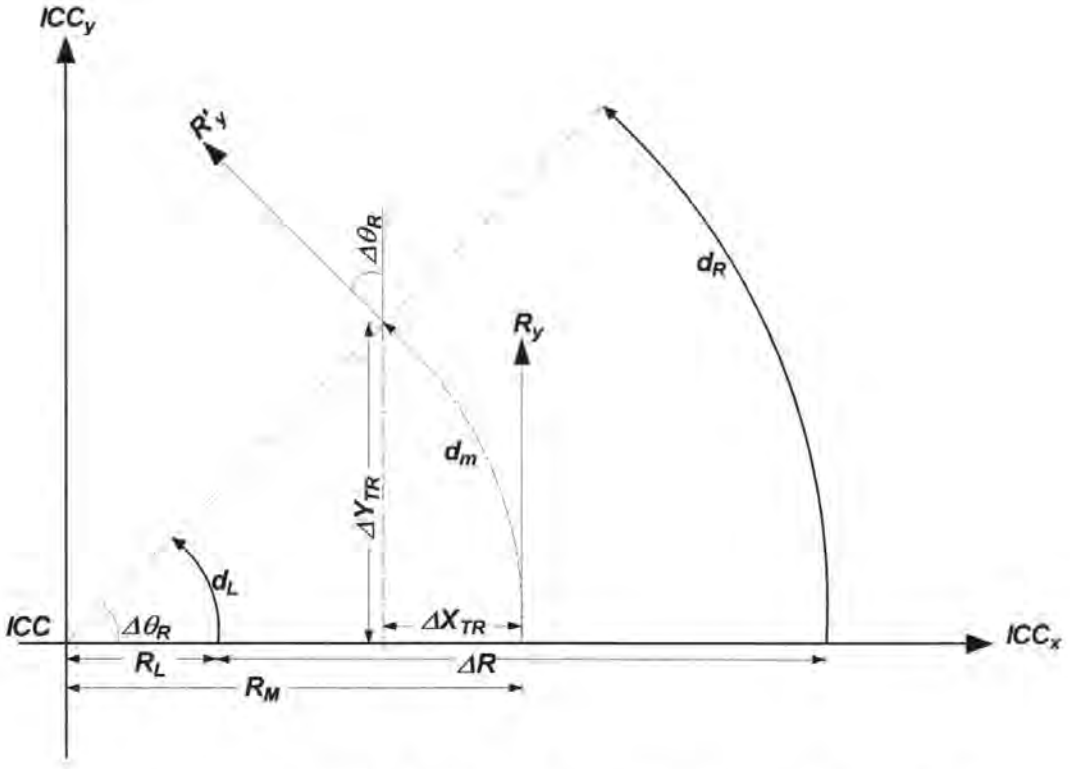


Figure 6.20: Conceptual diagram for the robot doing a leftward motion. The Instantaneous Centre of Curvature (*ICC*) shown is the centre of rotation for the robot and both its right and left wheels when the robot follows a circular motion. Based on this concept, the robot's new position can be found provided that distances travelled by both the robot's wheels (d_L and d_R) are known.

From figure 6.20, knowing the distance travelled by both wheels (d_L and d_R) and the distance between both wheels ΔR, the change of robot orientation $\Delta\theta_R$ can be obtained using equation 6.41.

$$d_m = \frac{(d_L + d_R)}{2} \tag{6.35}$$

$$d_L = R_L \Delta\theta_R \tag{6.36}$$

$$R_L = \frac{d_L}{\Delta\theta_R} \tag{6.37}$$

$$d_R = R_R \Delta\theta_R \tag{6.38}$$

$$R_R = R_L + \Delta R \tag{6.39}$$

By substituting equation 6.37 into equation 6.39, we obtain equation 6.40

$$R_R = \left(\frac{d_L}{\Delta\theta_R}\right) + \Delta R \tag{6.40}$$

and by substituting equation 6.40 into equation 6.38, we obtain the change of the robot angle $\Delta\theta_R$ as shown in equation 6.41.

$$d_R = \left(\frac{d_L}{\Delta\theta_R} + \Delta R\right)\Delta\theta_R$$
$$= d_L + \Delta R\Delta\theta_R$$

Therefore

$$\Delta\theta_R = \frac{(d_R - d_L)}{\Delta R} \tag{6.41}$$

The distance R_m from the robot's origin to the *ICC* was obtained using equation 6.43. This equation was obtained by substituting equation 6.37 into equation 6.42.

$$R_m = R_L + \left(\frac{\Delta R}{2}\right) \tag{6.42}$$

$$R_m = \left(\frac{d_L}{\Delta\theta_R}\right) + \left(\frac{\Delta R}{2}\right) \tag{6.43}$$

From figure 6.20 we solve Δx_{TR} and Δy_{TR}

$$\Delta x_{TR} = R_m - R_m\cos(\Delta\theta_R) \tag{6.44}$$

$$\Delta y_{TR} = R_m\sin(\Delta\theta_R) \tag{6.45}$$

Equations 6.41, 6.43, 6.44 and 6.45 are the actual equations used to determine Δx_{TR} and Δy_{TR}. Although these equations were derived based on the conceptual diagram for the robot doing a leftward motion, they are also applicable when the robot is doing a rightward motion. When the robot is doing a rightward motion, $\Delta\theta_R$ will be negative (clockwise) and this causes R_m to be negative. Therefore Δx_{TR} will become negative.

6.3.2.3 Conversion to the Map Coordinate System

This section illustrates the transformation of the robot position from the robot coordinate system to the map coordinate system. Figure 6.21 shows the transformation diagram.

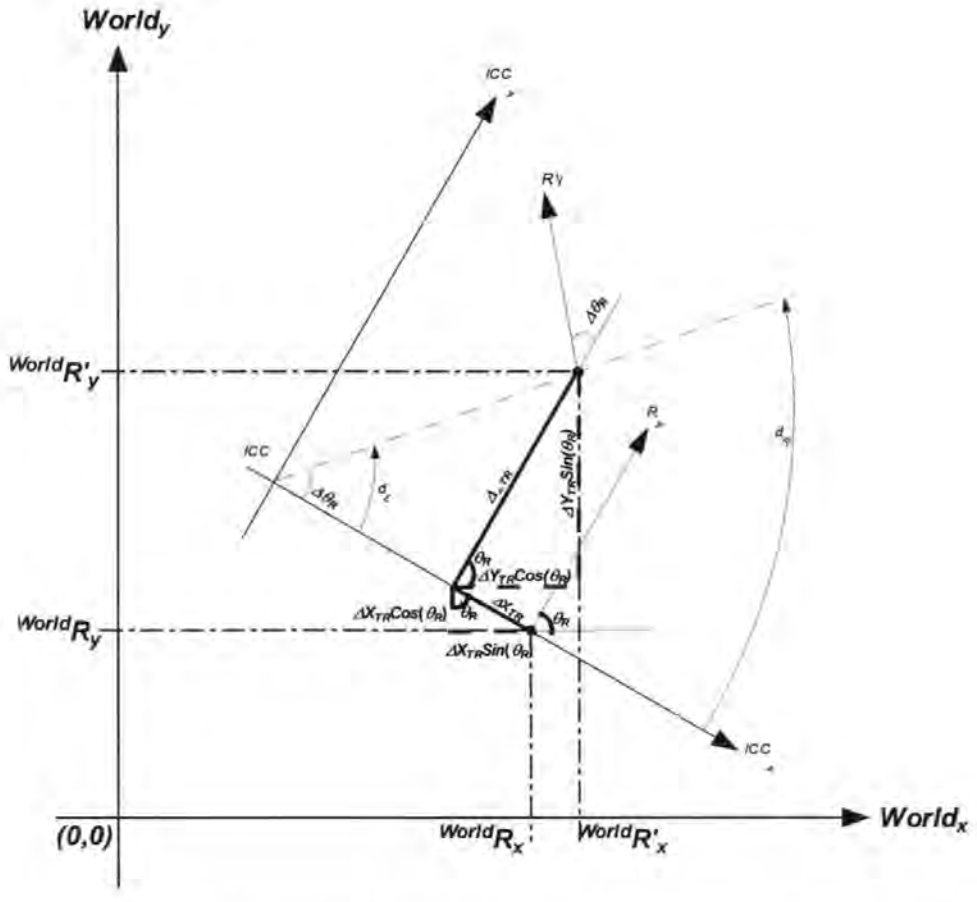


Figure 6.21: Conceptual diagram used for updating the robot position in the model map.

Previously, sections 6.3.2.1 and 6.3.2.2 illustrated the methods used to track the distance travelled by the robot from its old position to its new position in the robot old position coordinate system. This section will illustrate the methods used to transform the robot's new position from the robot's old position coordinate system to the map coordinate system.

The actual distance travelled by the robot in the x direction Δx_{TR} and y direction Δy_{TR} was obtained from section 6.3.2.2. If the robot was programmed to travel in a straight path followed by a turn, the distance travelled by the robot in the robot old coordinate system can be defined by the equation 6.46 for x_{TR} and equation 6.47 for y_{TR}.

$$x_{TR} = \Delta x_{TR} \tag{6.46}$$

$$y_{TR} = y_{TStraight} + \Delta y_{TR} \tag{6.47}$$

If the robot was performing a curved path, then $y_{Straight}$ will be zero (equation 6.48) and equation 6.47 becomes equation 6.49.

$$y_{TStraight} = 0 \tag{6.48}$$

$$y_{TR} = 0 + \Delta y_{TR} \tag{6.49}$$

Before placing the robot's new position onto the map, equation 6.50 and equation 6.51 are used to transform the robot new position with respect to the robot's old position coordinate frame to the map coordinate frame. The output from equations 6.50 and 6.51 is then registered onto the map as the robot's current position. Equation 6.52 is then used to determine the new orientation.

$$Robot_{x,new} = Robot_{x,old} + y_{TR}\cos(\theta_R) - x_{TR}\sin(\theta_R) \tag{6.50}$$

$$Robot_{y,new} = Robot_{y,old} + y_{TR}\sin(\theta_R) + x_{TR}\cos(\theta_R) \tag{6.51}$$

$$Robot_{\theta,new} = Robot_{\theta,old} + \Delta\theta_R \tag{6.52}$$

6.4 On-board Path Control

This section describes the primary controller on the robotic system. The NRBF encoder (see 5.2.2 & 5.2.3) is used based on the robot current position to provide the robot with a sub-target along the obstacle-free path. The sub-target is then used to calculate the necessary velocities needed by both the right and left wheels to drive the robot towards that sub-target. This process is repeated until the robot reaches its final destination (goal).

The algorithm begins by searching for the distance from the robot to the sub-target. This is done by using equations 6.53 to 6.55.

$$\Delta x = T_x - Robot_x \tag{6.53}$$

$$\Delta y = T_y - Robot_y \tag{6.54}$$

$$Dist = \sqrt{\Delta x^2 + \Delta y^2} \tag{6.55}$$

Then it determines the direction where the sub-target is located based on the robot's current position using equation 6.56. Using equation 6.57, it determines if the robot has to turn clockwise or anticlockwise to reach the target.

$$\delta = \tan^{-1}\left(\frac{\Delta x}{\Delta y}\right) \tag{6.56}$$

$$Rot_\theta = \delta - Robot_\theta \tag{6.57}$$

As the distance is used to determine the speed for the robot to navigate at, it is necessary to normalise the distance *Dist* so that the appropriate maximum speed for the robot to work at can be set. In this case the chosen top speed is 45 (out of the maximum of 100 allowed by the robot's hardware), and the robot will navigate at this speed when the distance between its current position and its sub-target is larger than 20 cm.

If Dist > 20

$$Dist = 20$$

$$LinearVel = 20.0 + 25.0\left(\frac{Dist}{20}\right) \tag{6.58}$$

The necessary speed required by the wheels for the robot to make a turn is then calculated based on the robot's desired orientation change.

Case $Rot_\theta < 0$

$$SpeedL = LinearVel \tag{6.59}$$

$$SpeedR = F\max\left[0, LinearVel\left(1 - \left(\frac{Rot\theta}{45}\right)^2\right)\right] \tag{6.60}$$

For $Rot_\theta > 0$

$$SpeedR = LinearVel \tag{6.61}$$

$$SpeedL = F\max\left[0, LinearVel\left(1 - \left(\frac{Rot\theta}{45}\right)^2\right)\right] \tag{6.62}$$

These speeds are sent to the motors to drive the robot toward its target.

The performance of the controller was tested with three different paths. The results are shown in figure 6.22.

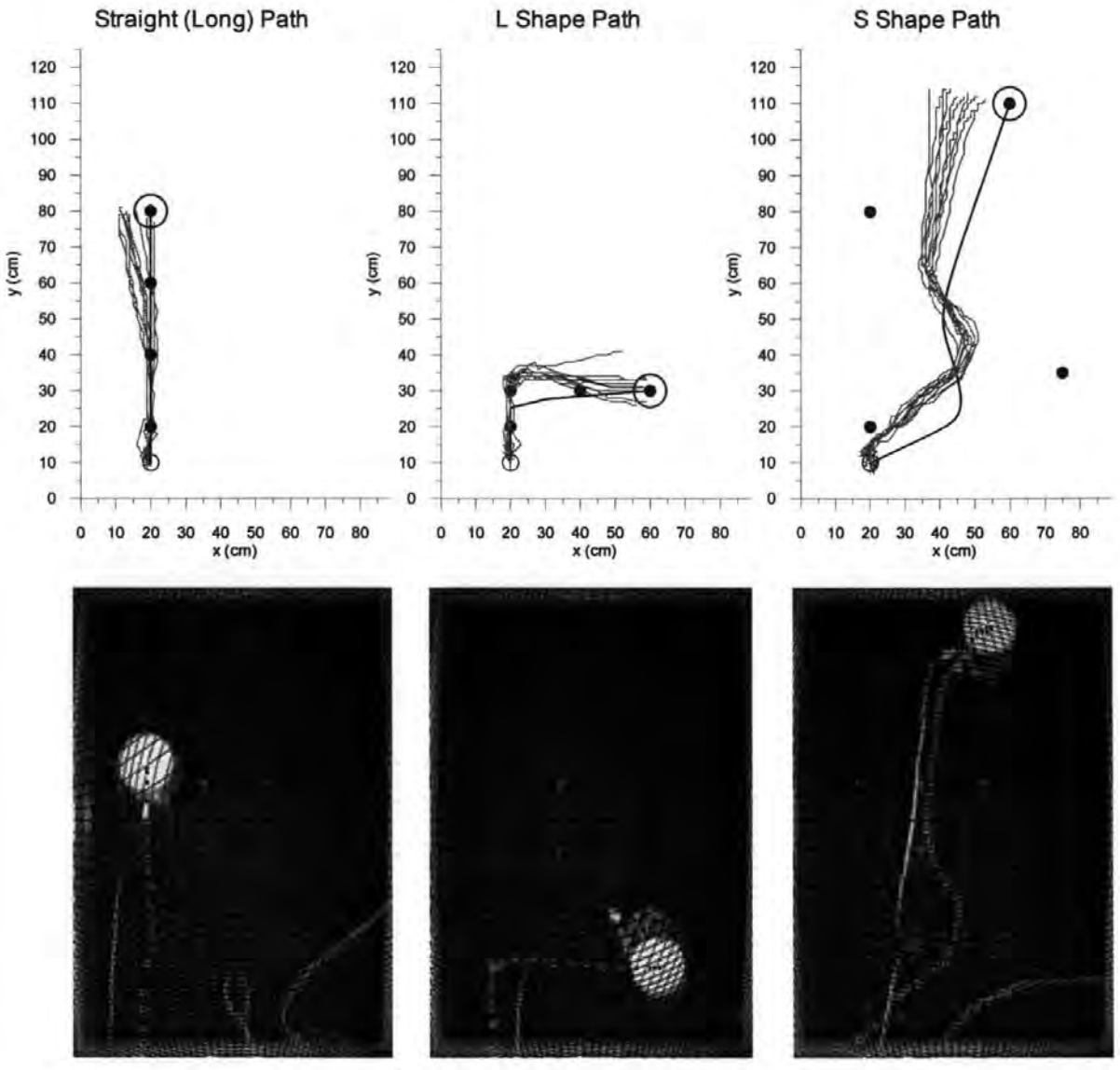


Figure 6.22: The result of the controller steering the robot from the initial position toward the goal based on the shaft encoders input as feedback. The first row shows the 10 recorded robot paths (in grey lines), the path created by the NRBF path encoder for the robot to follow (black lines) and the encoded waypoints (black dots). The second row shows the recorded path of a single run as a series of open circles superposed on the image from the robot's initial position to the robot's final position.

Results from the tests show that the performance of the controller guiding the robot along three different paths based only on the shaft encoders as feedback in the fast feedback loop is not significantly better than that using the dynamical model in the previous section (figure 6.19). This indicates that the new encoder disks did not solve all

the reliability problems. However, their measurements are not dependent on the battery charge level and it will therefore be used in the remainder of this thesis.

6.5 Retroactive Position Calibration Using Visual Feedback

This section deals with the robot's coordinate recalibration process based on the feedback from the robot's remote brain. The recalibration process is necessary since the robot relies on its shaft encoders (which drift with time) to keep track of its own position in the real world. Without the recalibration process, the robot will deviates from its path while "believing" that it is still on the obstacle-free path towards the goal.

6.5.1 Recalibration Equations

The aim of the coordinate recalibration process is to recalibrate the estimated current robot's position held on-board the robot (i.e. $\hat{P}(t_0 + n\tau)$) as often as possible, based on the robot's position obtained from the robot's remote brain (i.e. $P(t_0)$) which was determined from the image captured at time t_0. The idea is to record the positions $\hat{P}(t_0)$ and $\hat{P}(t_0 + n\tau)$ assumed by the robot at time t_0 and $t_0+n\tau$ respectively. Note that at time t_0, the remote brain captures an image to determine the robot's position and to plan an obstacle-free path. This information becomes available to the robot at time $t_0+n\tau$. By comparing the recorded robot position $\hat{P}(t_0)$ at time t_0 with the robot's position $P(t_0)$ derived from the image captured at time t_0, the shaft encoder drift having occurred up to the time t_0 becomes known. Compensating for this error will not ensure exact position knowledge of time $t_0+n\tau$ but limits the error to the possible drift having occurred since t_0.

