2018

Action Learning Experiments Using Spiking Neural Networks and Humanoid Robots

de Azambuja, Ricardo

Copyright Statement

Plymouth University

Doctoral Thesis

Action Learning Experiments Using Spiking Neural Networks and Humanoid Robots

Author:
Ricardo de Azambuja

Supervisors:
Dr. Angelo Cangelosi
Dr. Martin F. Stoelen

A thesis submitted to Plymouth University
in partial fulfilment for the degree of

Doctor of Philosophy

in the

Centre for Robotics and Neural Systems
School of Computing, Electronics and Mathematics

27th January 2018

Acknowledgements

First, I would like to express my gratitude to Angelo Cangelosi for all the support I received since I've arrived from Brazil. He was the best mentor I could have, I will consider him as *family* forever and that means he will never be able to get rid of me now ☺.

To my friend, second supervisor and boss at the soon-to-be agriculture robotics superpower Fieldwork Robotics Ltd, Martin F. Stoelen, that helped me a lot in many ways, opened many doors and even found time to play volleyball with me, my very *soft robotic* thanks!

During my whole PhD I'd kept a close relationship with Brazil, thanks to my friend and previous master's supervisor Valner J. Brusamarello. Because of him, I could met, unofficially supervise and start a nice friendship with Davi A. Sala. Thank you both and I hope the collaboration between UFRGS and University of Plymouth will only be strengthen from now on.

I could not forget to say thank you to my previous second supervisor, Samantha V. Adams, for introducing me to the *spiking neuron black magic*.

Also, many, many thanks for all the CRNS people. You know, too many names to cite (*spoc...*), but I can't avoid mentioning Daniel and Abdulla for co-authoring nice papers with me, my *good ol'* friend Frederico (for listening, *googling*, nodding and reproducing Nigella's best recipes) and, of course, the A225 people: Massimiliano (hackathon *bro*), Debora (free Ψ advice), Leszek (thought me how to say *dzień dobry*) and Giovanni (helped me reaching the required *illumination* to finish the last chapter of my thesis).

Finally, all this work would not be possible without the support and encouragement that I received from my wife Daniele and my son Enrico. They are the real heroes here for tolerating four year of bad mood, craziness, lack of husband/dad because of many trips to conferences, summer schools, meetings and workshops as well laptop noises during my usual extended night shift work. They know I love them and I hope all their sacrifice had worth it.

Most of things I've achieved during my PhD were only possible because of open source and free software available. I simply can not mention everything, but thanks to Python, Scipy, Matplotlib, Scikit-learn, Brian, LaTeX, texmaker, Zotero, Atom, ... the list is way too big to put it here. As a retribution, all the code I've generated is also available online on my repository: github.com/ricardodeazambuja.

This work was in part supported by the CAPES Foundation, Ministry of Education of Brazil (scholarship BEX 1084/13-5) and UK EPSRC project BABEL (EP/J004561/1 and EP/J00457X/1).

Declaration of Authorship

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Graduate Sub-Committee.

Work submitted for this research degree at the Plymouth University has not formed part of any other degree either at Plymouth University or at another establishment.

A programme of advanced study was undertaken, which included participation in the BABEL project, participation in a total of 21 courses organised within the Researcher Development Programme of the Plymouth University Graduate School, participation in the 2014 CapoCaccia Cognitive Neuromorphic Engineering Workshop, participation and successful completion of the GTA - An accredited course for General Teaching Associates (Intensive), participation in the Introduction to CUDA, Optimisation steps for neural networks with CUDA and Deep learning with DIGITS, participation in the Nengo Summer School 2016, participation in the Sixth SpiNNaker Workshop 2016, participation in the MILA - Deep Learning and Reinforcement Learning Summer Schools 2017, and participation in the supervision of Davi Alberto Sala during his MSc. project focused on Sensor Fusion and Liquid State Machines.

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

List of publications more closely related to the work developed in this thesis:

- **R. de Azambuja**, A. Cangelosi, and S.V. Adams. "Diverse, Noisy and Parallel A New Spiking Neural Network Approach for Humanoid Robot Control." In 2016 International Joint Conference on Neural Networks (IJCNN), 1134–42. Vancouver, 2016. doi:10.1109/IJCNN.2016.7727325.
- **R. de Azambuja**, F. B. Klein, M. F. Stoelen, S. V. Adams, and A. Cangelosi. "Graceful Degradation Under Noise on Brain Inspired Robot Controllers." In Neural Information Processing, edited by Akira Hirose, Seiichi Ozawa, Kenji Doya, Kazushi Ikeda, Minho Lee, and Derong Liu, 195–204. Lecture Notes in Computer Science 9947. Springer International Publishing, 2016. doi:10.1007/978-3-319-46687-3_21.

- **R. de Azambuja**, F.B. Klein, S.V. Adams, M.F. Stoelen and A. Cangelosi. "Short-Term Plasticity in a Liquid State Machine Biomimetic Robot Arm Controller." In 2017 International Joint Conference on Neural Networks (IJCNN). Anchorage, 2017. doi:10.1109/IJCNN.2017.7966283.
- **R. de Azambuja**, D.H. García, M.F. Stoelen and A. Cangelosi. "Neurorobotic Simulations on the Degradation of Multiple Column Liquid State Machines." In 2017 International Joint Conference on Neural Networks (IJCNN). Anchorage, 2017. doi:10.1109/IJCNN.2017.7965834.
- D. A. Sala, **R. de Azambuja**, V. J. Brusamarello and A. Cangelosi "Positioning Control on a Collaborative Robot by Sensor Fusion with Liquid State Machines." In Instrumentation and Measurement Technology Conference (I2MTC), 2017 IEEE International. Milano, 2017. doi:10.1109/I2MTC.2017.7969728.

Presentations in Conferences:

- The 2016 International Joint Conference on Neural Networks:
  "Diverse, Noisy and Parallel A New Spiking Neural Network Approach for Humanoid Robot Control.". Oral presentation.
- The 23rd International Conference on Neural Information Processing:
  "Graceful Degradation Under Noise on Brain Inspired Robot Controllers.". Oral presentation.
- The 2017 International Joint Conference on Neural Networks:
  "Neurorobotic Simulations on the Degradation of Multiple Column Liquid State Machines.". Oral presentation.
- The 2017 International Joint Conference on Neural Networks:
  "Short-Term Plasticity in a Liquid State Machine Biomimetic Robot Arm Controller.". Poster presentation.

Word count of main body of thesis: 38,270.

Signed: _____

Date: _____

Abstract

Ricardo de Azambuja

Action Learning Experiments Using Spiking Neural Networks and Humanoid Robots

The way our brain works is still an open question, but one thing seems to be clear: biological neural systems are computationally powerful, robust and noisy. Natural nervous system are able to control limbs in different scenarios with high precision. As neural networks in living beings communicate through spikes, modern neuromorphic systems try to mimic them by using spike-based neuron models. This thesis is focused on the advancement of neurorobotics or brain inspired robotic arm controllers based on artificial neural network architectures. The architecture chosen to implement those controllers was the spike neuron version of Reservoir Computing framework, called Liquid State Machines. The main goal is to explore the possibility of using brain inspired neural networks to control a robot by demonstration. Moreover, it aims to achieve systems robust to environmental noise and internal structure destruction presenting a graceful degradation. As the validation, a series of action learning experiments are presented where simulated robotic arms are controlled. The investigation starts with a 2 degrees of freedom arm and moves to the research version of the Rethink Robotics Inc. collaborative humanoid robot Baxter. Moreover, a proof-of-concept experiment is also done using the real Baxter robot. The results show Liquid State Machines, when endowed with an extra external feedback loop, can be also employed to control more complex humanoid robotic arms than a simple planar 2 degrees of freedom one. Additionally, the new parallel architecture presented here was capable to withstand noise and internal destruction better than a simple use of multiple columns also presenting a graceful degradation behaviour.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artificial neural network
API	Application programming interface
CPG	Central pattern generator
CPU	Central processing unit
DOF	Degree of freedom
DTW	Dynamic time warping
LSM	Liquid state machine
ROS	Robot operating system
SEB	Single-event burnout
SEGR	Single-event gate rupture
SEL	Single-event latchup
SEU	Single-event upset
SNN	Spiking neural network
STP	Short-term plasticity
SWN	Small-world network

To Daniele and Enrico.