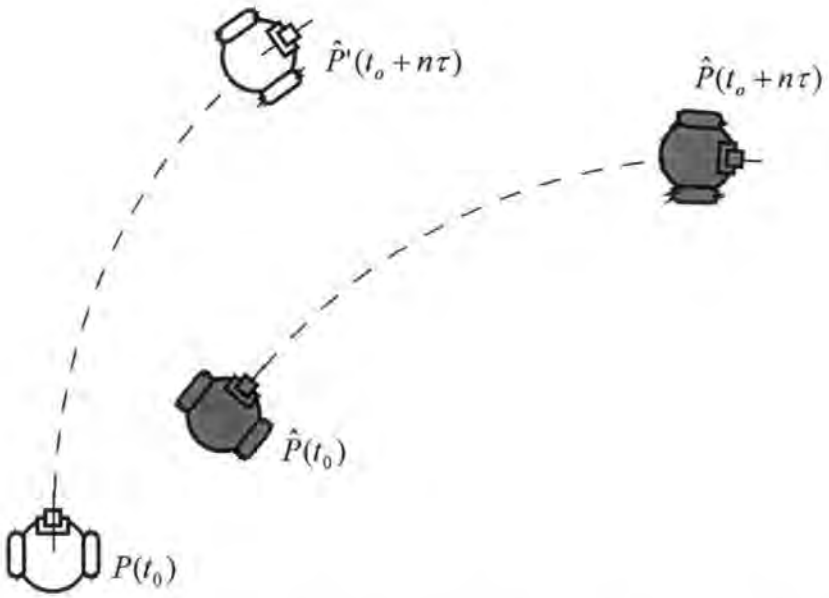


Figure 6.23: This figure illustrates the concepts of coordinate recalibration. The grey robots indicated by $\hat{P}(t_0)$ and $\hat{P}(t_0+n\tau)$ are the robot positions determined from the shaft encoders feedback at time t_0 and at time $t_0+n\tau$ respectively. The white robot indicated by $P(t_0)$ is the robot's actual position at time t_0 determined from the image obtained at time t_0 and that only becomes available to the robot at time $t_0+n\tau$. $\hat{P}'(t_0+n\tau)$ is the estimate of the current robot position at time $t_0+n\tau$ after recalibration.

The proposed recalibration algorithm is derived based on the following prior knowledge:

- the robot coordinate $P(t_0)$ of time t_0 obtained from the remote brain at time $t_0+n\tau$ is only true at the time when the image is captured at time t_0 (Since extracting information from the image and transfer it to the robot took some time, therefore by the time the robot receives this information, it is no longer valid, as the robot has already left the location where the image was obtained),

- there exist a relationship between the robot coordinate $\hat{P}(t_0)$ recorded at time t_0 and the robot coordinate $P(t_0)$ derived from the image captured at time t_0,

- hence there also exist a relationship between the robot coordinate $P(t_0)$ and robot current coordinate $\hat{P}(t_0+n\tau)$ since a relationship can be established between $\hat{P}(t_0)$ and $\hat{P}(t_0+n\tau)$.

Therefore by knowing the difference between $P(t_0)$ and $\hat{P}(t_0)$, and the distance travelled from $\hat{P}(t_0)$ to $\hat{P}(t_0 + n\tau)$, a better estimate of the robot's current position $\hat{P}'(t_0 + n\tau)$ can be determined. Two conceptual diagrams are shown in figure 6.24 and figure 6.25. These two diagrams are used to derive the coordinate recalibration formulas. The first diagram assumes that the coordinates $P(t_0)$ and $\hat{P}(t_0)$ are located at the same location but with different orientations. The second diagram assumes that the coordinates $P(t_0)$ and $\hat{P}(t_0)$ are located at different location but with similar orientations. Note that although these formulas were derived separately, they can be used together. In the actual application, they are used together since the difference between $P(t_0)$ and $\hat{P}(t_0)$ always involved their position and orientation.

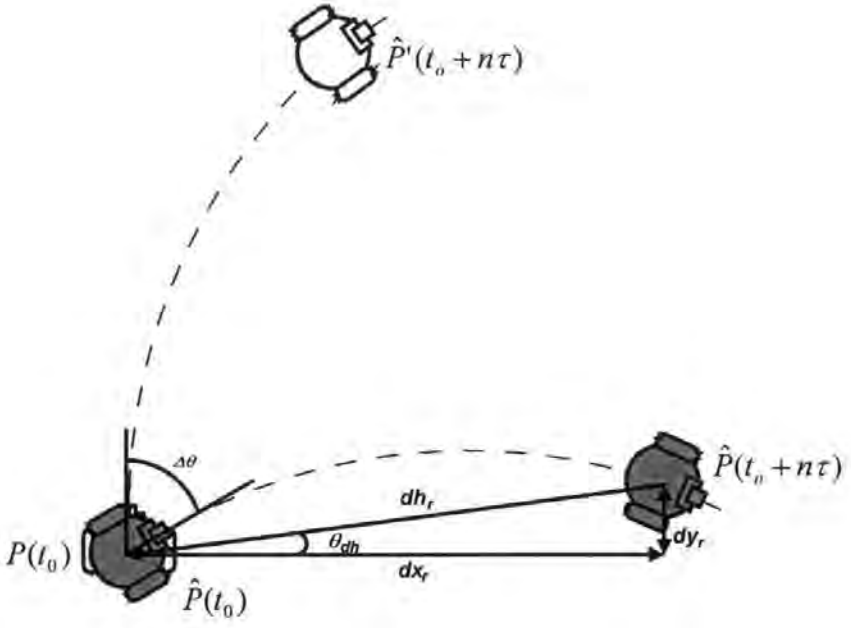


Figure 6.24: Diagram for the robot orientation recalibration algorithm.

Figure 6.24 shows the conceptual diagram where $P(t_0)$ and $\hat{P}(t_0)$ are located at the same position but with different orientation. The robot believes it has reached $\hat{P}(t_0 + n\tau)$ while it actually has reached $\hat{P}'(t_0 + n\tau)$. Based on the conceptual diagram, the actual robot position $\hat{P}'(t_0 + n\tau)$ at time $t_0+n\tau$ can be determined by using the equations shown below.

Firstly the relationship between $\hat{P}(t_0)$ and $\hat{P}(t_0 + n\tau)$ has to be established. This is done by using the equations below.

$$dx_r = \hat{P}(t_0 + n\tau).x - \hat{P}(t_0).x \tag{6.63}$$

$$dy_r = \hat{P}(t_0 + n\tau).y - \hat{P}(t_0).y \tag{6.64}$$

$$d\theta_r = \hat{P}(t_0 + n\tau).\theta - \hat{P}(t_0).\theta \tag{6.65}$$

$$dh_r = \sqrt{dx_r^{\,2} + dy_r^{\,2}} \tag{6.66}$$

$$\theta_{dh} = \tan^{-1}\left(\frac{dy_r}{dx_r}\right) \tag{6.67}$$

Knowing $P(t_0).\theta$ and $\hat{P}(t_0).\theta$, equation 6.68 is used to determined the angle difference $\Delta\theta$. The rotational angle θ_{rot} for dh_r when $\hat{P}(t_0)$ is rotated by $\Delta\theta$ can then be determined using equation 6.69

$$\Delta\theta = P(t_0).\theta - \hat{P}(t_0).\theta \tag{6.68}$$

$$\theta_{rot} = \theta_{dh} + \Delta\theta \tag{6.69}$$

Given that the distance dh_r and its rotational angle θ_{rot}, the coordinate $\hat{P}'(t_0 + n\tau)$ can be determined using equation 6.70 and equation 6.71, and its orientation using equation 6.72.

$$\hat{P}'(t_0 + n\tau).x = P(t_0).x + dh_r \cos(\theta_{rot}) \tag{6.70}$$

$$\hat{P}'(t_0 + n\tau).y = P(t_0).y + dh_r \sin(\theta_{rot}) \tag{6.71}$$

$$\hat{P}'(t_0 + n\tau).\theta = \hat{P}(t_0).\theta + d\theta_r \tag{6.72}$$

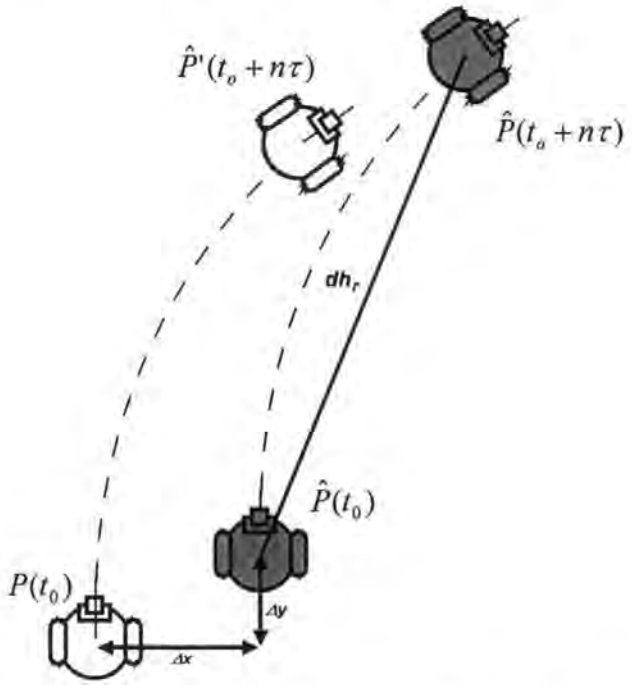


Figure 6.25: Diagram for the robot coordinate recalibration algorithm.

Figure 6.25 shows the concept diagram where $P(t_0)$ and $\hat{P}(t_0)$ are at different locations but have the same orientation. Therefore $\hat{P}'(t_0 + n\tau)$ can be obtain by determine the offset Δx of x axis and the offset Δy of y axis of $\hat{P}(t_0)$ relative to $P(t_0)$, and translates $\hat{P}(t_0 + n\tau)$ by offsets Δx and Δy. The offsets Δx and Δy are be determining using equations 6.73 and 6.74.

$$\Delta x = P(t_0).x - \hat{P}(t_0).x \tag{6.73}$$
$$\Delta y = P(t_0).y - \hat{P}(t_0).y \tag{6.74}$$

and the translation of $\hat{P}(t_0 + n\tau)$ to $\hat{P}'(t_0 + n\tau)$ is achieves using equations 6.75 and 6.76

$$\hat{P}'(t_0 + n\tau).x = \hat{P}(t_0 + n\tau).x + \Delta x \tag{6.75}$$
$$\hat{P}'(t_0 + n\tau).y = \hat{P}(t_0 + n\tau).y + \Delta y \tag{6.76}$$

If the coordinate and orientation of $P(t_0)$ and $\hat{P}(t_0)$ are different, equations 6.63 - 6.72 are used, as the first terms of equations 6.70 and 6.71 overcome the translation problem while the second terms of equations 6.70 and 6.71 solved the rotation problem.

6.5.2 Discussion

Solving the problem of delayed measurement by retroactive updating was initially proposed by Kosaka, Meng and Kak (1993) although one could argue, as done later in this section that the concept was already present in the design of the Smith Predictor (Smith, 1959).

Kosaka, Meng and Kak (1993) wanted to solve the stop-and-go motion problem by integrating visual information that was $n\tau$ time steps old into the tracking system. For that purpose, they stored a history of all commands (or shaft encoder readings) from the measured time t_0 to the time $t_0+n\tau$ when the delayed measurement becomes available. They also stored the measured position $\hat{P}(t_0)$ at time t_0. When the delayed measurement $P(t_0)$ becomes available, the new estimation of the current position $\hat{P}'(t_0+n\tau)$ is produced by recalculating the total displacement vector $\hat{d}(t_0,t_0+n\tau)$ from past commands, then rotates the displacement vector by the error $\Delta\theta$ between the measured heading $\hat{P}(t_0)$ and $P(t_0)$, and add it to the new measurement for time $P(t_0)$:

$$\hat{P}'(t+n\tau) = R(\Delta\theta)\hat{d}(t_0,t_0+n\tau) + P(t_0) \tag{6.77}$$

where R is the rotation matrix.

The requirement to store the history of commands in Kosaka, Meng and Kak (1993) was due to the incremental method used to calculate the position uncertainty. As

noted in Maeyama, Ohya and Yuta (1995), for re-estimating the position only, the total displacement is sufficient.

In Maeyama, Ohya and Yuta (1995) a new method is proposed to re-estimate the uncertainty without using the history of commands. This problem is not dealt with in this thesis, as images are acquired at the maximum possible rate, thus there is no advantage in having access to uncertainty information to decide when to recalibrate, as done in Kosaka, Meng and Kak (1993) and Maeyama, Ohya and Yuta (1995).

The method for recalibration of the position used in Maeyama, Ohya and Yuta (1995) differs from that in Kosaka, Meng and Kak (1993) in that in the former a more complex data fusion process is used for generating the new position $\hat{P}'(t_0 + n\tau)$. This consists of a maximum likelihood estimation including all measurement available at time t_0. Otherwise the principle is the same as in Kosaka, Meng and Kak (1993) where the total displacement since t_0 is estimated from odometric measurements.

In a more recent work, Larsen, Andersen and Ravn (1988) are concerned with how to set Kalman Filter parameters given that part of the measurements are delayed. The proposed solution is to extrapolate the delayed measurement $P(t_0)$ to the current time by adding to it the displacement $\Delta\hat{P}(t_0, t_0 + n\tau)$ as determined from all other sensors

$$\hat{P}^{extrapolate}(t_0 + n\tau) = P(t_0) + \Delta\hat{P}(t_0, t_0 + n\tau) \tag{6.78}$$

This extrapolated data is then fused with other measurements available at time $t_0+n\tau$ to produce the best estimation of the position at time $t_0+n\tau$.

The essential difference with the method proposed by Maeyama, Ohya and Yuta (1995) is that data fusion takes place here at time $t_0+n\tau$ rather than at time t_0.

Very similar principles are used in the design of the original Smith Predictor. There, the delayed measurement $P(t_0)$ is available at each time step, hence the recalibration takes place at every time step.

$$\hat{P}'(t_0 + n\tau) = R(\Delta\theta(t_0))\hat{d}(t_0, t_0 + n\tau) + P(t_0) \tag{6.79}$$

where

$$\hat{d}(t_0, t_0 + n\tau) = \hat{P}(t_0 + n\tau) - \hat{P}(t_0) \tag{6.80}$$

Note that this method requires updating at each time step a list of past position vectors $\hat{P}(i) = (\hat{x}(i), \hat{y}(i), \hat{\theta}(i))$, where $i = t_0, \ldots, t_0 + n\tau$. This is required to calculate the orientation error $\Delta\theta$ used to rotate the displacement vector (alternatively, one could keep in memory the list of displacements in every time step for the time span from t_0 to $t_0 + n\tau$).

$$\Delta\theta(t_0) = difference(\hat{\theta}(t_0), \theta(t_0)) \tag{6.81}$$

Note that the need to compare $\hat{P}(t_0)$ and $P(t_0)$ within the fast loop is usually not mentioned in standard descriptions of the Smith Predictor which are not concerned with navigation applications. Therefore the modified diagram of the Smith Predictor (figure 6.26) is proposed that properly shows the pieces of information needed for its operation in the case of a navigation application.

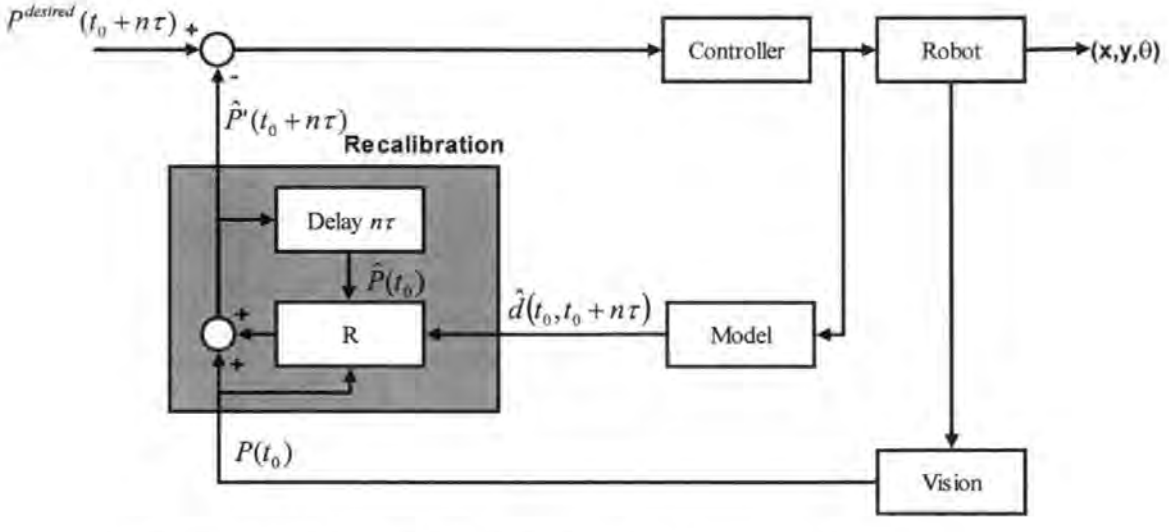


Figure 6.26: The modified Smith Predictor for navigation applications.

When the feedback is intermittent, a further modification is needed. This is because $P(t_0)$ is only available every $n\tau$ steps and consequently also $\Delta\theta$.

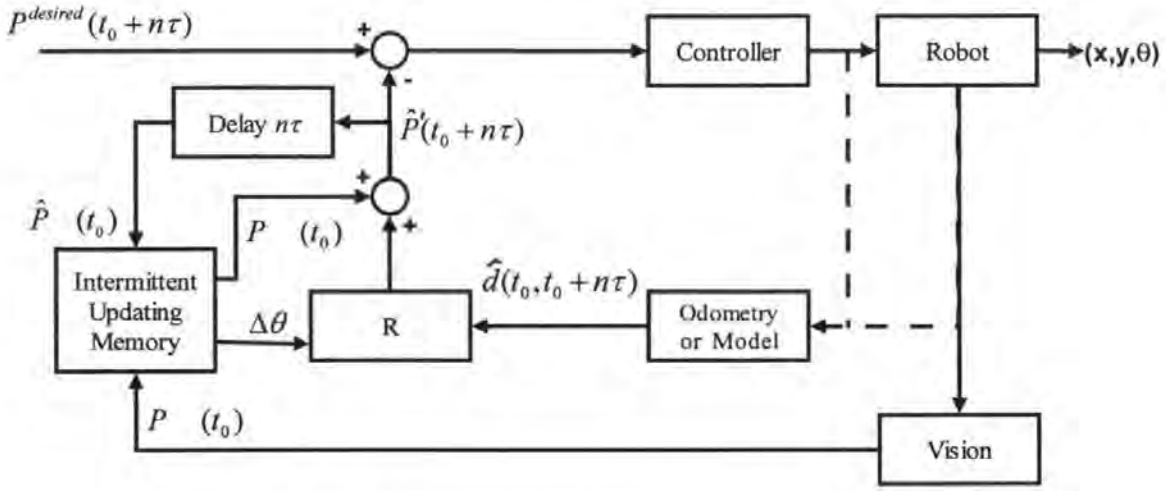


Figure 6.27: The modified Smith Predictor for navigation application with intermittent feedback.

In the proposed additional modification of the Smith Predictor (figure 6.27) only one value of the displacement needs to be updated between t_0 and $t_0+n\tau$ and only the previous estimate $\hat{P}(t_0)$ needs to be stored. These values are now kept in memory for use at every time step, and are changed only when new measurements become available (Figure 6.27).

As a result, intermittent retroactive updating in the Smith Predictor framework turns out to be conceptually equivalent to other methods proposed in different frameworks (Kosaka, Meng and Kak, 1993; Maeyama, Ohya and Yuta, 1995; and Larsen, Andersen and Ravn, 1988). Basically, all methods produce an estimate of the current position by adding the estimated displacement to the delayed measurement.

Chapter 7

Results

7.1 Experiments and Results

7.1.1 Experiments Description

The aim of the experiments described here is to verify if the concurrent control theory does work in practice and to compare the performances of the concurrent and the sequential control systems. The experiments were conducted in the robotic laboratory of the School of Computing, at the University of Plymouth. A robot environment with a size of 125cm by 89cm was built to conduct the experiments. The goal was located at coordinate (77,104), near the top right corner of the robot environment. The starting point of the robot was located at coordinate (45,16). The experiments were divided into two different configurations distinguished by the location of the obstacle. In the first experiment, the obstacle was placed at the centre left of the robot environment, at coordinate (45, 65), while for the seconds experiment the obstacle was place at the centre right of the robot environment at coordinate (75, 65). In experiment one, the expected path for the robot to reach the goal passes round the right of the obstacle while in experiment two, the expected path for the robot passes round the left of the obstacle then heads toward the goal.

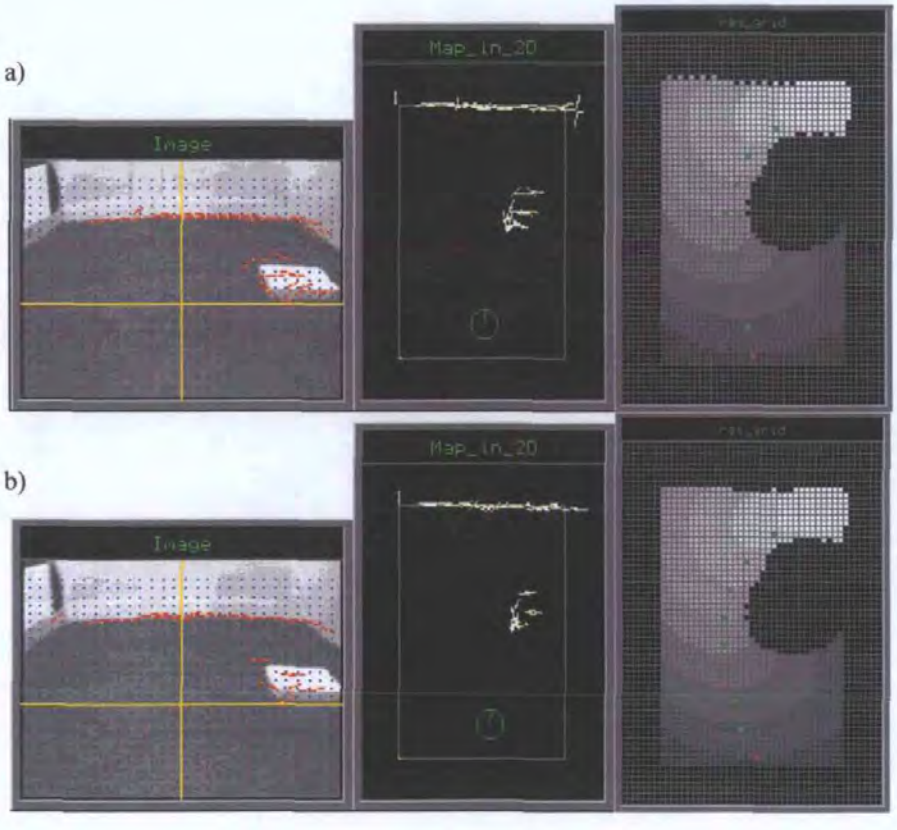
A camera was attached above the robot environment (figure 7.1) to record the path taken by the robot during the experiments.

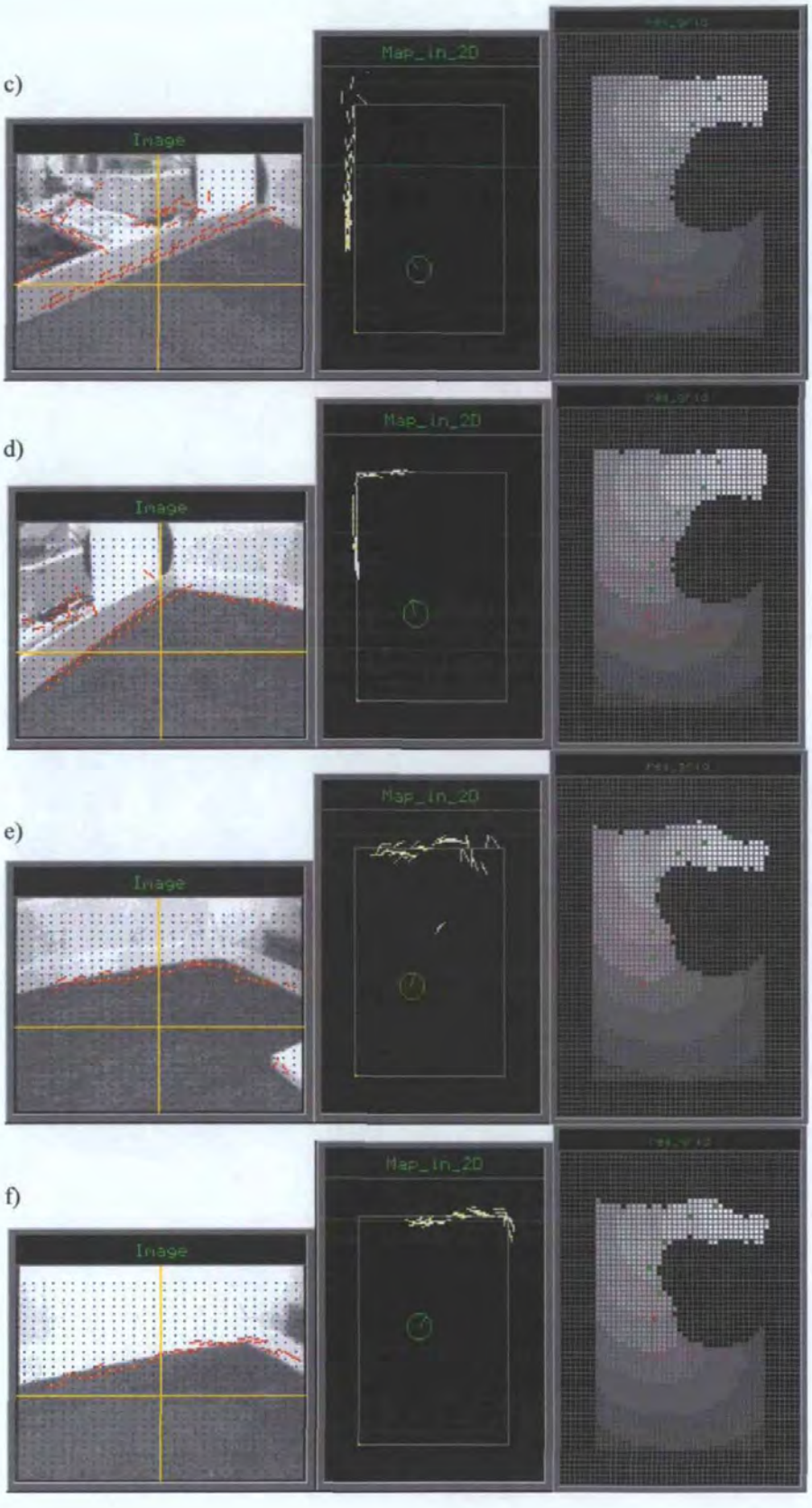
Figure 7.1: The overhead camera setup used to record the robot's motion during the experiments.

The experiments begin with the control system that uses the sequential control method, followed by the system that uses concurrent control method. During an experiment, the robot was asked to navigate towards the goal. The robot had to perform 10 trials for each obstacle configuration and control system. Its paths were recorded and are shown in section 7.12.

A typical sequence of images captured and plans produced during a successful navigation to the goal is illustrated in figure 7.2. In this sequence, a concurrent control method was used. The robot navigated from its initial position (45, 16) toward the goal which is located at position (77, 104) while avoiding the obstacle which was located at

position (75, 65). The 10 sets of figures shown in figure 7.2 illustrate this process. Each set shows, starting from the left, the captured image superimposed with image processing results (edges detection), the 2D map after self-localization (the edges fitting process) and the neuro-resistive grid with the generated waypoints (path). Examples of other robot paths for this configuration are shown in figure 7.4d. The travelled distance for such paths was typically 150 cm for a travel time of approximately 50 seconds (figure 7.3(b)). 10 images were processed during that time (see below). Video recordings showing the robot in motion as well as the computer screen can be found in the CD attached to this thesis.




c)
Image
Map_in_2D
d)
Image
Map_in_2D
e)
Image
Map_in_2D
f)
Image
Map_in_2D

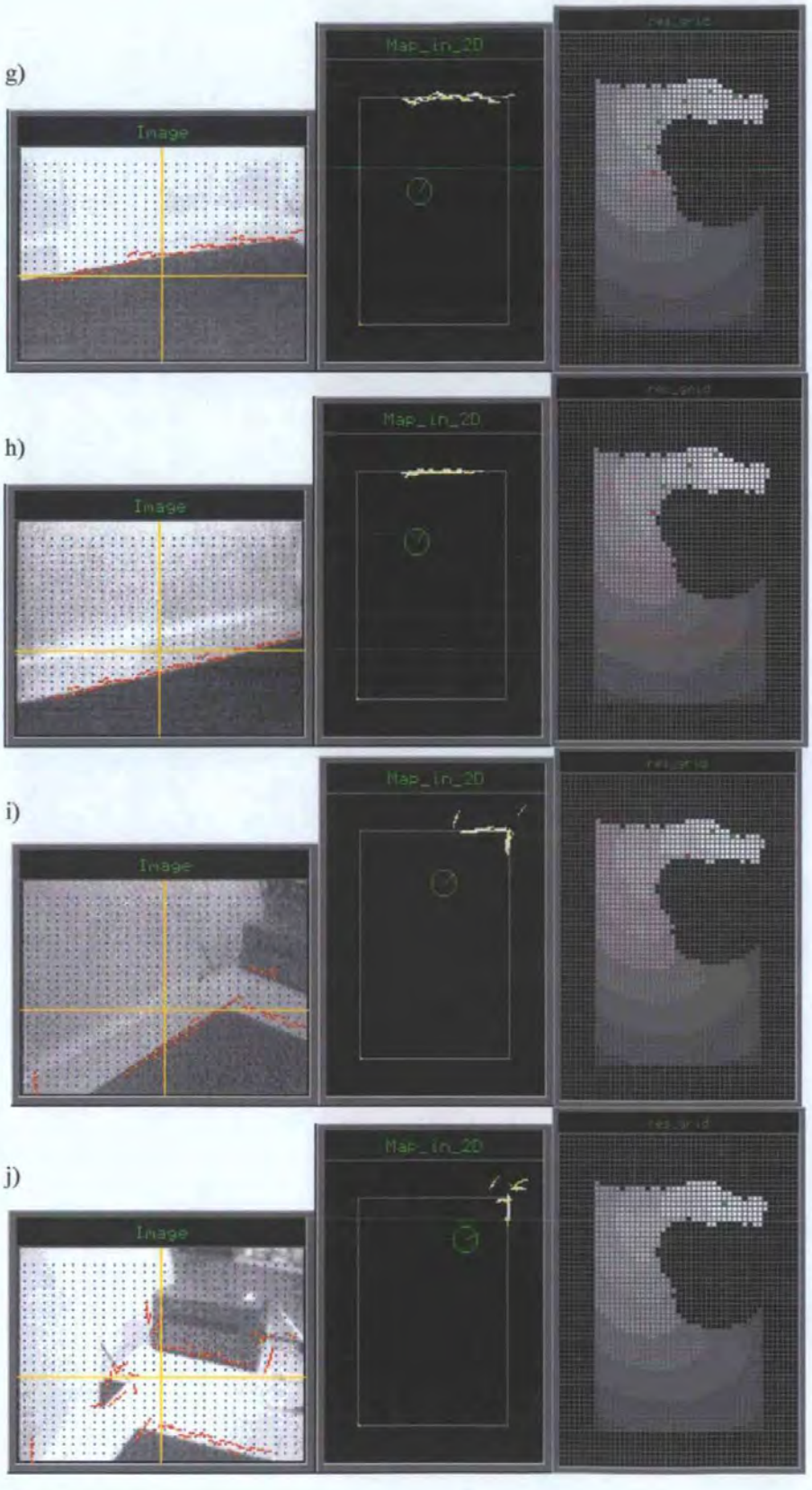


Figure 7.2: Example of the concurrent control system performing the navigation task of experiment 2. The figures (a-j) show the sequence of images captured and plans produced during a successful navigation to the goal. Starting from the left are the captured image, the 2D map illustrating the edges fitting process and the neuro-resistive grid.

7.1.2 Results

The experiments show two important points. Firstly they demonstrate that one of the aims of the thesis has been achieved, namely the design and test of a navigation system that uses vision as main position sensor. Secondly, that the time delays problem exists in the system that uses the sequential control strategy and that it can be overcome with the use of a concurrent control strategy.

Figure 7.3 illustrate an example of the distances-versus-time plot for the system that uses sequential control and the system that uses concurrent control.

Figure 7.3(a) shows the of stop-and-go motion effect in the system that uses the sequential control method. This system takes about 20 seconds longer to complete the navigation task compared to the system that uses concurrent control shown in figure 7.3(b). Figure 7.3(b) shows that the system does not exhibit the stop-and-go motion during navigation.

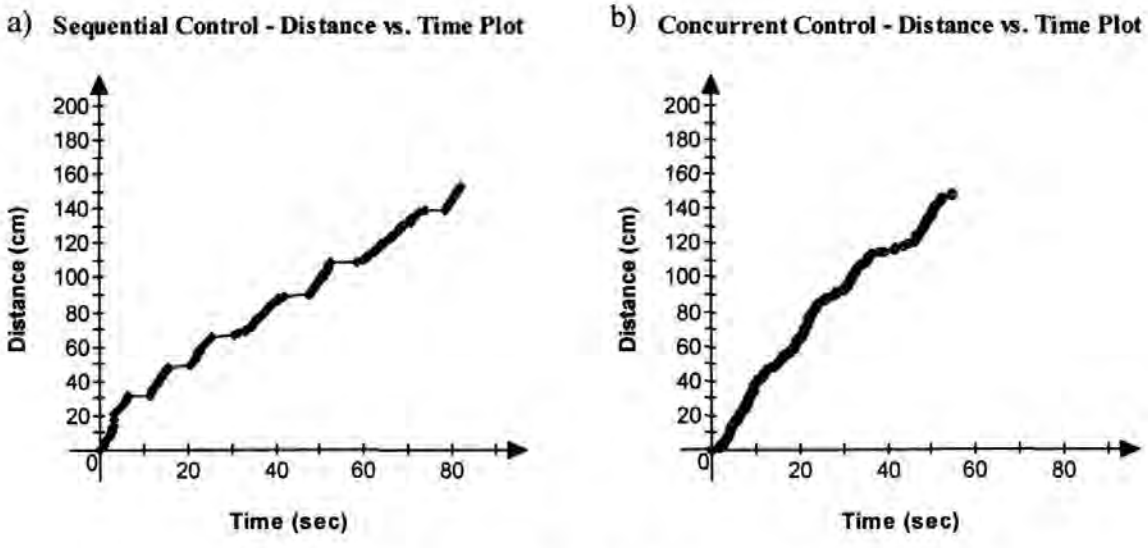


Figure 7.3: The Distance vs. Time plot of two systems that uses different control methods. (a) This figure shows that the system that uses sequential control method exhibits the stop-and-go motion, while (b) shows that the system that uses concurrent control does not exhibit this behaviour (the stop-an-go motion) and completes the task with 20 seconds quicker than (a).

The paths taken by the robot during the navigation experiments are shown in figure 7.4.