Chapter 1

Introduction

1.1 Motivations

The first general purpose programmable microprocessor was released to the market by Intel Corporation in 1971. It was a 4-bit Central Processing Unit (CPU) called Intel 4004. Since then, Intel's CPUs went from a single core and 2,300 transistors working at 740kHz to more than 14 cores and about 6 billion transistors at frequencies above 2GHz. Despite all this evolution in computational power, currently, the software running on computers mostly follows the serial paradigm. Super computers and recently Graphical Processing Units (GPU) are well known for parallel computations, but the applications are restricted to ones that require the same instruction simultaneously applied to multiple data or they simply break the code into chunks of serial algorithms.

Current binary logic depends on determinism to solve any proposed problem. This leads to systems that cannot deal with unpredictability: a digital processor only accepts commands that are part of its instruction set. Such a design choice has a tendency to simply stop working when exposed to intensive noise or when a small part fails as in a serial system if one link of the chain is broken it could become impossible to advance to the next instruction or state. Also, current digital systems cannot deal with noise levels that are too close to their logic threshold voltages. An ideal system should degrade gracefully, i.e., its functions

should deteriorate according to the amount of harm inflicted to it, maintaining some limited functionality even after having large portions destroyed.

The lack of robustness to noise brings serious consequences. Six years have passed since Fukushima's nuclear disaster and current technology is still not ready for such a big challenge. The high level of radiation in areas close to the reactors was lethal for human beings and the robots sent to the site have severely suffered from it. Radiation exposition could result in destructive phenomena that occur at the silicon level like Single-Event Latchup (SEL), Single-Event Gate Rupture (SEGR), Single-Event Burnout (SEB) or even more drastic physical events, hence making clear the need for more research. Modern computers, and therefore robot controllers, are designed around digital circuits and, despite several advances in manufacturing processes, design and simulation, they are still not immune to it. Digital systems also suffer from non-destructive radiation effects, since radiation can generate Single-Event Upsets (SEU) or "soft-errors" [1]. In addition to man-made radiation sources, space and terrestrial environments are also subjected to cosmic rays and naturally available radioactive isotopes.

Contrasting with digital systems, natural neural systems have an innate ability to adapt to new experiences, work in noisy environments as well to degrade gracefully when partially damaged. Moreover, small living beings still can not yet be completely *simulated* even in the most powerful computer up to date.

There is a clear evolutionary pressure for natural information processing systems to be fault tolerant or robust. If neuronal cells were easily damaged or suffered malfunctioning this would change drastically the overall behaviour of an organism, definitely restricting its chances of survival. According to Rogers and McClelland [2], graceful degradation is defined as "graded, probabilistic deficits, with some sparing of function, and with performance strongly influenced by the frequency or familiarity of the stimulus and/or its degree of consistency with other items". As such, it's possible to identify graceful degradation in a number of neural systems.

Efficiency is another characteristic seen all around nature designs. In the natural world, the little transparent nematode (*C. elegans*) can go through all its life cycle challenges

with something around 300 neurons and it is still capable of succeeding. Scale it up by roughly 300 million times and we have the average human brain. Besides all that computational power, the human brain is a great example of efficiency, since cortex and cerebellum together spend on average around 15W [3]. On the other hand, the Human Brain Project expects to simulate the whole brain, in the cellular level, using an exascale computer or 60MW [4]. Additionally, in the particular case of autonomous robots, it is necessary to have enough computational power to deal with an unpredictable environment, but without running out of battery before the end of the mission.

Nowadays, the use of robots is already a reality, but this is a revolution that so far has real economical outcomes mostly in repetitive tasks on the factory floor. According to the International Federation of Robotics (IFR), in 2013 about 178 thousand industrial robots were sold in the world. Since the release of the first industrial robot, at the 1957 International Trade Fair on Automation in Stockholm, called Planobot [5], it is estimated that a total cumulative production of more than 2.5 million different types of specialized robots are currently used in the industry. Those robots are working in sectors from heavy industry to consumer goods.

Consequently, any advance made in this field has the possibility to generate quality improvements and even huge savings at the end of the production chain. This, by itself, is a big motivation and governments around the world have already noticed and started to increase the funding for research that involves any type of commercial exploitation of robots.

Modern industrial robots are very efficient in specialized tasks. They have the ability and stiffness to repeat the exact same movement several times keeping the accuracy (or bias) and precision (or random noise). Those same virtues make necessary to prepare the environment where these robots are deployed to protect human beings working near to the robot station. In addition, industrial robots in most of the cases cannot use the same tools as we do. One solution to those, problems broadly known in the research community, is the use of a robot that resembles the human body and behaves alike - a humanoid robot. Cangelosi et al. [6] define the term “humanoid robot” as a robot with an

anthropomorphic body and human-like senses. However, as the 2015 edition of the DARPA Robotics Challenge presented us, tasks as simple as opening a door or using a drill are still very hard even for state-of-the-art humanoid robots.

One of the problems faced when dealing with humanoid robots is the complexity of those systems. As an example, the humanoid robot iCub [7] has more than fifty degrees of freedom, i.e. free parameters that can be set to define how the robot is positioned, making it a redundant structure as there are an infinitude of possible ways to accomplish the same task. Some of the inspiration to solve the problems raised by the development of humanoid robots also come from nature.

Guigon et al. [8] suggest it is still unknown how a complex redundant redundant body is controlled by the nervous system. However, the direction of movement in a three dimensional space was predicted using direct readings from motor cortical neurons [9]. Combined efforts from neuroscience, robotics and artificial intelligence to answer this kind of questions made a new field emerge: Neurorobotics [10]. Kaplan [11], for example, defines it as "the science and technology of embodied autonomous neural systems".

Based on the neurorobotic approach, the work presented here applies artificial neural networks to control humanoid robots, more specifically to the robot's arm. The current literature [12] states that natural stimuli is made of spatio-temporal patterns and cortical neurons are naturally sensitive to those; therefore, any model of cortical processing needs to be able to deal with this information and, according to Gerstner and Kistler [13], spiking neural networks have this ability.

Despite several studies published in the domain of robot control using spiking neural networks (SNNs), relatively few works addressing the more specific humanoid robot control based on SNN can be found in the literature (e.g. [14, 15, 16, 17]).

A neurorobotic system interacting with a real-world scenario, in order to mimic what happens with a human being, needs to integrate numerous input signals from the visual system, vestibular system (balance), actuators (proprioceptive feedback), short term memory, long term memory and information from different parts of the neural system.