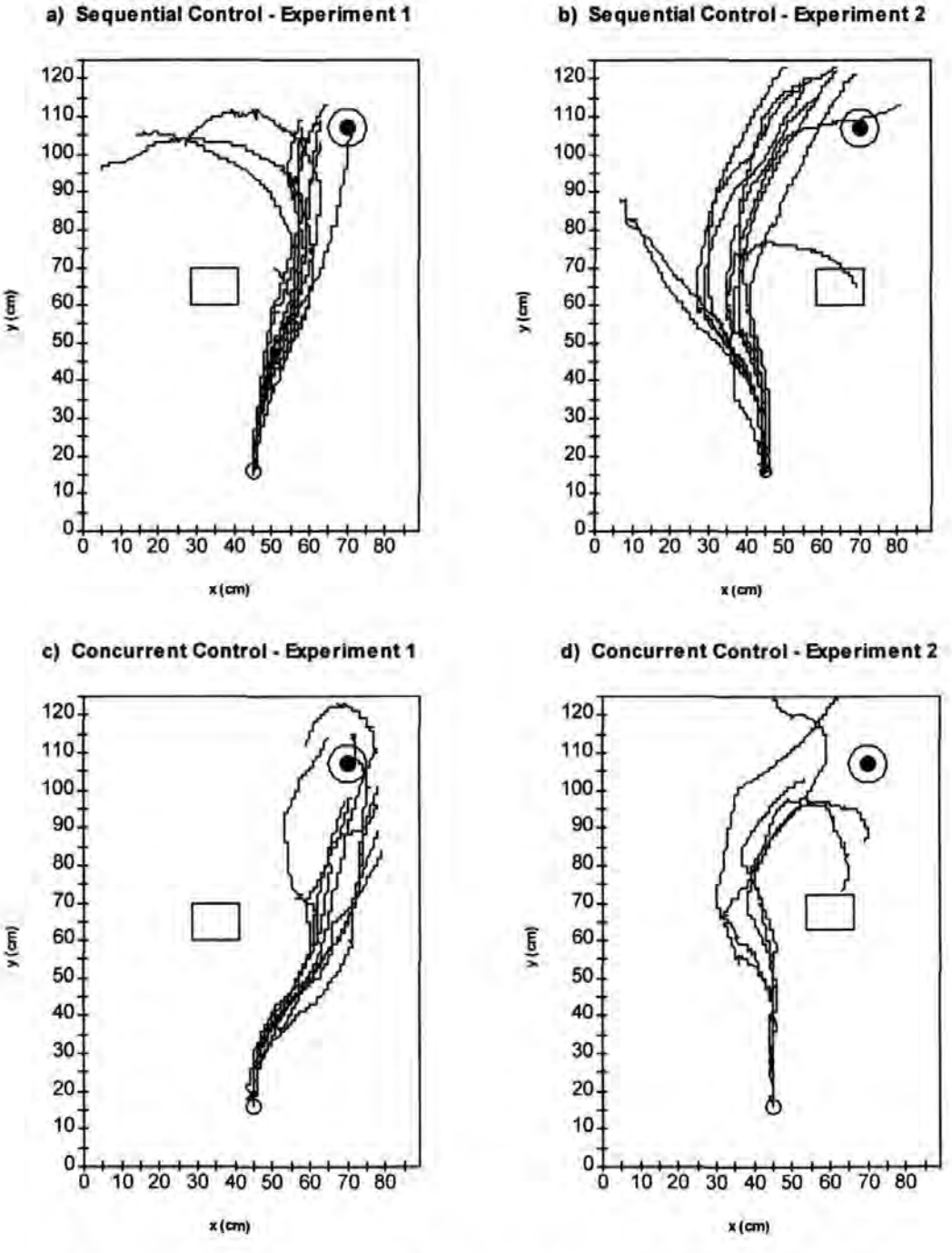


Figure 7.4: The experimental paths produced by the sequential control system (a, b) and the concurrent control system (c, d). Left and right figures are distinguished by the position of the obstacle. Start and goal positions are shown by an empty and a filled circle respectively.

In figure 7.4, one can notice a significant variability in the paths produced. Figure 7.4 (a), (b) and (c) show that 70% of the trials were within 20cm radius of the goal while figure 7.4 (d) shows no trial of such accuracy. This is due to the increased possibility to confuse corners in the self-localization process with paths of figure 7.4 (d). There were some unavoidable problems which were influenced by several factors during the experiment. These include the level of noise in the image, the accuracy of the self-localization process and the accuracy of the shaft encoders reading. These problems are discussed in the next section.

7.1.3 Problems Encountered

During the experiments vision, odometry, and communication problems were encountered.

Vision plays an important role in the navigation process and the performances of the system were often determined by the results of vision processing. Problem such as the environment light intensity changes could sometimes cause the vision system to consider certain parts of the walls as floor (i.e. shadows) or parts of the floor as obstacles (i.e. reflection of light sources). The presence of noise in the image when the robot was in motion (due to an unsteady antenna) often caused the vision system to generate phantom obstacles. These phantom obstacles cause problems for self-localization (i.e. phantom obstacles that are located close to the environment boundaries have the potential to be misinterpreted as detected edges) and path planning (i.e. blockage of valid routes).

Odometry problems were mainly caused by shaft encoders drift. Most severe odometry problems are caused by shaft encoders missing ticks. This occurs when then distance between the IR sensor and the reflector (i.e. encoder striped pattern) falls out of

the sensors operating range. Such problem can reduce the efficiency of the internal loop of the controller and lead to large errors in the displacement vector estimated from odometry and cause the robot to deviate from it's path as shown in figure 7.4 (c) and (d). This in turn affects the visual self-localization process that assumes that the robot is positioned somewhere close to the planned path with a heading roughly parallel to the path. A number of paths in figures 7.4 (a)–(d) where affected by this problem. For instance in figure 7.4 (a) the robot has sometimes mistaken the top left corner for the top right corner and heads towards a goal that now appears to be along the left wall.

Another problem is the lack of sufficient information for self-localization in certain images. It was shown in section 4.5 that the accuracy in the x-direction was reduced due to the limited number of visual clues provided by the side walls. During the navigation of the robot, the situation can become even worse, as only one wall may be visible in the image when the robot is close to it. As a result, vision can only provide position information along one direction (e.g. y-direction). In this case, the best that the remote brain can do is to estimate the position in the other direction by assuming that the robot has followed the desired path. The robot must then use this quite unreliable information for recalibration and is at high risk of getting lost.

The communication process also affects the efficiency of the navigation process. The communication is in principle safe, in that the robot uses security bytes to identify the source of the transmission and check-sums to detect corrupted data. If an error is detected it then requests the remote brain to resend the data. Sometimes the remote brain can miss this request and the robot must ignore recalibration and path data. In that case, the robot pursues its previous path, eventually reaching the last of the sent waypoints.

Another potential communication problem can lead to disorientation problems (i.e. vision-based self-localization establishes erroneous correspondences between detected edges and edges on the prior map). This is due to the fact that, during the communication process, the robot keeps executing the last motor command and may overshoot its target and could be too far from the desired path when the image for vision-based self-localization is captured. This is illustrated in figure 7.4 (d) where the communication process established while the robot was making a turn caused the robot to collide with the obstacle.

In such a case, vision can be the victim of the unreliability of the motion control in that the set of initial positions along the planned path assumed during self-localization does not cover the position that the robot has reached.

Overall vision often failed to perform accurate self-localization, either due to a clue-poor environment, or due to motion errors. Communication problems also contributed, but to a lesser extent.

Due to the large impact of these problems on the system's performance, it would have been of little significance to produce more quantitative evaluations of the performance. The key lessons learnt from these experiments will be discussed in section 8.2.

Chapter 8

Conclusion and Future Work

8.1 Contributions to Knowledge

This thesis has presented a complete vision-based navigation system that can plan an obstacle-avoiding path to a desired destination on the basis of an internal model (map) updated with information gathered from its visual sensors. It has demonstrated a control technique that addresses the stop-and-go motion problem by concurrent image processing and planning while the robot is in motion. Quantitative results of the systems behaviour were shown.

Contributions were made in the areas of vision, planning and control.

During the development of this system, a new floor-edges-specific filter was proposed to detect floor edges and at the same time determine their pose. An algorithm has been proposed to determine precisely the position of the edge in the filter window.

A self-localization algorithm that uses the detected edges and their orientation for estimating the robot's pose was developed. This is done by matching the detected floor edges with the nearest edges in the prior map. In order to limit the potential for aliasing

errors, self-localization is performed by assuming that the robot is located somewhere near to the planned path. The orientation of the robot can then be estimated simply from the average orientation mismatch between edges found in the image and the corresponding edges in the prior map.

The neural-resistive grid which is an ideal data structure for mapping and path-planning was implemented for the first time in a real-world actual application (instead of simulation in Bugmann, Taylor and Denham, (1994); Althöfer and Bugmann. (1995)). A novel scheme was proposed to represent the collision-free space, using divergent connections from the spatial memory to the neuro-resistive grid.

To overcome the stop-and-go motion problem caused by intermittent delayed measurements, a modified Smith Predictor combined with receding horizon control was successfully implemented. Experiments were conducted to demonstrate the system.

A novel implementation of the receding horizon control using NRBF net was proposed. The NRBF path encoder (previously proposed in Koay, Bugmann, Barlow, Philips and Rodney (1998) for an autonomous wheelchair) was implemented on-board the robot to continuously produce a target point to attract the robot toward and along the obstacle free path until the robot reaches the goal. This research demonstrates a system that performs automatic path encoding using the waypoints obtained from the remote brain at every remote brain program cycle, while the previous paper (Koay, Bugmann, Barlow, Philips and Rodney, 1998) demonstrated manual path encoding.

We have proposed two modifications of the Smith Predictor for its use in navigation systems, one with intermittent delayed measurements and the other without. The one for intermittent delayed measurements is used in the demonstrated system to

implement retroactive updating. It has been shown here that other recently proposed methods for handling delayed measurements (Kosaka, Meng and Kak, 1993; Maeyama, Ohya and Yuta, 1995; and Larsen, Andersen and Ravn, 1988) are formally equivalent to the modified Smith Predictor.

8.2 Problems and Difficulties Encountered

In this research, several problems that affect the performances of the system were noted. These essentially contributed to the robot failing to reach the goal in 30% of the trials due to collisions with obstacles or disorientation.

The vision-based self-localization process plays a crucial role in the success of the system in reaching the goal.

During the experiments, failure in vision-based self-localization process often caused the robot to deviate from the given path and collide with obstacles. Failures of the vision-based self-localization process were caused by matching the detected edges with the wrong edges in the prior map.

This can be traced back to several causes. Most of these were related to vision problems such as the presence of noise in the sampled image during navigation. Others were related to the accuracy of the shaft encoders readings and the on-board motion controller (e.g. the robot derived from the designated path due to shaft encoders feedback errors).

The presence of noise in the sampled image during navigation was caused by bad reception due to an unstable antenna on-board the robot. The robot's environment light intensity changes also contributed to the noise level as the vision system could consider

certain parts of the walls as floor (i.e. shadows) or parts of the floor as obstacles (i.e. reflection of light sources).

The noise in the image was often detected by the vision process as detected edges (i.e. floor edges or obstacles), since there is no algorithm for noise detection in the vision process. This confusion lead to the vision-based self-localization matching process to produces unreliable results that could generate disorientation problems.

During vision-based self-localization, those detected edges that did not find a match with edges from the prior map were assumed to be obstacles. These phantom obstacles could lead to the blockage of valid routes.

Apart from vision, the disorientation problem was also caused by the unreliability of the shaft encoders as their feedback could mislead the on-board motion controller to drive the robot away from the designated path. This could cause the vision-based self-localization algorithm to match the detected edges with the wrong edges in the prior map.

The communication process also had the potential of causing disorientation problems. During the communication process, the robot kept executing the last motor command while the robot used its processing power for receiving and handling communication data. This could cause the robot to deviate from the designated path, and cause the vision-based self-localization process to wrongly match detected edges with those in the prior map. Apart from this, the deviation from the designated path could also cause collisions with walls or the obstacle.

Apart from disorientation problems, the vision-based self-localization algorithm also had difficulties in determining the robot's position when the image did not contain enough visual cues, as discussed in section 4.5.2.

Communication problems such as the loss or corruption of data also posed a serious threat to continuous navigation, as this problem could cause the robot to stop at the final waypoint of the current encoded paths while still waiting for the latest path from the remote brain.

The current obstacle detection and registration procedure wasn't able to distinguish between real and phantom obstacles. Therefore all the detected edges were currently being registered into the neuro-resistive grid which is then used for path planning. This phantom obstacles problem should be addressed in future work.

Overall, almost all failures to reach the goal were a fatal combination of vision errors and control errors. The system was designed to allow vision to compensate for control errors, but due to the assumption in the self-localization process that the robot was following the planned path, self-localization was bound to fail when large control errors occurred. However, not making restrictive assumptions about the robot's pose at the time of image capture opens the door to aliasing problems, as many corners and walls look the same.

The design of a future system needs to be reassessed in this light.

8.3 Future work

Future work should aim mainly at producing a more robust vision-based self-localization process.

The disorientation problem could be overcome by decorating the environment with more visual cues. This can be done by first registering the landmarks or visual cues in the map either through sensing or using prior knowledge, then using these landmarks as clues to provides orientation information for matching the detected edges to the appropriate edges in the prior map.

Another method is to search for a landmark within the environment and begin tracking the landmark (Kosaka and Nakazawa, 1995) during the navigation process for deriving the robot's orientation. Note that this method is to track the landmark for relative orientation information; therefore only one landmark is needed at a time, as opposed to other techniques such as triangulation from landmarks that use more than one landmark to derive the robot's position and orientation.

The problem with the lack of visual cues for determining the robot's pose accurately can be overcome by having the camera turning to the sides (i.e. right and left) of the robot to obtain wide-field images. This should be done without the robot going into a static state, but this will require a carefully designed algorithm to enable the combination of the partial position information produced from each view.

As for the communication problem, this can be solved by having the remote brain on-board the robot whereby communication between the remote brain and the robot can be established reliably without lost transmission and corrupted data. Note that, with the

implementation of the remote brain on-board the robot, this would allow the remote brain to gain access into other information which was previously restricted due to the communication bandwidth. These include the robot's pose derived from the shaft encoders. With the remote brain on-board the robot, this opens many other possibilities such as the used of a gyroscope to provide additional orientation information.

Finally the phantom obstacle problem could be overcome by using a verification process in which the detected obstacles have to be confirmed before being placed permanently on the map. This can be done by searching for the same obstacle in two different pictures captured at different times and using occupancy grid techniques.

These are some of the proposed solutions to improve the performance of the system.

Appendix A

Solving the Rug Warrior's Motion Model using Linear Differential Equation of 1st order

General equation:

$$\frac{dy}{dx} + P(x)y = Q(x) \tag{a.1}$$

General solution:

$$y = \gamma(x)u(x) \tag{a.2}$$

where,

$$\gamma(x) = e^{-\int P(x)dx} \tag{a.3}$$

$$u(x) = \int \frac{Q(x)}{\gamma(x)} dx + C \tag{a.4}$$

Using the above method, we will solve for equation (6.26), showing below as equation (*a*.5).

$$\frac{dv}{dt} + Pv = Q \tag{a.5}$$

Firstly, we solve for $\gamma(t)$ where

$$\gamma(t) = e^{-\int_{t_0}^{t} P dt} \tag{a.6}$$

since P is constant therefore

$$\gamma(t) = e^{-P\int_{t_0}^{t} 1 dt} \tag{a.7}$$

$$\gamma(t) = e^{-P[t-t_0]} \tag{a.8}$$

Secondly we solve for $u(t)$,

$$u(t) = \int_0 \frac{Q}{\gamma(t)} dt + C \tag{a.9}$$

$$u(t) = \int_0 \frac{Q}{e^{-P[t-t_0]}} dt + C \tag{a.10}$$

$$u(t) = Q \int_0 e^{P[t-t_0]} dt + C \tag{a.11}$$

$$u(t) = \frac{Q}{P} e^{P[t-t_0]} \Big|_{t_0}^{t} + C \tag{a.12}$$

The solution for the velocity v is then obtain from the product of $\gamma(t)$ and $u(t)$

$$v(t) = \gamma(t) u(t) \tag{a.13}$$

$$v(t) = e^{-P[t-t_0]} \left(\frac{Q}{P} e^{P[t-t_0]} \Big|_{t_0}^{t} + C \right) \tag{a.14}$$

$$v(t) = e^{-P[t-t_0]} \left(\frac{Q}{P} (e^{P[t-t_0]} - e^{P[t_0-t_0]}) + C \right) \tag{a.15}$$

$$v(t) = e^{-P[t-t_0]} \left(\frac{Q}{P} e^{P[t-t_0]} - \frac{Q}{P} + C \right) \tag{a.16}$$

$$v(t) = \frac{Q}{P} (1 - e^{-P[t-t_0]}) + C e^{-P[t-t_0]} \tag{a.17}$$

Appendix B

Publication

1. Koay, K. L., Bugmann, G., Barlow, N. and Philips, M. (1998). Representation of Trajectories for Mobile Robot. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 185–194.

2. Bugmann G., Koay K., Barlow N., Phillips M. and Rodney D. (1998). Stable encoding of robot trajectories using normalised radial basis functions: Application to an autonomous wheelchair. In: D Caldwell, J Gray and P Robinson (eds), *Proceedings of 29th International Symposium Robotics (ISR'98)*, Pages 232–235. DMG Publishers: London.

Representation of Trajectories for Mobile Robots.

Kheng L. Koay*, Dr. Guido Bugmann**,
Dr. Nigel Barlow, Mike Phillips
School of Computing
University of Plymouth
Plymouth PL4 8AA, UK

Donald Rodney
Montpelier Road 4
London SE15 2HF, UK

Abstract. A neural network using Normalized Radial Basis Functions (RBF) is used for encoding the sequence of positions forming the trajectory of an autonomous wheelchair. The network operates by producing the next position for the wheelchair. As the trajectory passes several times over the same point, an additional phase information is added to the position information, which avoids the aliasing problem. The use of normalized RBFs' creates an attraction field over the whole space and enables the wheelchair to recover from any perturbations, for instance due to avoidance of people.

1. Introduction

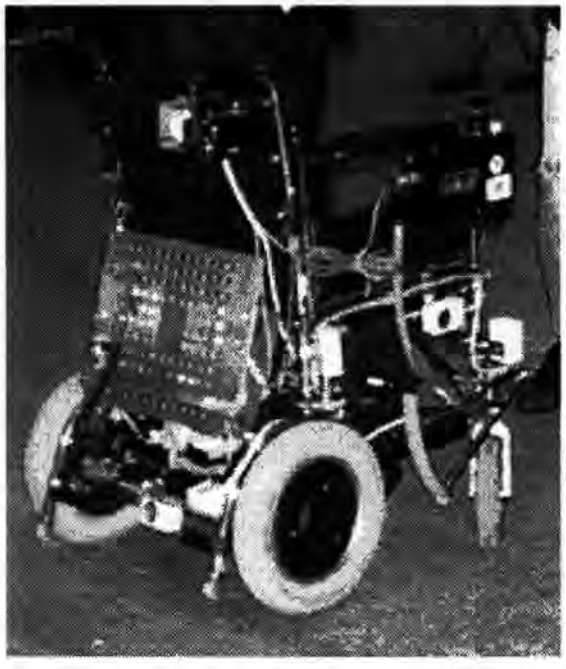


Figure 1. Wheelchair in the South London Gallery during the exhibition.

This paper describes a part of the control system of an autonomous wheelchair that was exhibited in the South London Gallery for a month in 1997. During 7 hours a day, the wheelchair had to perform a repeated sequence of circles, spirals and figures of eight in an unmarked 7m x 7m square area. The public was allowed to enter the area and the wheelchair used sonar for obstacle detection. An obstacle caused the wheelchair to stop. If the "obstacle" did not move after a few seconds, the wheelchair initiated an avoidance maneuver which caused it to leave the desired trajectory. Our problem was to design a control system that i) encodes the complex trajectory and ii) enables the wheelchair to recover from trajectory disturbances, e.g. due to obstacles. This later stability property would also enable the system to be insensitive to the starting point, when restarted in the morning.

In section 2 the use of a control approach based on a map in Cartesian co-ordinates rather than Perception-to-Action principles is justified. In section 3 the basics of the Normalized RBF network are given. In section 4 the encoding of the trajectory is described in details, including the method for encoding phase information. In section 5 properties of the system are discussed, such as the creation of an attraction field and learning capabilities. In section 6 the potential applications in a domestic environment are discussed. The conclusion follows in section 7. The self-localization method is described in the Appendix.

* khenglee@soc.plym.ac.uk
** gbugmann@soc.plym.ac.uk

2. Control Philosophy

The control system was designed as a three stage process. In the first stage, the position of the wheelchair within the gallery was determined. This was done by using a combination of sensors: Sonar, Vision, Shaft encoders and Gyroscope as described in the Appendix. In the second stage, a neural network (NN) used the position information to determine the next position in the trajectory. In the third stage, a standard control procedure (not described in this paper) was used to guide the wheelchair to that position. This task subdivision is similar to the one used in [9]. The main difference with the work in [9] is the use of a NN to encode the trajectory. In contrast to the segment-based representation used in [9], the NN produces a continuous sequence of new targets and "pulls" the wheelchair smoothly along the trajectory. The description of the NN and its properties is the main purpose of this paper.

Another approach, based on encoding Perception-to-Action sequences was considered but not retained due to the characteristics of the problem. In the Perception-to-Action approach [7, 10, 11, 14, 16], visual images from the environment or given sets of sensor readings are associated with given actions, e.g. "when this pattern is seen from this angle, turn left". This could not be used for following reasons: First with people moving around, the gallery could not provide a reproducible sensory signature of a position (problem also noted in [14]). We thought of using a camera directed toward the ceiling but this one did not have sufficiently distinctive patterns. Secondly, the demanded trajectory repeatedly passes in the same point with the wheelchair in the same orientation, this would have caused destructive aliasing (discussed in [2, 14, 16]). Thirdly, Perception-to-Action sequences are not stable against deviations of the trajectory. If the wheelchair find itself in an untrained position off the trajectory, no adequate control action is produced [7].

With the system proposed in this paper, only the desired trajectory needs to be encoded, but adequate control actions are produced over the whole space, and the aliasing problem is avoided.

3. Normalised RBF Nets

Standard Radial Basis Function (RBF) nets comprise a hidden layer of RBF nodes and an output layer with linear nodes [4,5]. The function of these nets is given by:

$$y_i(\mathbf{x}) = \sum_{j=1}^{n} w_{ij}\phi(\mathbf{x} - \mathbf{x}_j) \qquad (1)$$

where y_i is the activity of the output node i, $\phi(\mathbf{x}\text{-}\mathbf{x}_j)$ is the activity of the hidden node j, with a RBF function centred on the vector $\mathbf{x}_j$, $\mathbf{x}$ is the actual input and w_{ij} are the weights from the RBF nodes in the hidden layer to the linear output node (Figure 2). Such a net is a universal function approximator [15].

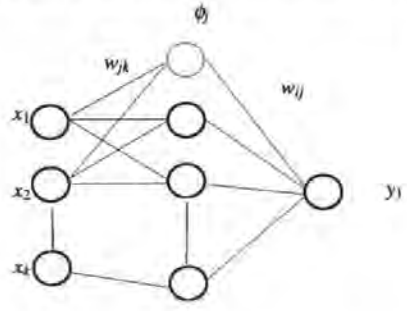


Figure 2. Network architecture for standard RBF nets and Normalized RBF

The function $\phi(\mathbf{x}\text{-}\mathbf{x}_j)$ of a hidden node j is usually the Gaussian Radial Basis Function:

$$\phi(\mathbf{x} - \mathbf{x}_j) = \exp\left(-\frac{\sum_{k=1}^{K}(x_k - w_{jk})^2}{2\sigma^2}\right) \qquad (2)$$

where σ is the width of the Gaussian and K is the dimension of the input space. The "weights" w_{jk} between node k in the input layer and node j in the hidden layer do not act multiplicatively as in other neuron models, but define the input vector $x_j = (w_{j1},...,w_{jK})$ eliciting the maximum response of node j (x_j *is* the "centre of the receptive field").

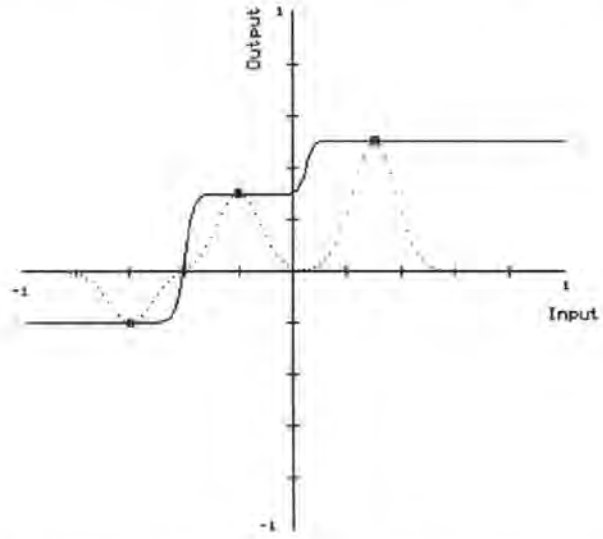


Figure 3. Comparison between standard RBF nets and Normalized RBF nets with three hidden nodes on an example of a 1-Dimensional trajectory. The trajectory has 4 way points: x = -0.6, -0.2, 0.3, 0.5. The trajectory can be represented as a mapping {-0.6 -> -0.2; -0.2 -> 0.3; 0.3 -> 0.5}. *Dotted line:* function of a standard RBF net approximating the mapping. *Full line:* Function of a Normalized RBF net.

Normalised RBF nets have a function very similar to the standard function, with the exception of a normalisation by the total activity in the hidden layer:

$$yi(\mathbf{x}) = \frac{\sum_j w_{ij}\phi(\mathbf{x} - \mathbf{x}_j)}{\sum_j \phi(\mathbf{x} - \mathbf{x}_j)} \qquad (3)$$

As a result, the output activity becomes an activity-weighted average of the input weights in which the weights from the most active inputs contribute most to the value of the output activity. For instance, in the extreme case where only one of the hidden nodes is active, then the output of the net becomes equal to the *weight* corresponding to that hidden node, whatever its actual activity. Thus RBF nodes in the hidden layer are used here as case indicators rather than as basis functions proper.

Figure 3 shows that each hidden node in Normalized RBF nets takes over a portion of the input space over which it determines the output of the net. Due to this property outputs of the normalized RBF net are always a point on the trajectory, even if the current position is not exactly a way point. In contrast, the standard RBF net produces outputs out of the trajectory for input positions that are not exactly on a way point.

A similar normalisation principle is used in the "centre of gravity defuzzification method ([5], pp 388-404). Our approach is a special case of the approach proposed by [17] for selecting linear functions $L_{ij}(\boldsymbol{x})$ (instead of the constant weights w_{ij} used here). In [16] expression (3) was used to compute normalised motor output vectors in robots. Normalised RBF nets show also very good properties in pattern classification applications [8].

A net with the function (3) was originally proposed for sequence encoding in the case of robot arm trajectories [1]. That architecture is extended here with a phase encoding feature that enables encoding of the complex trajectory of the wheelchair which passes repeatedly in the same point in space at different phases of the sequence.

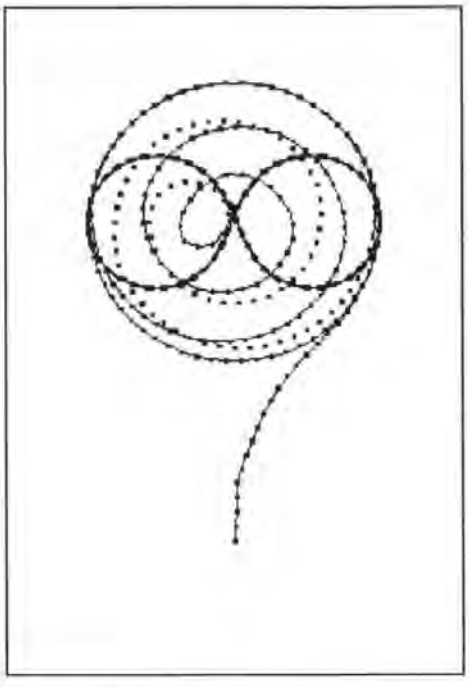


Figure 4. Trajectory encoded by the neural network. The rectangle indicates the walls of the gallery. The figure is produced by simulating the motion of a vehicle starting in the lower half of the image. The outward spiral is indicated by dots only. The motion of the real wheelchair is very similar but we have no recordings of it.

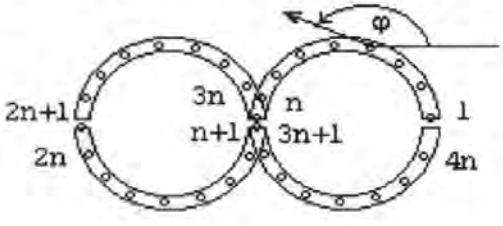


Figure 5. Definition of a figure of eight by four half-

4. Trajectory Encoding

The demanded trajectory for the wheelchair comprises two large circles along the periphery of a 7m x 7m square, then an inward spiral. Once in the center of the square, three successive figures of eight are performed, then an outwards spiral takes place. After that the sequence restarts with two circles (Figure 4).

A) *Decomposition in a sequence of half-circles*

To encode the trajectory with the proposed neural network, the demanded trajectory was divided into 25 half-circles (4 for the large circles, 5 for the inward spiral, 3x4 for the eight's and 4 for the outward spiral). Each half-circle was represented by 5 to 12 equidistant way points. By trial and error it was found that the best distance between way points was approximately 0.9m. The number of way points per half circle was chosen accordingly, depending on its radius.

B) *Neural network*

The NN was designed in such a way that when the wheelchair reached one way point, the output of the network indicated the position of the next way point and the orientation φ of the wheelchair at that position. These values are given as input to a standard control system which issues motor commands. Figure 5 shows an example of 4 half-circles characterizing one figure of eight. Figure 6 shows the part of the neural network encoding the figure.

Normalized RBF nets are well suited for this task because the output activity does not depend critically on the positions (x,y) given at the input. That is because nodes in the hidden layer generate a Voronoi Tesselation [8] of the input space and, for all input values within one of the partition, the output of the net is the same, actually the value of the weight between the active hidden node and the output node (Figure 3).

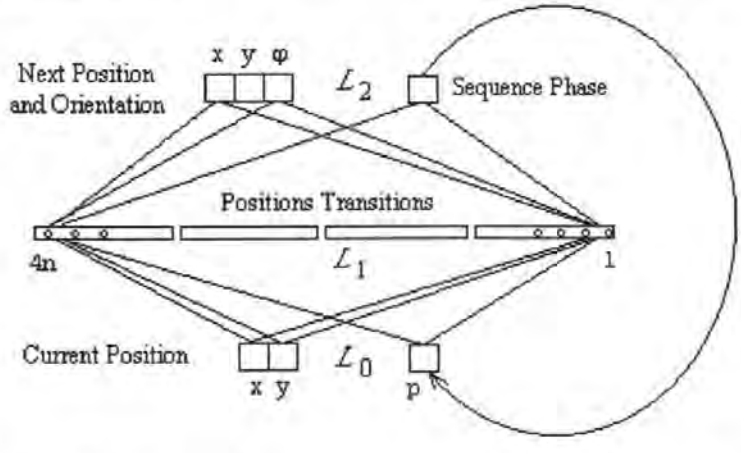


Figure 6. Neural network encoding the demanded trajectory. L_0: Input layer, L_1: Hidden layer, L_2: Output layer.

C) *Off-line Learning*

Learning the desired trajectory are done by a one pass learning procedure, by setting the input weights of each hidden node to the position (x,y) of one way point (equation 4), and its output weights to the position of the

$$w_{j,1} = x_n ; \quad w_{j,2} = y_n ; \quad w_{j,3} = p_n - 0.5 \tag{4}$$

$$w_{1,j} = x_{n+1} ; \quad w_{2,j} = y_{n+1} ; \quad w_{3,j} = \varphi_{n+1} ; \quad w_{4,j} = p_n \tag{5}$$

next way point in the trajectory (equation 5).

where x_n and y_n are the x and y Cartesian co-ordinate of way point n respectively, while p_n is the phase n and x_{n+1} and y_{n+1} is the x and y Cartesian co-ordinate of the next way point $(n+1)$ respectively, while φ_{n+1} is the expected orientation of the wheelchair at way point $n+1$. The use of phase information is explained in the next section.

This is a very fast training procedure. The number of recruited hidden nodes in the network represents the number of way points along the trajectory. An additional output node is used to encode the orientation φ of the wheelchair at the next way point, a parameter used by the low lever control algorithm.

C) *Avoiding aliasing by phase encoding*

It can be seen in Figure 5 that several half-circles have nodes centred on the same position. To make sure that only one of them becomes active at a time, a "Sequence Phase" node was added to the network used in [1] (Figure 6). The weights from each of the nodes in layer L_1 to the phase node are equal to their position in the sequence or "phase". For instance, if the first node in the sequence is active, the Sequence Phase node will have an output 1. If the 10^{th} is active, the output will be 10, etc. The output of that node is used as input by the "Position Transition" nodes in layer L_1. Their input weights for the phase are set to their phase - 0.5. For example, the 10^{th} node has a receptive field (for phases) centred on 9.5. In that way, nodes start to become activated when the system is in the phase prior to their own (or in their own) and when the wheelchair is in the position defined by the two weights from the "Current Position" in layer L_0. Therefore, when a position corresponds to many nodes, only the one receptive to the current phase becomes activated and can indicate the next position in the trajectory. A special routine was written to reset the phase at the end of the sequence, to enable a repeat of the trajectory.

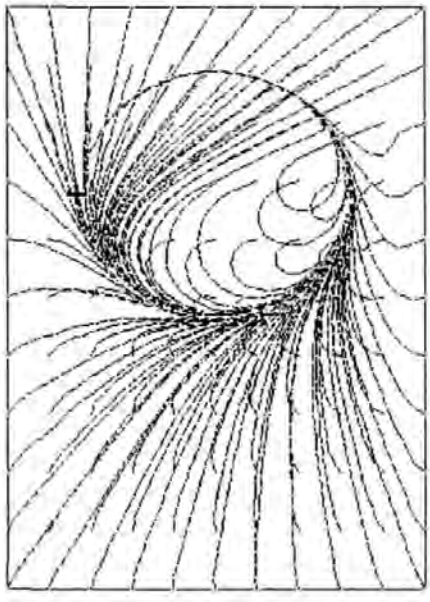


Figure 7. Illustration of the attraction field generated by the neural network for the large circle.

5. Properties of the trajectory generator

A) *Attraction field*

RBF nodes with a Gaussian function produce a response over the whole input space (x,y,p). The response is very weak for most combinations of position (x,y) and phase p. For instance, when the wheelchair is far from the trajectory, only a very weak response is elicited in any of the nodes in layer L_1 of the net. However, due to the normalization in (3), the network can output a value for the next position, as encoded in the weights to the layer "Next Position". Thus normalization results in an attraction field that leads the wheelchair towards the demanded trajectory from whatever starting point (Figure 7).

The smooth approach-curves in Figure 7 are due to the internal dynamics of the network. Let us assume a starting point as in Figure 4. Initially the phase p is set to 0.5, so that mainly the first node is activated (this node is centred on the position indicated by a cross in Figure 7, and is part of a descending half-circle). Thus the first goal position indicated by the network is one node ahead of the first node. However, being active, the first node causes the phase to become $p = 1$. This in turn enables the second node to become active, which gives now a goal

position one node ahead of the second node. Thus the wheelchair is given a changing goal as it approaches the trajectory. Interestingly, this movement of the goal also occurs when the wheelchair is on the trajectory, and it needs to be controlled to avoid goals running too far ahead of the actual position. This control involves either a lower frequency of updating of the Sequence Phase node, or a balancing of the role of position and phase in the activation of nodes in layer L_1, as explained below.