Following the neurorobotic outlook, good sources for inspiration are biological systems. A model to explain how human hand movements can be decomposed into a series of sub-movements was presented by Milner [18]. This is an interesting idea as it suggests simpler movements could be linearly added to generate more complex ones. In fact, experiments with spinalized frogs [19] also proved the existence of a linear behaviour between spinal cord sites and the endpoint forces. Using micro wire arrays implanted in the owl monkey (*Aotus trivirgatus*) cortex, Wessberg et al. [20] were able to predict three-dimensional hand trajectories using linear models or artificial neural networks.

An interesting biologically inspired artificial neural framework, Liquid State Machine (LSM), was introduced by Maass et al. [21] in 2002. LSMs are recurrent spiking neural networks where an external output layer called *readout* is the only part of the network that is subject to a learning process. One or more inputs are injected in a non-linear dynamical system, a group of artificial neurons modelling a column in the cortex (the *liquid*), and the disturbances generated are collected (liquid states) and interpreted in the output layer (readout). Therefore, the great attractiveness is the fact that is not necessary to calculate (learn) values for all the connections among neurons inside the liquid, but only for the external layer (readout learning) and it can be as simple as a linear regression. Some principles of LSM can be found *in vitro* neural networks [22] and parameters applied to the neuron model, connection probability and short-term plasticity are derived from studies based on the rat somatosensory cortex [23]. For that reason, the LSM could be seen as a possible model to implement a neural arm controller inspired by the results from Wessberg [20]. Any smooth dynamical system could be simulated by an LSM with the use of appropriate feedback [24]. Moreover, according to Hauser et al. [25], it is possible to emulate complex, non-linear computations with the use of a simple linear regression when Reservoir Computing, e.g. Liquid State Machine, is employed.

The use of an LSM to control a simple, planar, 2 degrees of freedom simulated arm was already implemented by Joshi and Maass [26]. However, the network was not exhaustively tested in order to verify its robustness or, for example, if the system was able to control more than 2 joints. In addition, their simulator [26] is not under active development, depends on the proprietary software Matlab® (CSIM) or make use of multiple libraries in

a C++ core (PCSIM) making it harder to deployed in newer systems. On top of that, a search in the literature could not provide any further works with implementations of robot arm controllers using the LSM framework.

The robustness of an LSM was presented before by Hazan [27], but instead of using a complex application they tested it in a simple classification task. Also, the LSM had no feedback loops connecting the readout output to the liquid input, limiting the system's memory and, therefore, its ability to process longer time series or to keep a consistent output.

Analog neuron based reservoir computational applications traditionally do not alter the network after the learning process is finished. Noise is applied during the initialization or training. Nevertheless, stochastic processes seem to be an important part of brain computational strategy [28] and the LSM technique implements noise levels compatible with what was found *in vivo* recordings [12].

Lewis and Klein [29] defend the idea that, even being universal approximators, traditional artificial networks end up being simple mapping devices and more complex models are necessary in order to be able to deal with the temporal behaviour of neurorobotic tasks. The work presented here aims to develop such dynamical models based on the spiking neural network approach.

1.2 Research Goals

This thesis is focused on the advancement of neurorobotics or brain inspired humanoid robot controllers based on artificial neural network architectures. Its main goal is to explore the possibility of using brain inspired neural networks to control a robot by demonstration. Moreover, it aims to achieve systems robust to environmental noise and internal structure destruction presenting a graceful degradation. As for validation, action learning experiments using the research version of the Rethink Robotics Inc. industrial humanoid robot Baxter are utilised.

Since the author of this thesis was part of the BABEL project[1][2], one extra constraint was that all the resultant solutions had to be capable of running on the neuromorphic architecture SpiNNaker [30]. In addition, SpiNNaker is mainly programmed by using computational models based on the spiking neuron framework; therefore, all the work developed here was created on top of the spiking neural model.

In order to reach the needs stated above, biologically inspired artificial neural networks are employed. This work is based on the idea that to address the problems in a novel and efficient way it is necessary to develop new computational models (on-line computation) inspired by the neural networks found in nature.

All the work presented in this thesis is based on an initial scientific hypothesis that Liquid State Machines, endowed with an extra external feedback loop, could be extended to control robotic arms with more than 2 degrees of freedom. More precisely, this work aims to test if such systems can learn by demonstration how to reproduce arm movements. Additionally, it is verified whether the use of parallel Liquid State Machine robot controllers brings any improvement in relation to the robustness of such controllers when compared to the simple use of multiple columns, and whether they present a graceful degradation when exposed to damage or harsh environments.

Summarising what was stated above, the most important results and benefits reached with the work presented here were:

- Generation of innovative new ways to control robots using a truly parallel approach instead of the classical serial paradigm.
- Development of control systems that are able to better deal with noise or even take advantage of it.
- Extension of the Liquid State Machine framework, using the learn by examples (action learning) methodology to control commercially available, real-world robotic arm models with more than 2 degrees of freedom.

[1] http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/J004561/1
[2] http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/J00457X/1

1.3 Thesis Structure

This thesis is divided into 8 chapters and 5 appendices created in order to address the research questions raised. All the chapters are presented in a chronological order reproducing how the work evolved during my PhD. Chapters 3 to 7 describe the experiments resulted from this PhD investigation, their results, and most of the contents have already been published as articles. In the appendices, all the articles related with this thesis are presented.

Chapter 2 presents the main background necessary to the development of this work as well the literature review about spiking neuron network based robot control (Neurorobotics). New tools developed during this PhD are also presented there.

A study verifying a new Python based implementation of a planar, 2 degrees-of-freedom (DOF) LSM arm controller is presented in the Chapter 3 and some demonstrations of the system's potential are done as well novel robustness experiments. This initial experimental chapter was necessary to verify and understand the current state-of-the-art in robot control using Liquid State Machines. The implementation and extension of the work initially presented by Joshi and Maass was very important because it set the scene to what such a system was capable of.

Chapter 4 introduces, according to the author's knowledge, the first 4 DOF LSM based robot arm controller and the results of its implementation using the humanoid robot Baxter. This controller was still based on the new Python implementation presented in Chapter 3; therefore, the number of possible trials was heavily limited to the low-speed interpreted scripting language.

While in the first chapters the robot controllers were capable of commanding the simulated robotic arm in straight lines, in the subsequent ones the straight lines are replaced by closed trajectories. In Chapter 5, some of the problems verified during the first implementation of LSM based arm controller from Chapter 4 were solved by the introduction of a novel framework using multiple LSMs together with a normalisation of the signals inspired by electronics instrumentation. Thanks to the new multi-threaded specialised simulator

implemented during this PhD[3], it was possible to increase the complexity of the experiments. Therefore, a total of three closed shapes (square, triangle and circle) were employed to compare and contrast the new framework against the state-of-the-art implementations based on LSMs.

One of the research questions raised during this PhD was related to the robustness of robot controllers based on Liquid State Machines. Natural neural systems are capable to degrade gracefully when subjected to damage. Therefore, the robot arm control system introduced in Chapter 5 is tested against noise and decimation of its internal nodes in Chapters 6 and 7, respectively.

The last chapter presents the final conclusions, a summary of the contributions to knowledge and suggestions of future works.

[3]https://github.com/ricardodeazambuja/Bee

Chapter 2

Background

2.1 Introduction

In the last chapter, the central motivations that led to the development of my PhD research were presented together with the research goals produced based on those arguments. However, a deep review of the state-of-the-art was not introduced and this will be one of the aims of this chapter.

This thesis was developed employing different experiments where, mostly, the arm of a simulated robot is directly taught, by a human teacher, to generate some specific endpoint trajectories. These experiments mainly consisted of artificial neural networks simulations, time series analysis and robotics.