It can be seen from (2) that the activity of any node in layer L_1 of the net is the product of three one-dimensional Gaussian functions centred on their preferred x, y and p respectively. Let us assume that these Gaussians' have different widths. If the width for p is large (low selectivity), the winning (most active) node is determined by the position of the wheelchair. However, if the width for p is small (high selectivity), the value of p becomes most important in determining the activity of the node. In this case, the net can run through the sequence irrespective of the position of the wheelchair. We have found that a good balance between the role of position and phase is obtained when the width $\sigma = 1$ for the phase and the position (in meter).

B) *Aliasing*

In this work the position (x,y) of the vehicle was used as input to the sequence encoding network. Aliasing occurs when the same (or similar) position reoccurs at different times in the trajectory. By adding a phase node we have avoided that the vehicle jumps from one phase of the trajectory to another, hence solving the aliasing problem.

In Perception-to-Action systems, aliasing is also a problem [7, 14, 16]. The difference is that some (position specific) complex sensory picture is used instead of the position (x,y). It should also be possible to avoid aliasing in these systems by adding phase information to the picture.

C) *Performance*

In average, the wheelchair worked independently for 45min. At that time it was usually lost in some corner of the gallery and an operator had to replace it at the starting position and reset the program. The batteries however needed only one charge per day. For the purpose of providing a show, these performances were acceptable. The duration of autonomous operation was limited solely by the problems of self-localization. As mentioned in the appendix, the lighting conditions in the gallery did not allow a dynamic recalibration of the orientation using a CCD camera. This in turn prevented to use sonar reliably to measure the distances to the walls, which requires the orientation to be known (see appendix). Hence self-localization relied solely on shaft encoders.

Some difficulties in precisely following the desired trajectory were due to dynamic limitations of the low level control algorithm. A compensating measure was to define the motion speed separately for each semi-circle, with a slower speed for the smaller half-circles. Further work is needed at that level.

However, the trajectory encoding system described here showed no problems.

6. Potential applications in a domestic environment

Theoretically, the wheelchair can be programmed on-line, with a new hidden node (way-point) added to the network at fixed distance intervals while the wheelchair is being pushed through a desired trajectory. Different trajectories can be encoded by using extra output nodes broadcasting the identity of the trajectory, e.g. some code for the goal and the starting point. Therefore, the proposed trajectory encoding system has the potential for use in domestic environments.

One point that may need some thoughts is the fact that the density of way points needs to be larger in segments with high curvature, requiring really a variable interval between way points. Another point to consider is the fact that the attractive field does cross walls (unlike fields in Laplacian planning methods [6]), hence it is preferable to initiate the path-following procedure when close to a way point.

The biggest limitation currently is the self-localization procedure which needs to be much more robust. For that purpose, we are now developing vision based techniques for layout recognition and analysis.

7. Conclusion

A simple neural network has been described that encodes trajectories in a stable way, allowing recovery from disturbances and implementing a new phase encoding principle that solve the aliasing problem. The wheelchair produced a satisfactory show for a whole month in an art gallery. For domestics applications, improvements in self-localization and low-level control are needed.

8. Acknowledgment

The authors gratefully acknowledge support in many forms by British Aerospace Systems & Equipment (Plymouth), Mike Denham, Peter Frere (Lucas Advanced Engineering Centre, Birmingham), Steve Hill, the Henry Moore Foundation, David Keating, Peter Nurse, Penny and Giles Drives Technology Ltd, Plymouth Disability Equipment Centre, Paul Robinson, Alan Simpson, the South London Gallery, and others mentioned in the web page: http://www.tech.plym.ac.uk/soc/research/neural/research/wheelc.htm

9. References

[1] K. Althoefer and G. Bugmann (1995) "Planning and Learning Goal-Directed Sequences of Robot-Arm movements", in Fogelman-Soulié F. and Gallinari P. (eds), *Proc. of the International Conference on Artificial Neural Networks (ICANN'95)*, Paris, Vol. 1, 449-454.

[2] D. H. Ballard and S. D. Whitehead (1992) "Learning visual behaviours" In H. Wechsler, editor, *Neural Networks for Perception*, volume 2, pages 8-39. Academic Press.

[3] V. Braitenberg (1984) "Vehicles, Experiments in Synthetic Psychology", MIT Press, Cambridge, Massachussets.

[4] D. S. Broomhead and D. Lowe (1988) "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems*, 2, pp. 321-355.

[5] M. Brown and C. Harris (1994) "Neurofuzzy Adaptive Modelling and Control", Prentice Hall, Hemel , UK.

[6] G. Bugmann, J. G. Taylor, and M. Denham (1994) "Route finding by neural nets" in J. G. Taylor (ed) "Neural Networks", Alfred Waller Ltd, Henley-on-Thames, p. 217-230.

[7] G. Bugmann, G. (1997) "A Connectionist Approach to Spatial Memory and Planning" as Chap. 5 in L.J. Landau and J.G. Taylor (eds) "Basic Concepts in Neural Networks: A survey", In the Series: Perspectives in Neural Networks, Springer, London, pp. 109-146.

[8] G. Bugmann (1998) "Normalized Radial Basis Function Networks", To appear in *Neurocomputing:* Special Issue on Radial Basis Function Networks.

[9] I. J. Cox (1991) "Blanche - An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle", *IEEE Transactions on Robotics and Automation*, Vol.7, No. 2, April 1991.

[10] P. Gaussier and S. Zrehen (1995) "PerAc: A neural architecture to control artificial animals", *Robotics and Autonomous Systems*, 16, pp. 291-320.

[11] Maja J. Mataric (1991) "Navigating With a Rat Brain: A Neurobiologically-Inspired Model for Robot Spatial Representation", *Proc. SAB'91*, Paris, pp.169-175.

[12] J. Moody and Ch. Darken (1989) "Fast learning in networks of locally-tuned processing units.", *Neural Computation*, 1, 281-294.
[13] W. L. Nelson and I. J. Cox, "Local path control of an autonomous vehicle", in *Proc. IEEE Int. Conf. Robotics Automat.*, 1988, pp. 1504-1510.

[14] C. Owen and U. Nehmzow, "Middle scale navigation - a case study", Proc. AISB 97 workshop on "Spatial Reasoning in Animals and Robots", *Tech report series, Department of Computer Science, Manchester University*, ISSN 1361-6161. Report number UMCS-97-4-1.

[15] M. J. D. Powell (1987) "Radial Basis Functions for Multivariate Interpolation: A Review", in Mason J.C. and Cox M.G. (eds) *Algorithms for Approximation*, Clarendon Press, Oxford, pp. 143-167.

[16] R. P. N. Rao and O. Fuentes (1996) "Learning Navigational Behaviour using a Predictive Sparse Distributed Memory", *Proc. of From Animals to Animats: The Fourth International Conference on Simulation of Adaptive Behaviour*, MIT Press, pp. 382-390.

[17] J. Shao, Y. V. Kee and R. Jones (1993) "Orthogonal Projection Method for Fast On-Line Learning Algorithm of Radial Basis Function Neural Networks", *Proc. INNS World Congres on Neural Networks*, Portland Oregon, USA, Vol.3, pp. 520-535.

Appendix: Self Localization

Robot self-localization is important for keeping the wheelchair on its trajectory (Figure 4) over extended periods of time. Two localization methods were used in this wheelchair project. One method was static localization, which is used to confirm the initial position of the wheelchair in the morning, or after a reset. The other method was dynamic localization, which involved correcting the wheelchair position and orientation during task performance.

The wheelchair was controlled by a laptop Pentium PC attached at the back (Figure 1) running the neural network simulation software CORTEX-PRO which also handled sensor integration. The sensors used in these operations are described below. All the inputs from these sensors were given a weighting, based on how much these inputs were entrusted. The weighted average (i.e. equation A.1) was then used to reinitialize the wheelchair position and orientation. The concept of multi-sensor fusion was used here to produce a more robust self-localization. Equation A.1 illustrates the calculation of the orientation φ which is based on up to three sensors.

$$\varphi = \frac{\varphi\_shaft \times w\_shaft + \varphi\_camera \times w\_camera + \varphi\_gyro \times w\_gyro}{\varphi\_shaft + \varphi\_camera + \varphi\_gyro} \qquad (A.1)$$

where φ_shaft is the input given by shaft encoders integration, w_shaft is the weight given to the shaft encoder, φ_camera is the input given by camera integration, w_camera is the weight given to the camera, φ_gyro is the input given by the gyroscope, w_gyro is the weight given to the gyroscope. The weights can be set according to how much drift each sensor has, and other factors. For initial tests, these were all set to 1 during updating cycles when the sensor were able to provide data, and to zero at other times.

A) *Vision: Robot orientation tracking*

A QuickCam camera was mounted at the upper right back of the wheelchair with its lens pointed towards the ceiling for horizontal beam searching using a "Hough Transform". These horizontal beams were to be used to calculate the wheelchair-heading vector. However, during test runs at the South London Art Gallery, it turned out that the camera was blinded by the spot lights which shone down from the ceiling. This prevented the use of the camera, hence *w_camera* was set to zero.

B) *Gyroscope: Robot orientation tracking*

A single axis Rate Gyroscope was mounted on the wheelchair to helped the wheelchair track its orientation (i.e. wheelchair heading). During test runs, the rate gyroscope was found to drift more than the orientation calculated from shaft encoding, so it was not suitable for re-calibration, hence *w_gyro* was also set to zero.

C) *Shaft Encoder: Robot orientation and position tracking*

Two incremental shaft encoders were used with the wheelchair to help keep track of its own location within its internal map. These shaft encoders consist of two striped pattern (200 stripes per rotation for a diameter of 31.5cm), glued to the wheels and photo-reflectors. These detect the reflected light from the striped pattern and produce a series of pulse-trains during the wheels' rotation. These pulse-train outputs were stored in an incremental counter. The counter was then used to calculate the distance traveled by the wheels. Distances traveled by each wheel (i.e. dL for left wheel and dR for right wheel) were integrated to calculate the wheelchair's new position (in Cartesian co-ordinate x and y) and orientation φ.

D) *Sonar: Position tracking and obstacle detection*

Eight Polaroid sonar range-finding systems (Polaroid 6500 Sonar Kits) with operational range from 0.30m to 10m were used for distance measurements and obstacle detection. Most of the sensors were looking ahead, to avoid collisions with spectators (Figure. A.1). If objects were detected nearer than 1.7m to the wheelchair, the wheelchair stopped and executed an obstacle avoidance routine.

Obstacles that did not move after a few seconds were consider to be static. In this case, the wheelchair turned away from the obstacle, rotating by a fixed angle in the direction opposite from where the obstacle was detected. If no more obstacle was then detected, the wheelchair followed the direction given by the NN, and reentered the trajectory. If no free path was detected the wheelchair stopped and continually beeped.

In the case where the obstacle moved within a few seconds, the wheelchair resumed its trajectory.

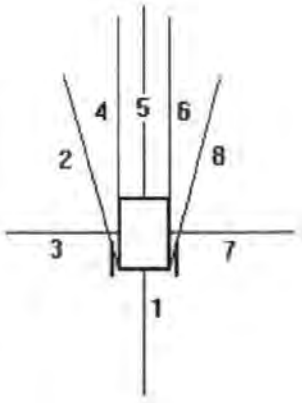


Figure A.1. Configuration of the sonar sensors attached on the wheelchair.

During static localization, sonar numbers 1, 3, 5 and 7 were used to determine the stationary wheelchair position, relative to the wall in the room. As the wheelchair moved, measurements were dynamically taken with the sonar sensors (1, 3, 5 and 7) when perpendicular to the walls in the room. Measurements were hence taken when the orientations of the wheelchair were 0°, 90°, 180°, and 270°. These measurements were used to calculate weighed average (equation A.1). The weighted average was then used as the wheelchair's current position and orientation.

Proceedings of the 29th Intl. Symp. on Robotics, Advanced Robotics : Beyond 2000, 27 - 30 April 1998, Birmingham, UK

Stable Encoding of Robot Trajectories using Normalised Radial Basis Functions: Application to an Autonomous Wheelchair.

Dr. Guido Bugmann, Kheng L. Koay,
Dr. Nigel Barlow, Mike Phillips
School of Computing
University of Plymouth
Plymouth PL4 8AA, UK

Donald Rodney
Montpelier Road 4
London SE15 2HF, UK

Abstract **- A neural network using Normalised Radial Basis Functions (RBF) is used for encoding the sequence of positions forming the trajectory of an autonomous wheelchair. The network operates by producing the next position for the wheelchair. As the trajectory passes several times over the same point, an additional phase information is added to the position information, which avoids the aliasing problem. The use of normalised RBFs' creates an attraction field over the whole space and enables the wheelchair to recover from any perturbations, for instance due to avoidance of people.**

I INTRODUCTION

This paper describes a part of the control system of an autonomous wheelchair that was exhibited in the South London Gallery for a month in 1997. During 7 hours a day, the wheelchair had to perform a repeated sequence of circles, spirals and figures of eight in an unmarked 7 m x 7 m square area. The public was allowed to enter the area and the wheelchair used sonar for obstacle detection. An obstacle caused the wheelchair to stop. If the "obstacle" did not move after a few seconds, the wheelchair initiated an avoidance manoeuvre which caused it to leave the desired trajectory. Our problem was to design a control system that i) encodes the complex trajectory and ii) enables the wheelchair to recover from trajectory disturbances, e.g. due to obstacles. This later stability property would also enable the system to be insensitive to the starting point, when restarted in the morning.

In section II the use of a control approach based on a map in Cartesian co-ordinates rather than Perception-to-Action principles is justified. In section III, the basics of the Normalised RBF network are given. In section IV, the encoding of the trajectory is described in details, including the method for encoding phase information. In section V general properties of the system are discussed, such as the creation of an attraction field and learning capabilities. The conclusion follows in section VI.

II CONTROL PHILOSOPHY

We designed the control system as a three stage process. In the first stage, the position of the wheelchair within the gallery was determined. This was done by using a combination of sensors: Sonar, Vision, Shaft encoders and Gyroscope. In the second stage, a neural network (NN) used the position information to determine the next position in the trajectory. In the third stage, a standard control procedure was used to guide the wheelchair to that position. The NN continuously gave a new target before the old one was reached and "pulled" the wheelchair along the trajectory. Only the second stage is described in this paper.

Another approach, based on encoding Perception-to-Action sequences was considered but not retained due to the characteristics of the problem. In the Perception-to-Action approach, visual images from the environment or given sets of sensor readings are associated with given actions, e.g. "when this pattern is seen from this angle, turn left". This could not be used for following reasons: First with people moving around, the gallery could not provide a reproducible sensory signature of a position. We thought of using a camera directed toward the ceiling but this one did not have sufficiently distinctive patterns. Secondly, Perception-to-Action sequences are not stable against deviations of the trajectory. If the wheelchair find itself in an untrained position off the trajectory, no adequate control action is produced.

With the system proposed in this paper, only the desired trajectory needs to be encoded, but adequate control actions are produced over the whole space.

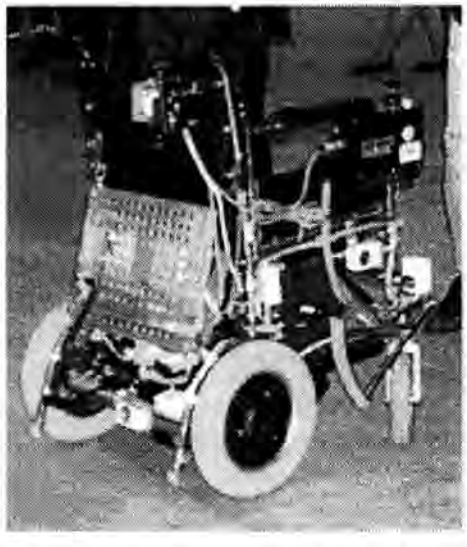


Fig. 1. Wheelchair controlled by a laptop Pentium PC attached at the back running the neural network simulation software CORTEX-PRO. Sonar sensors in small white boxes are used to avoid collisions and for self-localisation.

III NORMALISED RBF NETS

Standard Radial Basis Function (RBF) nets comprise a hidden layer of RBF nodes and an output layer with linear nodes [3,4]. The function of these nets is given by:

$$y_i(x) = \sum_{j=1}^{n} w_{ij}\,\phi(x - x_j) \qquad (1)$$

where y_i is the activity of the output node i, $\phi(x\text{-}x_j)$is the activity of the hidden node j, with a RBF function centred on the vector x_j, x is the actual input and w_{ij} are the weights from the RBF nodes in the hidden layer to the linear output node. Such a net is a universal function approximator [6].

The function $\phi(x\text{-}x_j)$ of a hidden node j is usually the Gaussian Radial Basis Function:

$$\phi(x - x_j) = \exp\left(-\frac{\sqrt{\sum_{k=1}^{K}(x_k - w_{jk})^2}}{2\sigma^2}\right) \qquad (2)$$

where σ is the width of the Gaussian and K is the dimension of the input space. The "weights" w_{jk} between node k in the input layer and node j in the hidden layer do not act multiplicatively as in other neuron models, but define the input vector $x_j = (w_{j1},...,w_{jK})$ eliciting the maximum response of node j (x_j *is* the "centre of the receptive field").

Normalised RBF nets have a function very similar to the standard function, with the exception of a normalisation by the total activity in the hidden layer:

$$y_i(x) = \frac{\sum_j W_{ij}\,\phi(x - x_j)}{\sum_j \phi(x - x_j)} \qquad (3)$$

As a result, the output activity becomes an activity-weighted average of the input weights in which the weights from the most active inputs contribute most to the value of the output activity. For instance, in the extreme case where only one of the hidden nodes is active, then the output of the net becomes equal to the *weight* corresponding to that hidden node, whatever its actual activity. Thus RBF nodes in the hidden layer are used here as case indicators rather than as basis functions proper.

A similar normalisation principle is used in the "centre of gravity defuzzification method ([4], pp 388-404). Our approach is a special case of the approach proposed by [8] for selecting linear functions $L_{ij}(x)$ (instead of the constant weights W_{ij} used here). In [7] expression (3) was used to compute normalised motor output vectors in robots. Normalised RBF nets show also very good properties in pattern classification applications [2].

A net with the function (3) was originally proposed for sequence encoding in the case of robot arm trajectories [1]. That architecture is extended here with a phase encoding feature that enables encoding of the complex trajectory of the wheelchair which passes repeatedly in the same point in space at different phases of the sequence.

IV TRAJECTORY ENCODING

The demanded trajectory for the wheelchair comprises two large circles along the periphery of a 7m x 7m square, then an inward spiral. Once in the centre of the square, three successive figures of eight are performed, then an outwards spiral takes place. After that the sequence restarts with two circles (fig. 2).

A) *A sequence of half-circles*

The demanded trajectory was divided into 25 half-circles (4 for the large circles, 5 for the inward spiral, 3x4 for the eight's and 4 for the outward spiral). Each half-circle was represented by 4 to 12 RBF nodes, depending on the its diameter. The receptive field centres of the nodes were equidistantly distributed along the half-circle. Their three output weights represented the position (x,y) and orientation φ of the wheelchair at the next position (next node) in the half-circle. These values are given as input to a standard control system which issues motor commands. Fig. 3. shows an example of 4 half-circles characterising one figure of eight. Figure 4 shows the part of neural network encoding the figure.

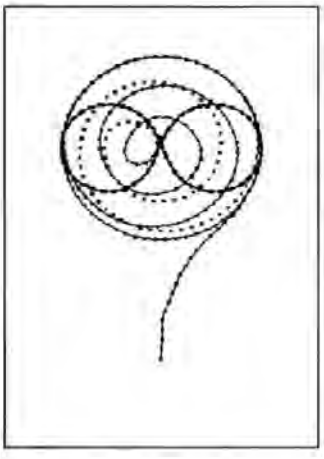


Fig. 2. Trajectory encoded by the neural network. The rectangle indicates the walls of the gallery. The figure is produced by simulating the motion of a vehicle starting in the lower half of the image. The outward spiral is indicated by dots only. The motion of the real wheelchair is very similar but we have no recordings of it.

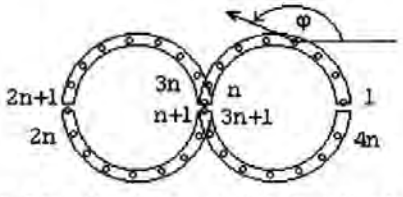


Fig. 3. Definition of a figure of eight by four half-circles.

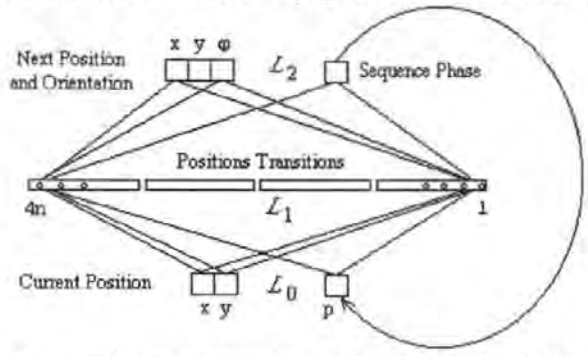


Fig. 4. Neural network encoding the demanded trajectory. The width of the receptive fields for the positions was set to a fifth of the radius of the half-circle. For the phases, the width receptive field was set to 1. L_0: Input layer, L_1: Hidden layer, L_2: Output layer.

B) *Avoiding aliasing by phase encoding.*

It can be seen in Fig. 3 that several half-circles have nodes centred on the same position. To make sure that only one of them becomes active at a time, a "Sequence Phase" node was added to the network used in [1] (Fig. 4). The weights from each of the nodes in layer L_1 to the phase node are equal to their position in the sequence (or "phase"). For instance, if the first node in the sequence is active, the Sequence Phase node will have an output 1. If the 10th is active, the output will be 10, etc. The output of that node is used as input by the "Position Transition" nodes in layer L_1. Their input weights for the phase are set to their phase - 0.5. E.g. the 10^{th} node has a receptive field (for phases) centred on 9.5. In that way, nodes start to become activated when the system is in the phase prior to their own (or in their own) *and* when the wheelchair is in the position defined by the two weights from the "Current Position" in layer L_0. Therefore, when a position corresponds to many nodes, only the one receptive to the current phase becomes activated and can indicate the next position in the trajectory. A special routine was written to reset the phase at the end of the sequence, to enable a repeat of the trajectory.

V PROPERTIES

A) *Attraction field*

RBF nodes with a Gaussian function produce a response over the whole input space (x,y,p). The response is very weak for most combinations of position (x,y) and phase p. For instance, when the wheelchair is far from the trajectory, only a very weak response is elicited in any of the nodes in layer L_1 of the net. However, due to the normalisation in (3), the network can output a value for the next position, as encoded in the weights to the layer "Next Position". Thus normalisation results in an attraction field that leads the wheelchair towards the demanded trajectory from whatever starting point (Fig. 5).

The smooth approach-curves in Fig. 5 are due to the internal dynamics of the network. Let us assume a starting point as in fig. 2. Initially the phase p is set to 0.5, so that mainly the first node is activated (this node is centred on the position indicated by a cross in Fig. 5, and is part of a descending half-circle). Thus the first goal position indicated by the network is one node ahead of the first node. However, being active, the first node causes the phase to become $p = 1$. This in turn enables the second node to become active, which gives now as goal position one node ahead of the second node. Thus the wheelchair is given a changing goal as it approaches the trajectory. Interestingly, this movement of the goal also occurs when the wheelchair is on the trajectory, and it needs to be controlled to avoid goals too far ahead of the actual position. This control involves either a lower frequency of updating of the Sequence Phase node, or a balancing of the role of position and phase in the activation of nodes in layer L_1, as explained below.

It can be seen from (2) that the activity of any node in layer L_1 of the net is the product of three one-dimensional Gaussian functions centred on their preferred x, y and p respectively. Let us assume that these Gaussians' have different widths. If the width for p is large (low selectivity), the winning (most active) node is determined by the position of the wheelchair. However, if the width for p is small (high selectivity), the value of p becomes most important in determining the activity of the node. In this case, the net can run through the sequence irrespective of the position of the wheelchair. We have found that a good balance between the role of position and phase is obtained when the width for the phase is 1. In practice selecting a different width σ for each input requires multi-variate RBF nodes, but that poses no special problems.

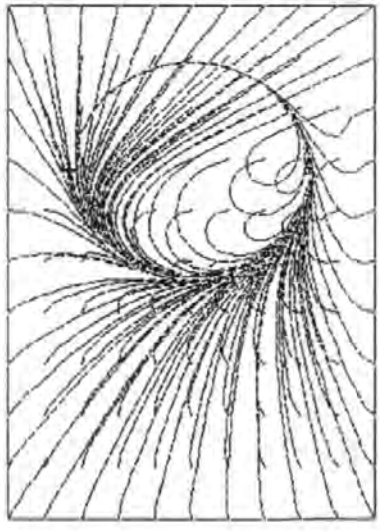


Fig. 5. Illustration of the attraction field generated by the neural network. The figure shows simulated trajectories with various starting positions. The initial phase is set to 0.5, so that only the first node can initially be active (and determine the target of the motion). Its activity however sets the phase to 1 which enables the succeeding node to become active, and so on. This causes a progressive curvature of the simulated trajectory towards nearer nodes on the demanded trajectory. Only the initial steps of the trajectory are shown. The cross marks the position of the receptive field of the first node.

B) *Learning*

In this application, the trajectory was defined in advance and the weights of the network were set accordingly. However, the trajectory can also be learnt on the spot. In this case, a user pushes the wheelchair through the desired trajectory, while the neural network progressively recruits new nodes in layer L_1.

Due to the attraction field, only the desired trajectory needs to be learnt. The wheelchair can then enter into the trajectory from any starting point and recover from deviations.

C) *Aliasing*

In this work the *(x,y)* position of the vehicle was used as input to the sequence encoding network. Aliasing occurs when the same (or similar) position reoccurs at different times in the trajectory. By adding a phase node we have avoided that the vehicle jumps from one phase of the trajectory to another, hence solving the aliasing problem.

In Perception-to-Action systems, aliasing is also a problem [7]. The difference is that some (position specific) complex sensory picture is used instead of the *(x,y)* position. It should also be possible to avoid aliasing in these systems by adding phase information to the picture.

VI CONCLUSION

A simple neural network has been described that encodes trajectories in a stable way, allowing recovery from disturbances and implementing a new phase encoding principle that solve the aliasing problem.

VII ACKNOWLEDGEMENT

The authors gratefully acknowledge support in many forms by British Aerospace Systems & Equipment (Plymouth), Mike Denham, Peter Frere (Lucas Advanced Engineering Centre, Birmingham), Steve Hill, the Henry Moore Foundation, David Keating, Peter Nurse, Penny and Giles Drives Technology Ltd, Plymouth Disability Equipment Centre, Paul Robinson, Alan Simpson, the South London Gallery, and others mentioned in the web page: http://www.tech.plym.ac.uk/soc/research/neural/research/wheelc.htm

VIII REFERENCES

[1] K. Althoefer and G. Bugmann (1995) "Planning and Learning Goal-Directed Sequences of Robot-Arm movements", in Fogelman-Soulié F. and Gallinari P. (eds), *Proc. of the International Conference on Artificial Neural Networks (ICANN'95)*, Paris, Vol. 1, 449-454.

[2] G. Bugmann (1996) "A note on the use of weight-averaging output nodes in RBF-based mapping nets", Technical Report CNAS-96-01.

[3] D.S. Broomhead and D. Lowe (1988) "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems*, 2, pp. 321-355.

[4] M. Brown and C. Harris (1994) *Neurofuzzy Adaptive Modelling and Control*, Prentice Hall, Hemel Hempstead, UK

[5] J. Moody and Ch. Darken (1989) "Fast learning in networks of locally-tuned processing units.", *Neural Computation*, 1, 281-294.

[6] M.J.D. Powell (1987) "Radial Basis Functions for Multivariate Interpolation: A Review", in Mason J.C. and Cox M.G. (eds) *Algorithms for Approximation*, Clarendon Press, Oxford, pp. 143-167.

[7] R.P.N. Rao and O. Fuentes (1996) "Learning Navigational Behaviour using a Predictive Sparse Distributed Memory", *Proc. of From Animals to Animats: The Fourth International Conference on Simulation of Adaptive Behaviour*, MIT Press, pp. 382-390.

[8] J. Shao, Y.V. Kee and R. Jones (1993) "Orthogonal Projection Method for Fast On-Line Learning Algorithm of Radial Basis Function Neural Networks", Proc. INNS World Congres on Neural Networks, Portland Oregon, USA, Vol.3, pp. 520-535.

Appendix C

This CD-ROM contains video clips and the program source codes.

1. Video Clips demonstrating the system performing the navigation tasks

- **Concurrent Control** -The system uses the concurrent control strategy developed in the thesis.
  Take 1.1, 1.2, 1.3, 1.4 and 1.5 – illustrate typical paths taken by the robot when the position of the obstacle is located at the centre left of the robot environment. In all these cases, the goal is located near the top right corner of the robot environment. Note that these takes correspond to some of the paths shown in figure 7.4(c).
  Screen 1 – illustrates a typical screen shoot of the remote brain's process when the position of the obstacle is located at the centre left of the robot environment.
  Take 2.1 – illustrates a typical path taken by the robot where the position of the obstacle is located at the centre right of the robot environment. In this case, the goal is located near the top right corner of the robot environment. Note that this take corresponds to one of the paths shown in figure 7.4(d).

- **Sequential Control** - The system here is using a control strategy described in the thesis as "sequential control".
  Take 3.1 and 3.2 – illustrate typical paths taken by the robot when the position of the obstacle is located at the centre left of the robot environment. In all these cases, the goal is located near the top right corner of the robot environment. Note that these takes correspond to some of the paths shown in figure 7.4(a).
  Screen 3 – illustrates a typical screen shoot of the remote brain's process when the position of the obstacle is located at the centre left of the robot environment.
  Take 4.1 and 4.2 – illustrate typical paths taken by the robot when the position of the obstacle is located at the centre right of the robot environment. In all these cases, the goal is located near the top right corner of the robot environment. Note that these takes correspond to some of the paths shown in figure 7.4(b).
  Screen 4 – illustrates a typical screen shoot of the remote brain's process when the position of the obstacle is located at the centre right of the robot environment.

  Note: Here the robot receives a waypoint from the remote brain instead of a simple motion command. Therefore, it does not stop as often as shown in figure 7.3.a).

2. Program Source Codes of the Computer System "Remote Brain"

3. Program Source Code of the Robotic System

4. Program Source Code of the Robot Tracking with Overhead Camera

Bibliography

Allotta, B., Conticelli, F. and Colombo, C. (1998). Asymptotically Stable Visual Servoing of 6-DOF Manipulators. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 101-108.

Althöfer, K. and Bugmann, G. (1995). Planning and Learning Goal-directed Sequence of Robot Arm Movements. *Proceedings of ICANN'95*, Volume 1, Pages 449-454.

Anderson, R. L., Alvertos, N. and Hall, E. L. (1982). Omnidirectional Real Time Imaging Using Digital Restoration. SPIE High Speed Photography, Vol. 348.

Andrews, J. R. and Hogan, N. (1983). Impedance Control as a Framework for Implementing Obstacle Avoidance in Manipulator. *Control of Manufacturing Processes and Robotic Systems*, Eds. Hardt, D. E. and Book, W., ASME, Boston, Pages 243-251.

Asoh, H., Motomura, Y., Asano, F., Hara, I., Hayamizu, S., Itou, K., Kurita, T. and Matsui, T. (2001). Jijo-2: An Office Robot That Communicates and Learns. *IEEE Intelligent Systems*, Volume 16, No.5, Pages 46-55.

Atiya, S. and Hager G.D. (1993). Real-Time Vision-Based Robot Localization. *IEEE Transactions on Robotics and Automation*, Volume 9, No.6, Pages 785-800.

Bak, M., Larsen, T. D., Norgaard, M., Andersen, N. A., Poulsen, N. K. and Ravn, O. (1998). Location Estimation using Delayed Measurements.
(Available at http://www.iau.dtu.dk/~tdl/amc98paper.ps.gz)

Ballard, D. H. and Brown, C. M. (1982). *Computer Vision*. Prentice Hall, New Jersey.

Beck, C. (1925). Apparatus to Photograph the Whole Sky. *Journal of Scientific Instrumention*, Volume 2, Pages 135-139.

Belongie, S., Carson, C., Greenspan, H. and Malik., J. (1998). Color- and texture-based image segmentation using the expectation-maximization algorithm and its application to content-based image retrieval. *In Proceeding of the Sixth International Conference on Computer Vision*, Pages 675-682.

Bezdek, J. C., Hall, L. O. (1993). Review of MR image segmentation techniques using pattern recognition, *Medical Physics*, Volume 20, No. 4, Pages 1033-1048.

Bialkowski, W. L. (1983). Application of Kalman Filters to the Regulation of Dead Time Processes. *IEEE Transactions on Automatic Control*, Volume 28, No. 3, Pages 400-406.

Borenstein, J. and Koren, Y. (1989). Real-time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man, and Cybernetic*, Volume 19, No. 5, Pages 1179-1187.

Borenstein, J. and Koren, Y. (1991a). Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento, California, Pages 1398-1404.

Borenstein, J. and Koren, Y. (1991b). The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robot. *IEEE Transactions on Robotics and Automation*, Volume 7, No. 3, Pages 278-288.

Borenstein, J., Everett, H. R. and Feng, L. (1996). Where am I? Sensors and Methods for Mobile Robot Positioning. Edited and compiled by J. Borenstein, University of Michigan.
(Available at http://www-personal.umich.edu/~johannb/shared/pos96rep.pdf)

Broomhead, D.S. and Lowe, D. (1988). Multivariable Functional Interpolation and Adaptive Networks, *Complex Systems*, Volume 2, Pages. 321-355.

Brown, M. and Harris, C. (1994). *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall, Hemel Hempstead, UK

Brown, R. G. and Donald, B.R. (2000). Mobile robot self-localization without explicit landmarks. *Algorithmica*, Volume 26, No. 3-4, Pages. 515-559.

Bugmann, G., Taylor, J. G. and Denham, M. J. (1994). Route Finding by Neural Nets. *Application of Modern Heuristic Methods: Neural Networks*, J.G. Taylor (ed), Unicom & Alfred Waller Ltd. Publ., Pages 217-230.

Bugmann, G.(1996). A note on the use of weight-averaging output nodes in RBF-based mapping nets, *Technical Report CNAS-96-01*.

Bugmann, G., Koay, K. L., Barlow, N., Phillips, M. and Rodney, D. (1998). Stable Encoding of Robot Trajectories using Normalised Radial Basis Functions: Application to an Autonomous Wheelchair. *Proceeding of the 29th Intl. Symp. Robotics (ISR'98)*, 27-30 April, Birmingham, UK, Pages 232-235. DMG Publishers: London.

Bugmann, G. (1998). Normalized Radial Basis Function Networks. *Neurocomputing (Special Issue on Radial Basis Function Networks)*, 20, Pages 97-110. (ISSN : 0925-2312)

Buhmann, J., Burgard, W., Cremers, A. B., Fox, D., Hofmann, T., Schneider, F. E., Strikos, J. and Thrun. S. (1995). The Mobile Robot Rhino. *AI Magazine*, Volume 16, No. 2, Pages31-38.

Chang, C. C. and K. T. Song (1997). Environment Prediction for a Mobile Robot in a Dynamics Environment. *IEEE Transactions on Robotics and Automation*, Volume 13, No. 6, Pages 862-872.

Choi, J., Sellen, J. and Yap. C. K. (1994). Approximate Euclidean shortest path in 3-space. *Proceedings of the 10th ACM Symposium on Computational Geometry*, Pages 41-48.

Connolly, C. I., Burns, J. B. and Weiss R. (1990). Path planning using Laplace's Equation. *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, Pages 2102-2106.

Cox, I. J. (1989). Blanche: Position Estimation for an Autonomous Robot Vehicle. *IEEE/RSJ International workshop on Intelligent Robots and Systems*, Pages 432-439.

Cox, I. J. (1991). Blanche–An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *IEEE Transactions on Robotics and Automation*, Volume 7, No. 2 Pages 193-204.