The following sections will guide the reader through the necessary background to reproduce the work presented here. Relevant literature and basic theory for the experiments are detailed in the next sections; however, some aspects that are exclusively related to one specific experimental chapter are explained in the respective methods section in order to keep the line of thought.

2.2 Neurorobotics

The development of robots able to deal more effectively with real world challenges by the combination of the knowledge from Neuroscience, Robotics and Artificial Intelligence resulted in the formation of a new multidisciplinary research field known as Neurorobotics [10]. Kaplan [11] considers embodiment of neural systems as the most important aspect of Neurorobotics. Another definition can be found in the work of Lewis and Klein [29] where neurorobots are defined as robots controlled by the use of accurate biological neuron models. Neurorobotics suggests that, instead of algorithms, neural based high dimensional dynamical systems are implemented in this type of robots considering brains probably do not compute Jacobians in order to control limbs [29]. One of the seminal works was developed in 1988 with a neural network model for the generation of motor commands [31]. Seth et al. [32] stated that the implementation of neurobotic systems could result in the development of more effective robots. Inspired by the work of Krichmar [33], in this thesis a *neurorobot* is defined by two criteria:

1. Action Learning
    - Participates in a behavioural task.
    - Interacts actively with a real-world scenario.
    - Uses its embodiment to perceive and react to the environment.
2. Bio-inspired neural system
    - Behaves as a result of an artificial bio-inspired neural system.

Action learning is a participative learning model where the acquired knowledge comes from the experience or the interaction with the environment [34]. Therefore, embodiment (the "organismoid" embodiment as defined by Ziemke [35]) is an important part of the action learning framework. According to Cangelosi et al. [6], since the early years children already make use of the relationship between body and objects to make sense of the world around them.

Nowadays, most robots need to be programmed in order to accomplish a task. Every time the task changes, the robot must be reprogrammed. Humanoid robots, easily presenting more than 6 degrees of freedom (a common industrial robot standard), become a bigger challenge to users when this traditional approach to set up tasks is used. Using one variation of action learning known as imitation learning it is possible to overcome this problem. This, alone, is not a new idea as the importance of imitation learning for humanoid robots control was explored before [36, 37, 38].

One simple application of imitation learning can be seen as forced teaching techniques more commonly known in the machine learning field as supervised learning. Haykin [39] defines supervised learning as the modification of the synaptic weights in response to samples or examples showed to the neural network during the training period.

In the current work, neurobotic systems are implemented by the use of Spiking Neural Networks (SNN) as they are a more realistic bio-inspired version of the traditional non-spiking connectionist models (Section 2.2). This idea comes also in consonance with what is presented by Maass [40] since a SNN have at least the same computational power as multilayer perceptrons and sigmoidal neural networks.

2.2.1 Robotics and Artificial Spiking Neural Networks

Currently, neural networks are employed to robotics in a wide range of applications that spread among several fields, making almost impossible to list the current state-of-the-art in general terms. Ablameyko [41] presents a historical review where neural networks were developed initially as brain models for robotic systems or *experimental systems*. Those first brain models (during the Fifties and Sixties) used terms as "sensory units" (instead of the modern "input units"), "associative units" ("hidden units") and "response units" ("output units"). In that scenario, the inputs came from sensors and output went to actuators. The same work lists early robotic applications as Road Driving Vehicle Controller, Robotic Inspection of Aircraft Skin and Robot Models for Motion Planning. However, in the review presented here, the applications were restricted to Neurorobotics using the biologically

inspired spiking neuron model, as they have a closer relation to the work developed in this thesis.

Starting in a chronological order, and initially skipping the applications with humanoid robots to be reviewed in the Section 2.2.2, most of the works found in the literature are related to small mobile robots. The mobile robot Khepera was employed as the embodiment platform for the SNN in many of those works [37, 42, 43, 44].

A work from 2004 [45] already applied genetic algorithms controlling the weights in a SNN to evolve a mobile robot controller. Burgsteiner [37] implemented an imitation learning scheme using LSM and the mobile robot Khepera to move avoiding obstacles. Spike-timing-dependent plasticity (STDP) and an one-hidden-neuron SNN were also used to create a sensor-fusion system to control a mobile Khepera robot [42]. The mobile robot CASIA-I controlled by a SNN using Hebbian learning and the ultrasonic sensory inputs was presented by Wang et al. [46]. Six years after the publication of Hagras et al. [45], a simulated Khepera robot was used in a brain inspired SNN architecture with the help of a working memory [43]. Insect inspired SNN prey localisation system (angle and distance) was proposed by Adams et al. [47]. An imitation learning approach using a mobile robot controlled by SNN where the learning comes from evolutionary algorithms was presented by Batllori et al. [48]. Behaviour experiments in a simulated Khepera robot using SNN as brain-controller and STDP was proposed by Cyr and Boukadoum [44].

A different solution to the robot control problem was introduced by the authors by Jimenez-Fernandez et al. [49] using a SNN based PID controller tested in a DC motor using an AER-ROBOT board. Lego Mindstorms NXT robotic kit in an autonomous interaction experiment using an evolutionary adaptation of SNN has also been proposed [50]. The Arduino based mobile DFRobotShop Rover robot communicating through Wi-Fi with a host PC where the SNN simulator executing a reward based Hebbian learning system was presented by Helgadottir et al. [51].

As a last example of robot control using SNN, Markowska-Kacszmar and Koldowski [52] controlled not a robot, but a computer car racing game that could be seen as a simulated robot, trained by an evolutionary algorithm.

2.2.2 Robot Arm Control using Artificial Spiking Neural Networks

Industrial robots have been used since the Unimate's release in 1961 [53]. These robots are very efficient, but normally only for specialized tasks. They also need a special environment because of their dimensions and weight (one cannot simply fit an industrial robot in front a steering wheel of a car without lots of adaptations). Humanoid robots have the potential to be deployed directly replacing and/or assisting the human being in monotonous or dangerous tasks. The idea of a human like robot sounds perfect, but the complexity of such a robot makes the controlling and programming very hard. Just to give the reader an idea about this complexity, the humanoid robot iCub [7] has a total of 53 degrees of freedom (DOF). A single arm of a commercial humanoid robot like Baxter (from Rethink Robotics Inc.) has 7 DOF without counting the tool (hand). The number of DOF depends on the workspace the manipulator is supposed to work and interact in. In order to be positioned (location and orientation) in the 3D space an arm would need only six controllable DOF. This situation is known as holonomic. The human arm is a redundant system (more controllable DOF than necessary) and, in technical terms, this means that the inverse kinematics does not have an unique solution. When the task level is considered, the control system has an even greater challenge as behaviour goals have an infinitude of possible ways they can be executed [54]. Biomechanical joints were studied by the Russian physiologist Nikolai Bernstein already in 1930 [55] what resulted in the famous expression "degrees of freedom problem".

In the literature, it is possible to find a good amount of works where movements are associated with neural activity *in vivo* [9, 56, 57, 58, 59, 60, 61, 62]. One thing all those works have in common is the spiking nature of the biological neuron.

In contrast, if the literature is searched for robot arm control using artificial SNN, instead of several works as mentioned above for the case of neural activity associated with movements, it is only possible to find just a few. Maybe the reason for this lack of SNN arm controllers is that it is difficult to implement such systems - naturally parallel and event-driven - using serial programming style. With the advent of neuromorphics chips like SpiNNaker [30],

Neurogrid [63], Spikey [64], ROLLS [65], TrueNorth (IBM) and Zeroth (Qualcomm) this could start changing.

One of the first successful applications of SNN to the control of a robotic arm was made by Joshi and Maass [26] where two different simulated 2 DOF planar arms were controlled to follow four endpoint trajectories using an LSM made of Leaky Integrate and Fire (LIF, for more details see Section 2.3.2.2) neurons. Gamez et al. [17] used a simulated version of the muscle actuated CRONOS robot, called SIMNOS, with its arm controlled by a SNN, but the work focuses more on the simulators and interfaces than in the arm control.

A 2 DOF robot arm controlled by a SNN for the task of goal location was presented by Rowcliffe and Feng [66] and, as in the work of Joshi and Maass [26], straight line trajectories with Gaussian velocity profiles were used, but instead of four trajectories (goals) the system could deal with only two.

Carrillo et al. [67] presented an even more biologically inspired model, based on the cerebellum and using a SNN and STDP, for learning acts as a predictive corrective module for one joint of a 2 DOF compliant arm application.

The use of a firing rate based approach together with STDP (a total of 240 neurons) learning a 2D coordinate transformation of the polar representation of an arm position to a Cartesian representation was presented by Wu et al. [68]. They claimed the network was able to perform the transformation between 2D Cartesian to polar coordinates.

While the previous works were able to control only a 2 DOF arm, a SNN controller able to deal with one of the iCub's arms (using only 4 DOF) was presented by Bouganis and Shanahan [16]. A total of 12.000 Izhikevich neurons [69] were used, together with STDP and motor babbling (motor learning process based on repeating random exploratory movements) for the training scheme. According to the data presented in the paper, the final controller was not able to follow trajectories, but move approximately in the direction according to the Cartesian goal position. There was no mention of the error at the end of the movement.

A 1 DOF simulated robot arm controlled by a total of 144 excitatory and 64 inhibitory neurons using reinforcement learning was introduced by Chadderdon et al. [70]. This simple arm was driven by a pair of extensor and flexor artificial muscles and it was able to follow simple step functions.

A self-organizing network using STDP was employed to learn trajectories (2 DOF planar simulated arm) according to Srinivasa and Cho [71]. The network was composed of $7,000$ neurons and $1.2M$ STDP synapses taking $1,500s$ to reach an acceptable error level. The prediction of the direction during the generation of the trajectories had a better performance than during the point-by-point trajectory generation.

A biomimetic model was trained (four different trajectories [72]) using spike-timing dependent reinforcement learning to drive a simple kinematic two-joint virtual arm in a motor task requiring convergence on a single target. Their SNN had 384 excitatory and 128 inhibitory neurons. The virtual musculoskeletal arm was the interface between the SNN and the Whole Arm Manipulator to control 2 of its 7 DOF.

Neymotin et al. [73] extended to 2 DOF the work of Chadderdon et al. [70] with arm positions calculated at a frequency of 20Hz and 704 neurons connected probabilistically. The system was able to reach five target positions (training set) from arbitrary starting ones (no trajectory learning).

A versatile spiking cerebellar simulated network [74] was able to compensate perturbations in a PD controller. The 2 DOF arm was subjected to an external force field and the simulated cerebellum had to actuate to compensate. A low-pass filter was also used in order to convert from spikes to torque (firing rate from 20 Deep Cerebellar Nuclei Cells).

A discrete analog hardware implementation of a spiking neural network was employed for the control of a 1 DOF robotic arm. The limb was actuated by artificial muscles that were implemented with *Flexinol*® (shape memory alloy actuator wires[1]). A lamprey motor neural network was the inspiration for the implementation. Having the feedback from an encoder, the system was able to control the joint through a range of about 50 degrees [75].

[1] http://www.dynalloy.com/flexinol.php

The Neural Engineering Framework [76] in conjunction with Neurogrid [63] was applied to the generation of a set of functions that together constituted a 3 DOF arm controller [77]. This type of implementation resembles a readout in an LSM [23] (where the neuron pools are the liquids) but they fit individual functions instead of the whole controller. Then those functions are combined to generate the controller (which they refer to as control decompositions). One important claim in the paper was being the first time a neuromorphic system was able to control a 3 DOF arm in real-time.

An inverse kinematic system based on anti-Hebbian-Hebbian learning (AHaH) is used in the implemented arm controller of Nugent and Molter [14]. They were able to train a SNN to control a planar arm to reach moving and static targets. The simulated robot arm had its joints actuated by opposing muscles composed of several fibres. Arms from 3 up to 21 joints were used. The system could not follow trajectories and the performance was measured against a random controller by counting how many steps (or wrong positioning) it took until the target was reached.

2.2.3 Conclusions

The work presented in this thesis started motivated by the neurorobotics ideas. Therefore, it was essential to use a bio-inspired system. Spiking Neural Networks (SNN) are a more realistic, bio-inspired, version of the traditional non-spiking connectionist models and, according to Maass [24], SNN have at least the same computational power as multilayer perceptrons and sigmoidal neural networks.

Using the Liquid State Machine (LSM) framework, Joshi and Maass [26] were capable to produce of the first successful applications of SNN to the control of a robotic arm. However, it was a simple 2 DOF planar arm and they did not show real-time capabilities. Most of the available works using SNN for robotic control are restricted to simple arm models. Bouganis and Shanahan [16] developed a controller for 4 DOF, but the data presented in the paper shows a final controller not able to follow trajectories and with no mention about the error at the end of the movements.

Probably, the architecture most similar to LSMs is the Neural Engineering Framework (NEF) [76]. Menon et al. claimed [77] it was the first time a neuromorphic system was able to control a 3 DOF arm in real-time. However, NEF lacks connections inside its neuro pools (ensembles) and recurrency is mostly done only through processed outputs. In a possible physical analogue implementation, there are no studies showing NEF would not be as robust as LSM in relation to cross talking (LSM by design has random connections among neurons inside the liquid).

Finally, Nugent and Molter [14] presented a system that was capable to train simulated planar arms with up to 21 joints. Their final controller could reach moving and static targets, but nothing was done in relation to following trajectories. Besides, the source code was not available under standard open source license.

2.3 Methods and Materials

2.3.1 Artificial Neural Networks

The human brain can be seen as a complex, non-linear and parallel computational system. All this computational power is supposed to come from the dense networks formed by the brain's neurons. The connection point between neurons is known as a synapse. Artificial neural networks (ANNs) are implementations (software or hardware based) inspired by the the discoveries made throughout the last hundred years about the brain. A neural network is defined by Haykin [39] as a massively parallel distributed processor. This characteristic, together with its adaptive - or learning - power, make the neural networks to be inherently fault tolerant since every neuron contributes with only a small part of the computation (massively parallel system) and even if bigger areas are compromised the network could change its weights in order to self correct the disturbances (learning or adaptation).

In the work of Maass [40], neural networks are classified into three different generations: first, second and third. The first generation is composed by binary gates widely known as perceptrons or McCulloch-Pitts neurons. They are considered universal for digital computations and every boolean function can be implemented using the proper network structure.

Nowadays, some of the fashionable deep architectures are implemented using this generation through the use of Boltzmann machines [78]. More complex neuron models using continuous activation functions define the second generation. The continuous behaviour of the implemented neuron gave rise to the discovery of the revolutionary back-propagation training method as gradient descent demands a continuously differentiable function. Compared to the first generation, second generation neural nets are able to compute functions where the input and output are analogue values.

As real neurons send and receive information using pulses (spikes), the second generation of artificial neural networks uses the biological interpretation of continuous activation as a firing rate (number of spikes produced in a given time window). This type of code becomes problematic when fast computations are considered since firing rates are mean values and in a very short time window a meaningful rate cannot be defined [79].