Crevier, D. (1993). AI: The Tumultuous History of the Search for Artificial Intelligence. Pages.115. *Basic Books* (*Harper Collins*), New York.

DeSouza G. N. and Kak A. C. (2002). Vision for Mobile Robot Navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 24, No. 2, Pages 237-267.
(Available at http://rvl1.ecn.purdue.edu/Projects/MobileRobotics/pami.ps)

Despande, P. B. and Ash, R. H. (1988). Computer Process Control. *ISA Pub*, 2nd Ed.

Everett, H. R.(1995). *Sensors for Mobile Robots: Theory and Application.* A K Peters Ltd., ISBN: 1568810482.

Fikes, R., Nilsson, N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, Volume 2, Pages 189-208

Giuffrida, F., Massucco, C., Morasso, P., Vercelli, G., Zaccaria, R. (1995). Multi-Level Navigation Using Active Localisation System

Golovan, A. A. and L.A. Mironovskii (1993). An Algorithmic Control of Kalman Filters. *Automation and Remote Control*, Volume 54, No. 7 Pt 2, Pages 1183-1194.

Haralick, R. M., and Shapiro, L. G. (1985). Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, Volume 29, Pages 100-132.

Hu, H., J.M. Brady, J. Grothusen, F. Li and P.J. Probert (1995). LICAs: A Modular Architecture for Intelligent Control of Mobile Robots. *International Conference on Intelligent Robots and Systems*, Volume 1, Pages 471-476.

Iyengar, S. S. Jorgensen, C. C., Rao, S. V. N. and Weisbin, C. R. (1986). Robot Navigation Algorithms Using Learned Spatial Graphs. *Robotica*, Volume 4, Part 2, Pages 93-100.

Janet, J.A., Gutierrez-Osuna, R., Chase, T.A., White, M. and Luo, R.C. (1995). Global Self-Localization for Autonomous Mobile Robots Using Self-Organizing Kohonen Neural Networks. *Proceeding IEEE/RJS International Conference on Intelligent Robots and Systems*, Volume 3, Pages 504-509

Jasiobedzki, P. (1995). Detecting Driveable Floor Regions. *International Conference on Intelligent Robots and Systems*, Volume 1, Pages 264-270.

Jensfelt, P. and Kristensen, S. (1999). Active global localisation for a mobile robot using multiple hypothesis tracking. *In Proc. of the IJCAI-99 Workshop on Reasoning with Uncertainty in Robot Navigation*, Pages 13-22.

Jensfelt, P. (2001). Approaches to Mobile Robot Localization in Indoor Environments. *PhD thesis*, Department of Signals, Sensors and Systems, Royal Institute of Technology (Kungl Tekniska Hogskolan).

Jiang, K., Seneviratne, L. D. and Earles, S. W. E. (1997). Time-optimal smooth-path motion planning for a mobile robot with kinematic constraints. *Robotica*, Volume 15, No.5, Pages 547-553.

Jiang, K. C., Seneviratne , L. D. and Earles, S. W. E. (1999). A shortest path based path planning algorithm for nonholonomic mobile robots. *Journal of Intelligent and Robotic Systems*, 24, Pages 347-366.

Jogan, M. and Leonardis, A. (2000). Robust localization using panoramic view-based recognition. *Proceeding of 15th International Conference of Pattern Recognition*, Volume 4, Pages 136-139.

Jones, J. L. and Flynn, A. M. (1993). *Mobile Robots Inspiration to Implementation.* A K Peters Ltd.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, Pages 35-45.

Kambhampati, C., A Delgado, J. D. Mason and K. Warwick (1997). Stable Receding Horizon Control Based on Recurrent Networks. *IEE Proceeding of the Control Theory Appl.*, Volume 144, No. 3, Pages 249-254.

Khatib, O. (1985). Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, Pages 500-505.

Ko, W., Seneviratne , L. D. and Earles, S. W. E. (1994). Extended triangulation algorithm for robot path planning with obstacle avoidance. *Proceeding Engineering Systems Design and Analysis*. American Society of Mechanical Engineers, Volume 6, Pages 101-108.

Koay, K. L., Bugmann, G., Barlow, N. and Philips, M. (1998). Representation of Trajectories for Mobile Robot. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 185-194.

Koreichi, M. L., Babaci, S., Chaumette, F., Fried, G., and Pontnau, J. (1998). Visual Servo Control of A Parallel Manipulator for Assembly Tasks. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 109-116.

Kosaka, A. and Kak, A. C. (1992). Fast Vision-Guided Mobile Robot Navigation Using Model-Based Reasoning and Prediction of Uncertainties. *Computer Vision, Graphics, and Image Processing – Image Understanding*, Volume 56, No. 3, Pages 271-329.

Kosaka, A., Meng, M. and Kak, A. C. (1993). Vision-Guided Mobile Robot Navigation Using Retroactive Updating of Position Uncertainty. *Proceeding of IEEE International Conference on Robotics and Automation.* Volume 2, Pages 1-7

Kosaka, A. and Nakazawa, G. (1995). Vision-Based Motion Tracking of Rigid Objects Using Prediction of Uncertainties. *Proceeding of IEEE International Conference on Robotics and Automation,* Pages 2637-2644.

Krogh, B. H. (1984). A Generalized Potential Field Approach to Obstacle Avoidance Control. *First World Conference on Robotics Research*, RIA.

Kuipers, B., and Byun, Y. T. (1991). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, Volume 8, Pages 47-63

Kwon, Y. D. and Lee, J. S. (1995). An Obstacle Avoidance Algorithm for Mobile Robot: the Improved Weighted Safety Vector Field Method, *10th IEEE/ International Symposium on Intelligent Control*, Monterey, CA.

Kwon, Y. D. and Lee, J. S. (1996). A Local Path Generation Method using Obstacle Vectors and Via Points. *World Automation Congress.*

Kwon, W. H., P. S. Kim and P. Park (1999a). A Receding Horizon Kalman FIR Filter for Discrete Time-Invariant Systems. *IEEE Transactions on Automatic Control*, Volume 44, No. 9.

Kwon, W. H., P. S. Kim and P. Park (1999b). A Receding Horizon Kalman FIR Filter for Linear Continuous-Time Systems. *IEEE Transactions on Automatic Control*, Volume 44, No. 11.

Larsen, T. D., Andersen, N. A. and Ravn, O. (1988). Incorporation of Time Delayed Measurements in a Discrete-time Kalman Filter. *In To appear in proceedings for CDC '98*, Tampa, Florida.

Latombe, J. C. (1991). *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA.

Levine, W. S. (1996). *The Control Handbook.* CRC Press, ISBN: 0849385709.

Lee, D. T. and Drysdale, R. L. (1981). Generalize Voronoi Diagrams in a Plane. *SIAM Journal of Computing*, Volume 10, No. 1, Pages 73-87.

Li, S., Nagata, S. and Tsuji, S. (1995). A Navigation System Based upon Paranomic Representation. *International Conference on Intelligent Robots and Systems*, Volume 1, Pages 142-147.

Lorigo, L. M., Brooks, R. A. and Grimson, W. E. L. (1997). Visually-Guided Obstacle Avoidance in Unstructured Environments. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pages 373-379.

Lozano-Perez, T. and Wesley, M. (1979). An Algorithm for planning collision-free paths among polyhedral Obstacles. *Communications of the ACM*, Volume 2, No. 3. Pages 560-570.

Lui, B., Choo, S. H., Lok, S. L., Leong, S. M., Lee, S. C., Poon, F. P. and Tan, H. H. (1994). Finding the Shortest Route Using Cases, Knowledge, and Dijkstra's Algorithm. *IEEE Expert*, Pages 7-11.

Maeyama, S., Ohya, A. and Yuta, S. (1995). Non-stop outdoor navigation of a mobile robot---Retroactive positioning data fusion with a time consuming sensor system---. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pages 130-135.

Marshall, J. E. (1979). Control of Time-Delay System. *IEE Control Engineering Series 10.*

Maybeck, P. S. (1979). *Stochastic Models, Estimation, and Control.* Volume 1. Academic Press, Inc.

Mayne, D. Q. and Michalska, H. (1990). Receding Horizon Control of Nonlinear Systems. *IEEE Transactions on Automatic Control*, Volume 35, No. 7, Pages 400-406.

Miall, R. C., Weir, D. J., Wolpert, D. M. and Stein, J. F. (1993). Is the Cerebellum a Smith Predictor?. *Journal of Motor Behaviour*, Volume 25, No. 3, Pages 203-216.

Miyamoto, K. (1964). Fish Eye Lens. *Journal Letter*, Volume 54, Pages 1060-1061.

Molton, N., Se, S., Brady, J. M., Lee, D. and Probert, P. (1988). A Stereo Vision-Based Aid for the Visually Impaired. *Image and Vision Computing*, Volume 16, No. 4, Pages 251-263.

Moody, J. and Darken, Ch. (1989). Fast learning in networks of locally-tuned processing units, *Neural Computation*, Volume 1, Pages 281-294.

Moravec, H. P. (1983). The Stanford Card and the CMU Rover. *Proceedings of the IEEE*, Volume 71, No. 7, Pages 872-884.

Moravec, H. and A. Elfes (1985). High Resolution Maps from Wide Angle Sonar. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'85)*, Pages 116-121.

Morel, J.-M. and Solimini, S. (1995). *Variational Methods in Image Segmentation.* Boston, MA, Birkhauser, ISBN: 0-8176-3720-6.

Murray, D. and Jennings, C. (1997). Stereo Vision Based Mapping and Navigation for Mobile Robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, Pages 1694-1699.

Murray, D. and Little, J (1998). Using Real-Time Stereo Vision for Mobile Robot Navigation. *Proceeding of the IEEE Workshop on Perception for Mobile Agent*, Pages 19-27

Nilsson, N. (1969). A Mobile Automaton: An Application of Artificial Intelligence Techniques. *Proceedings IJCA. Reprinted in Autonomous Mobile Robots: Control, Planning and Architecture*. Volume 2, Pages 233-239

Nilson, N. J. (1982). *Principles of Artificial Intelligence*. Springer-Verlag, New York.

Ohya, A., Kosaka, A. and Kak, A. (1998). Vision-Based Navigation by a Mobile Robot with Obstacle Avoidance using Single-Camera Vision and Ultrasonic Sensing. *IEEE Transactions on Robotics and Automation*, Volume 14, No. 6, Pages 969-978.

O'Dunlaing, and Yap, C. K. (1983). The Voronoi Method for Motion-Planning 1: The Case of a Disk, *Technical Report 53*, Courant Institute.

O'Dunlaing, and Yap, C. K. (1985). A Retraction Method for Planning the Motion of a Disk. *J. Algorithm*, Volume 6, Pages 104-111.

Okutomi, M. and Kanade, T. (1993). A Multiple-Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 15, No.4, Pages 353-363.

Onoguchi, K., Takeda, N. and Watanabe, M. (1995). Planar Projection Stereopsis Method for Road Extraction. Pages 249-256.

Otega, J. G. and Camacho, E. F. (1996). Mobile Robot Navigation in a Partially Structured Static Environment, Using Neural Predictive Control. *Control Engineering Practice*, Volume 4, No. 12, Pages 1669-1679.

Pal, N. R. and Pal, S. K. (1993). A review on image segmentation techniques, *Pattern Recognition*, Volume 26, No. 9, Pages 1277-1294.

Paletta, L., Prantl, M. and Pinz, A.(1998). Reinforcement Learning for Autonomous Three-Dimensional Object Recognition. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 63-72.

Pappas, T. N. (1992). An adaptive clustering algorithm for image segmentation, *IEEE Transactions on Signal Processing*, Volume 40, Pages 901-914.

Pocchiola, M. and Vegter, G. (1995). Computing the visibility graph via pseudo-triangulations. *Proceedings 11th ACM Annual Symposium on Computational Geometry*, Pages 248-257.

Powell, M. J. D. (1987). Radial Basis Functions for Multivariate Interpolation: A Review, in Mason J.C. and Cox M.G. (eds) *Algorithms for Approximation*, Clarendon Press, Oxford, Pages 143-167.

Ricardo, S. O., Michel, D. and Viviane, C. (1998). Controlling the Execution of a Visual Servoing Task. *Proceedings of the 6th International Symposium on Intelligent Robotics Systems*, Pages 127-136.

Rao, R. P. N and Fuentes, O. (1996). Learning Navigational Behaviour using a Predictive Sparse Distributed Memory, *Proceeding of From Animals to Animats: The Fourth International Conference on Simulation of Adaptive Behaviour*, MIT Press, Pages 382-390.

Roberts, L. G. (1965). Machine Perception of Three Dimensional Solid, in Optical and Electro-optical Information Processing. Ed. J. P. Tipper et al., MIT Press, Cambridge, Massachusetts.

Ronco, E. Arsan, T. and Gawthrop, P. J. (1998). Open-Loop Intermittent Feedback Optimal Predictive Control: Practical Continuous-time GPC. *IEE Proceedings on Control Theory and Applications*.

Ronco, E. (1998). Open-Loop Intermittent Feedback Optimal Control: a probable human motor control strategy. *Technical Report: EE-98005*, Systems and Control Laboratory, University of Sydney.
(Available at http://merlot.ee.usyd.edu.au/tech_rep/EE98005.html).

Ronco, E. and Hill, D. J. (1999). Open-Loop Intermittent Feedback Optimal Predictive Control: a human movement control model. *Technical Report: EE-99003*, School of Electrical and Information Engineering, University of Sydney.

Schwartz, J. T. and Sharir, M. (1983a). On the Piano Movers' Problem I: The special case of a rigid polygonal body moving amidst polygonal barriers. Commun. Pure Appl. Math., 36, Pages 345-398.

Schwartz, J. T. and Sharir, M. (1983b). On the Piano Movers' Problem II: General techniques for computing topological properties of real algebraic manifolds. Adv. Appl. Math., 4, Pages 298-351.

Schwartz, J. T. and Sharir, M. (1983c). On the Piano Movers' Problem III: Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers. *Robotics Research*, Vol. 2, No. 3, Pages 46-75.

Se, S., Lowe, D. and Little, J (2001). Vision-Based Mobile Robot Localization and Mapping using Scale-Invariant Features. *Proceedings of the IEEE International Conference on Robotics and Automation*, Pages 2051-2058.

Seneviratne, L. D., Ko, W. S. and Earles, S. W. E. (1997). Triangulation-based path planning for a mobile robot. *Proceeding IMechE, Part C, Journal of Mechanical Engineering Science*, Volume 211, No. 5, Pages 365-371.

Shah, S. and Aggarwal, J. K. (1994). A Simple Calibration Procedure for Fish-Eye (High Distortion) Lens Camera. *International Conference on Robotics and Automation*, Munich, Germany, Pages 3422-3427.

Shao, J., Kee Y. V and Jones, R. (1993). Orthogonal Projection Method for Fast On-Line Learning Algorithm of Radial Basis Function Neural Networks, *Proceeding of the INNS World Congres on Neural Networks*, Portland Oregon, USA, Volume 3, Pages 520-535.

Shilman, S. V. (1994). Adaptive Kalman Filters, *Doklady Akademii Nauk*, Volume 338, No. 6, Pages 742-744.

Smith, O. J. (1959). A Controller to Overcome Dead Time. *ISA J.*, Volume 6, no. 2, Pages 28-33.

Takahashi, O. and Schilling, R. J. (1989). Motion Planning in a Plane using Generalized Voronoi Diagrams. IEEE Transactions on Robotics and Automation, Volume 5, No.2, Pages 143-150.

Thrun, S. and Bucken, A. (1996). Integrating Grid-Based and Topological Maps for Mobile Robot Navigation. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Pages 944-950.

Thrun, S. (1998). Learning Metric-Topological Maps for Indoor Mobile Robot Navigation, *Artificial Intelligence*, Volume 99, No. 1, Pages 21-71.

Thrun, S., Gutmann, J-S., Fox, D., Burgard, W. and Kuipers, B. (1998). Integrating Topological and Metric Maps for Mobile Robot Navigation: A Statistical Approach. *Proceedings of the 15th AAAI Conference*. Pages 989-996.

Tomatis, N., Nourbakhsh, I., Siegwart, R. (2001). Simultaneous Localization and Map Building: A Global Topological Model with Local Metric Maps. *Proceeding of the IEEE/RJS International Conference on Intelligent Robots and Systems.*

Udupa, S. (1977). Collision Detection and Avoidance in Computer Controlled Manipulators. *Ph.D. Dissertation*, Dept. of Electrical Engineering, California Institute of Technology, Pasadena, CA.

Uhrmeister, B. (1994). Kalman Filters for A Missile with Radar and/or Imaging Sensor. *Journal of Guidance Control and Dynamics*, Volume17, No.6, Pages1339-1344.

Vasseur, H.A., Pin, F.G. and Taylor, J.R. (1991). Navigation of a Car-like mobile robot using a ecomposition of the environment in convex cells. *Proceeding of the IEEE International Conference on Robotics and Automation*, Pages 1496-1502.

Vandoren , V. J. (1996). The Smith Predictor: A Process Engineer's Crystal Ball, Control Engineering, Pages 61-62.

Vlassis, N., Motomura, Y., Hara, I., Asoh, H. and Matsui, T. (2001). Edge-based features from omnidirectional images for robot localization. *Proceeding of the IEEE International Conference on Robotics and Automation.* Pages 1579-1584.

Weng, J., Cohen, P. and Herniou, M. (1992). Camera Calibration with Distortion Models and Accuracy Evaluation. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, Volume 14, No. 10, Pages 965-980.

Wilcox, B. H., Gennery, D. B., Mishkin, A.H., Cooper, B. K., Lawton, T. B., Lay, N. K. and Katzmann, S. P. (1987). A Vision System for a Mars Rover. *SPIE Mobile Robots II*, Volume 852, Pages 172-179.

Wilfong, G.T. (1988). Motion Planning for an Autonomous Vehicle. *Proceeding of the IEEE International Conference on Robotics and Automation*, Pages 529-533.

Williams, C. and Becklund, O. (1972). *Optics: A Short Course for Engineers & Scientists.* John Wiley & Sons, Inc.