The third generation of neural networks employs spiking neurons (e.g. integrate and fire neurons) as the basic unit. Maass [40] states that the third generation is computationally more powerful than the first two generations (when the necessary number of neurons to a certain task is considered) and, on top of that, the spiking neuron has, in the worst scenario, the same computational power. Also, according to Adams et al. [80], there are evidences from neurobiology suggesting spikes are important for cognitive and behavioural tasks.

2.3.2 Spiking Neuron

The classical work about spiking neuron models published by Gerstner and Kistler [13] explains the ideal spiking neuron as divided into three parts: dendrites, soma and axon. The dendrites are the input gates, soma the equivalent to the central processing unit and axon the output interface. In the literature, the neuron sending a pulse is defined as the pre-synaptic neuron and the receiving neuron is the post-synaptic neuron.

2.3.2.1 Rate versus Time based coding

Experimental results suggest a simple firing rate code, as used in the second generating of neural networks, is not capable to explain some of the fast processing seen in the brain as when a human responds to visual scenes - that demand several processing steps - spending only about $400ms$ [13]. Moreover, according to Buonomano and Maass [12], temporal information is very important when natural stimuli are being processed - the ability to compress language in Morse code is one given example. Time based codes also made possible the development of neuron adaptability theories as the STDP (Spike-timing dependent plasticity, a variation of Hebbian learning implementation - see [81] for a detailed review) and STP (Short-Term Plasticity).

2.3.2.2 Leaky Integrate and Fire

Several spiking neuron models have been proposed in the literature. A comparative study in relation to biological plausibility and computational efficiency was presented by Izhikevich [69]. As the aim of the current work is to analyse higher-level functions, the systems developed only make use of Leaky Integrate and Fire neurons (LIF).

The basic LIF model behaves as a capacitor-resistor circuit with an added circuitry in order to generate the spike (action potential) and also keep it discharged during the refractory period [13]. Perhaps, the simplest way to define a LIF neuron was given by Lewis and Klein [29] and it is "a capacitor with a decision making capability".

$$\frac{dv}{dt} = \frac{i_e(t) + i_i(t) + i_{offset} + i_{noise}}{c_m} + \frac{v_{rest} - v}{\tau_m} \tag{2.1a}$$

$$\frac{di_e}{dt} = -\frac{i_e}{\tau_{syn_e}} \tag{2.1b}$$

$$\frac{di_i}{dt} = -\frac{i_i}{\tau_{syn_i}} \tag{2.1c}$$

A variant of the LIF model with exponential currents is used in this thesis. It can be partially represented by the set of differential equations as seen in the Equation (2.1) where c_m is the membrane capacitance (in F), τ_m the membrane time constant (in s), τ_{syn_e} and τ_{syn_i} decay time of the excitatory and inhibitory synaptic current respectively (in s), v_{rest} the membrane resting potential (in V), i_{offset} a fixed noisy current and i_{noise} a variable noisy current (in A).

However, the way LIF neuron models are implemented requires more than differential equations. As detailed explained in [13], the model needs comparators to generate the spikes and also timers to control the refractory periods.

Moreover, the LIF model implemented in this thesis is initialised with random values, drawn from uniform distributions, for its membrane initial voltage, membrane reset voltage, offset currents (i_{offset}) and it receives a Gaussian white noise, in the form of an injected current, during every time step. All the values used in most of the experiments presented in this thesis can be found on Table 2.1.

TABLE 2.1: Liquid State Machine default parameters used in this thesis.

Parameter	Value	Unit
Membrane time constant (τ_m)	30.0	ms
Membrane capacitance (C_m)	30.0	nF
Synapse time constant (exc. - τ_{syn_e})	3.0	ms
Synapse time constant (inh. - τ_{syn_i})	6.0	ms
Refractory period (exc.)	3.0	ms
Refractory period (inh.)	2.0	ms
Membrane Threshold	15.0	mV
Membrane Reset	[13.8, 14.5]	mV
Membrane Initial	[13.5, 14.9]	mV
i_{offset}	[13.5, 14.5]	nA
i_{noise} (μ)	0.0	nA
i_{noise} (σ)	1.0	nA
Transmission delay (exc.)	1.5	ms
Transmission delay (inh.)	0.8	ms

2.3.2.3 Short-Term Plasticity

Short-term plasticity (STP) is the dynamical change in the synaptic efficacy over time [82]. It can also be compared to dynamical memory buffers [83]. Also, according to Rotman

et al. [84], STP increases the transmission of information in the frequency range of the naturally occurring spikes.

In this thesis, the STP implementation was based on the example from the Brian Simulator (see Section 2.3.6.1) documentation (version 1.4) [85, 86]. The Brian's *Synapses* method was used with the parameters from Listing 2.1 in order to simulate the synaptic dynamics described by Markram et al. [85]. In addition to the implementation described here, the Brian Simulator can also implement the Short-term plasticity using its STP class.

```
model='''x : 1
         u : 1
         w : 1
         tauf : 1
         taud : 1
         U : 1'''
pre='''u=U+(u-U)*exp(-(t-lastupdate)/tauf)
      x=1+(x-1)*exp(-(t-lastupdate)/taud)
      i+=w*u*x
      x*=(1-u)
      u+=U*(1-u)'''
```

LISTING 2.1: Brian's Synapse parameters definition for Short-term plasticity modelling.

The *Synapses* method, using the parameters from Listing 2.1, implements the simple phenomenological model [85] presented in Equation 2.2. Where u is the utilization (release probability), τ_f is the time constant for facilitation, τ_d is the time constant for depression, τ_s is the synapse time constant, x is the normalized variable and t_{sp} is the spike time.

$$\frac{du}{dt} = -\frac{u}{\tau_f} + U(1-u^-)\delta(t-t_{sp}) \tag{2.2a}$$

$$\frac{dx}{dt} = \frac{1-x}{\tau_d} - u^+x^-\delta(t-t_{sp}) \tag{2.2b}$$

$$\frac{die}{dt} = -\frac{ie}{\tau_s} + wu^+x^-\delta(t-t_{sp}) \tag{2.2c}$$

TABLE 2.2: Short-term plasticity default parameters.

Parameter	Value				Unit
	EE	EI	IE	II	
Scaling (A)	70.0	150.0	-47.0	-47.0	nA
τ depression (D)	1.1	0.125	0.7	0.144	ms
τ facilitation (F)	0.05	1.2	0.02	0.06	ms
Use (U)	0.5	0.05	0.25	0.32	-

The parameters for the synaptic dynamics are from Joshi and Maass [26]. For STP based simulations presented on this thesis, all parameters were drawn from a Gaussian distribution whose mean values (μ) are reproduced in Table 2.2 and the standard deviation is 50% of $|\mu|$. According to the type of connection (EE, EI, IE and II, where E and I stand for excitatory and inhibitory neuron, respectively) a different value was used. The parameters presented in Table 2.2 can be found in Listing 2.1 and Equation 2.2 by simply replacing: A by w, D by τ_d and F by τ_f.

2.3.3 Liquid State Machines

The Liquid State Machine (LSM), first proposed by Maass et al. [21], is a neural computation framework based on Spiking Neural Networks (SNNs). One or more input signals are injected in a non-linear dynamical system, a group of artificial neurons modelling a column in the cortex, and the disturbances generated are collected (liquid states) and interpreted in the output (readout). A simple diagram representing an LSM is presented in Figure 2.1.

The basic idea of the LSM comes from Support Vector Machines (SVM) or Support Vector Networks [87]. In the SVM, data is mapped into a space of higher dimension in order to increase the chance to make it separable (more details can be found in [88]).

Maass et al. [21] also demonstrated the universal computational power of the LSM model when some idealized conditions, separation and approximation properties, are met. As a better way to clarify its basic principles, a pond is often used as a metaphor (the liquid) because, as pebbles are thrown on the water surface, waves (disturbances) appear and interact with each other. After some time without activity in the liquid (e.g. they ran out of pebbles) the drawings formed on the surface by the waves edges slowly disappear. The

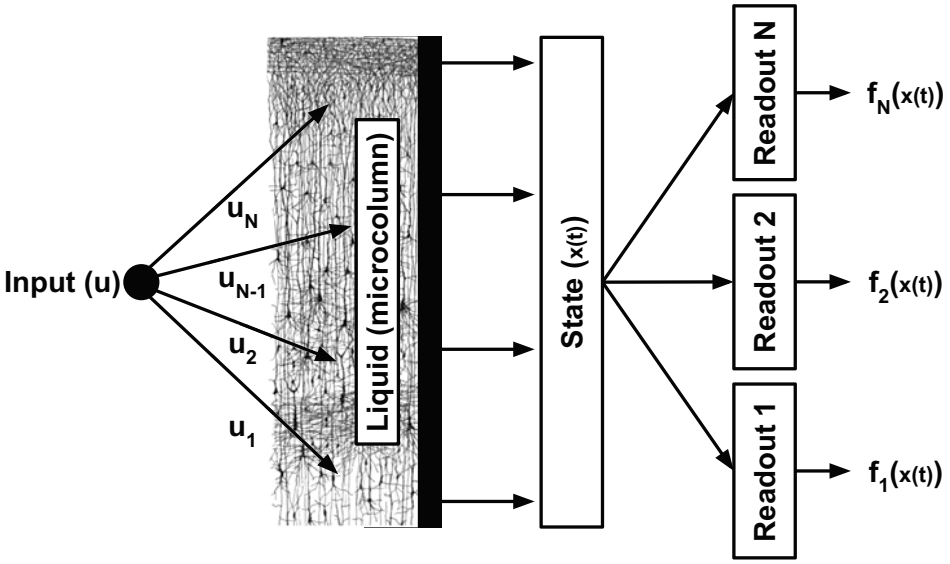


FIGURE 2.1: Liquid State Machine Structure. One or several inputs are injected into the liquid as spikes or currents over time. The reaction of the liquid to those input time series generates a state vector ($\boldsymbol{x}(t)$) composed of what results from low-pass filtering the liquid's output spikes. Multiple readout layers can be trained for classification or regression ($f(x(t))$). Liquid background image adapted from Santiago Ramon y Cajal public domain drawing.

group formed by all the elements interacting with each other can be seen as a non-linear dynamical system with only one attractor state - when the waves totally fade out.

The LSM has in its perturbed states all the information representing the current and past inputs (in theory, the fading memory never reaches the zero - empty, but in practical terms the past inputs last a finite time) and this makes it an interesting tool to process time-series data [89, 90, 91, 92, 93]. All the computations being accomplished by the system are easily accessed using, for example, a linear classifier on the readout neurons such as a least-squares linear regression or even a perceptron. Schurmann et al. [94] justify the use of such a simple classifiers stating that the high-dimensional space created by the liquid increases the probability of having a linearly separable classification problem at the end.

An LSM usually is composed of Leaky Integrate-and-Fire (LIF) neurons (see Section 2.3.2.2), connected in a recurrent pattern [21, 95], forming what is known as Small-World Network (SWN) [96]. The probability of creation for each connection depends on the distance between neurons and it is calculated by Equation 2.3. An example of generated network that follows this model can be seen in Figure 2.2.

$$P_{conn} = Ce^{-\frac{D(a,b)^2}{\lambda^2}} \tag{2.3}$$

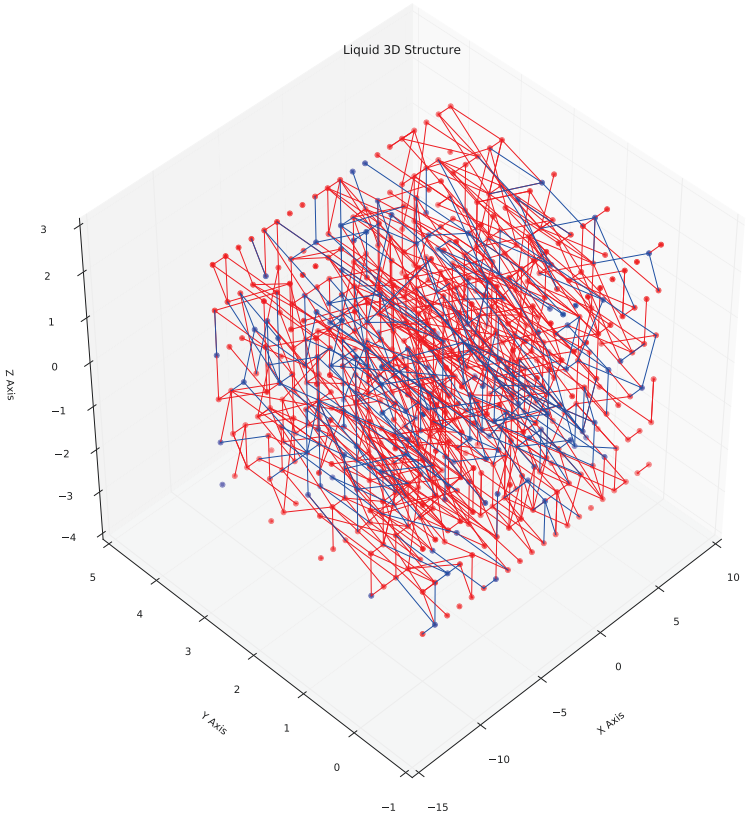


FIGURE 2.2: Visualization of the liquid's shape (showing only the connections between two different neurons). Red dots indicate the excitatory neurons while blue ones the inhibitory. Lines coloured in red are excitatory connections and blue inhibitory. Self connections are not displayed. The parameters used to generate the liquid's structure were the same as presented by Joshi and Maass [26].

Bassett and Bullmore [97] suggest this type of network presents an appealing way to model the brain connections based on empirical and theoretical motivations. Compared to models where an All-to-All connection methodology is used, SWN can make use of far less wiring yet able to generate high dynamical interactions through many indirect feedback loops when a properly chosen probability connection is used. According to Watts and Strogatz [96], SWN combines high clustering ability with a short path length. Generating a connection density plot of the network presented in Figure 2.2, it is possible to see the formation of clusters where the connection density is much bigger than in the rest of the network (see Figure 2.3).

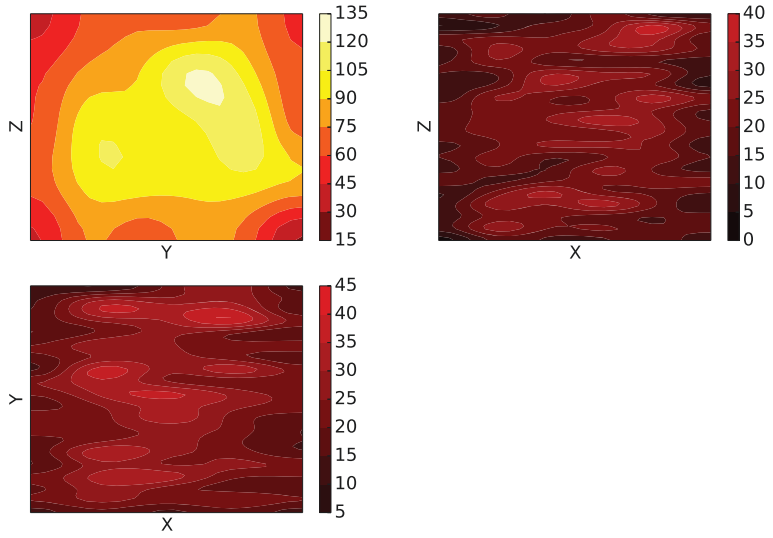


FIGURE 2.3: Density plots of the total number of connections (summed in the axis perpendicular to the plane) according to one of the 3D planes (XY, XZ and YZ) calculated from the liquid showed in Figure 2.2. The colour scales are the same to all three images. The images show that connections are concentrated in the inner part of the liquid (brighter areas).

According to Maass et al. [23], the parameters applied to the neuron model, connection probability and Short-Term Plasticity (STP) all came from real measurements of micro-circuits in rat somatosensory cortex.

In addition to the biologically plausible neuron model and network parameters, this approach implements noise levels that account for what is found *in vivo* recordings [12] and it is known that stochastic processes are very important for brain functioning [28] making the LSM not only a biologically plausible model, but a physically plausible one because it is impossible to avoid noise in the real world since several noise sources are actuating inside the brain [98]. Moreover, in the work of Dockendorf et al. [22] it was proved that some principles of LSM could be found within in vitro neural networks of cortical neurons making clear it's not only computationally useful, but could help to explain how biological neural systems work.

Furthermore, the whole idea of the LSM can be extended to applications that are not directly related to neural systems. Fernando and Sojakka [99] applied the LSM principles, in a very literal way following the pond metaphor, to pattern recognition using some

motorized toy mounting pieces, a camera and a transparent tank filled with water. Even the possibility of using bacteria as a liquid has been already considered [100]. Emotions were recognized from human faces [101] and speech processed [102] both using LSM.

Since the basic theoretical idea behind the LSM was also developed concurrently as Echo State Networks [103], Decorrelation-Back-propagation Learning [104] and previously as Temporal Recurrent Neural Network [105] all those ideas were later subsumed as *Reservoir Computing*.

The LSM, as initially defined [21], was considered to have a main limitation related to the extension of its fading memory and only computations requiring integration in a time range of about $300ms$ (this value changes according to the parameters used) were possible. In summary, the LSM was not able to be applied to tasks with long duration. However, Maass et al. [24] proved that, with appropriate feedback, the LSM becomes capable of emulating any dynamical system, conceivable analogue computer or arbitrary Turing machines. In addition, such systems can still perform in the presence of realistic noise levels, but they have the computational power reduced.

2.3.4 Baxter Humanoid Robot

The Baxter Robot is a humanoid robot produced by Rethink Robotics Inc. aimed to be a safe by design collaborative robot. The robot is composed of two arms with 7 DOF each with the ability to attach different end effectors as grippers or suction pads increasing even more the number of degrees-of-freedom. At the end of each arm are located distance sensors and a camera. Another camera is attached to a screen that functions as Baxter's interactive face. It is also an affordable alternative (according to Rethink Robotics website, its base price is 25,000 dollars) to traditional industrial robots, but capable to work next to human beings without the need of special environments or protective cages. Baxter research version uses the Robot Operating System (ROS) [106] to interact with the external world receiving commands and sending information. Also the robot was designed to be intrinsically safe with compliant spring based arm joints known as Series Elastic Actuators (SEA) [107] and software-based safety mechanisms that can detect self collisions as well

sudden changes in the joint torques protecting the users. Nevertheless, all the safety procedures that depend on software can only be activated while the position mode is used making the use of direct torque control almost impracticable as the whole idea of the robot is to be safe to work among human beings. In this thesis, joint angles are sent to the robot in order to control it, therefore without losing its safety aspects.

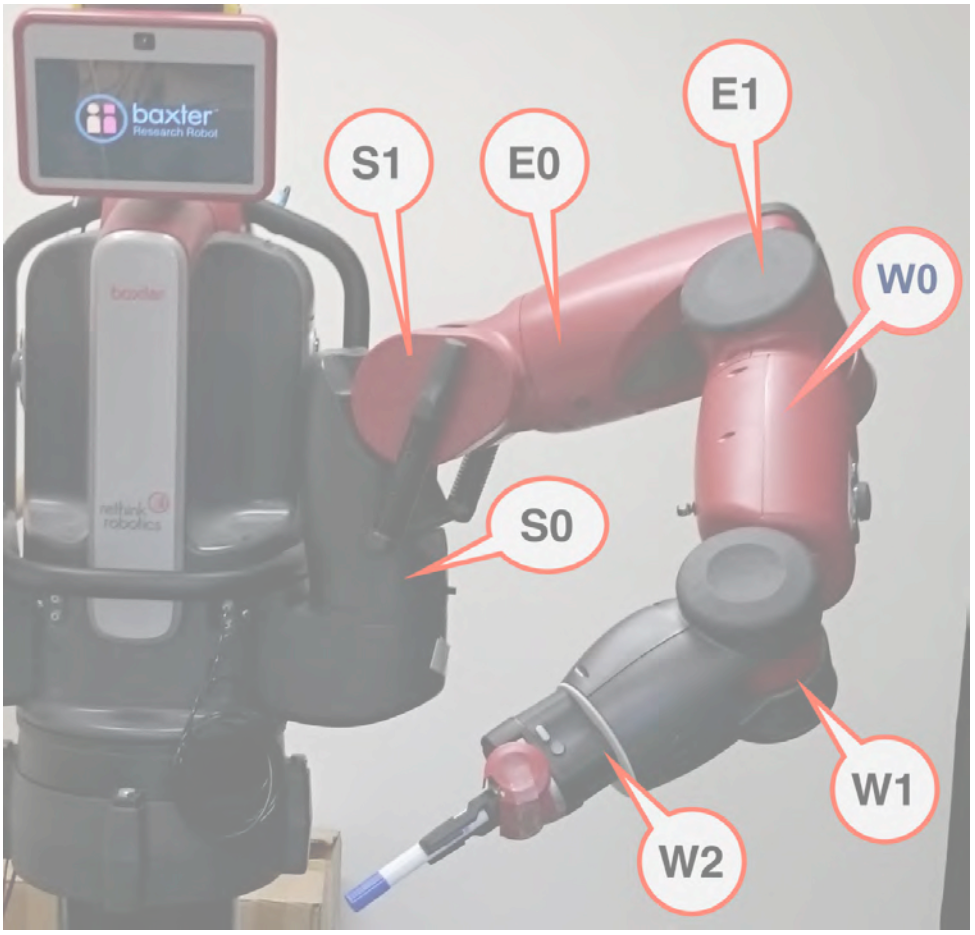


FIGURE 2.4: Picture of Baxter's arm at University of Plymouth robotics lab with indications of its joint names for Shoulder(S1, S2), Elbow(E1, E2) and Wrist(W1, W2 and W3).

2.3.5 Virtual Robot Experimentation Platform - V-REP

The Virtual Robot Experimentation Platform (V-REP) [108] is a cross-platform portable simulator that runs on Linux, Windows and OS X based computers. It works as a modular system where it is possible to develop simulations on up to three physics engine (Bullet Physics, ODE and Vortex Dynamics) as well doing inverse / forward kinematics, collision

detection, distance calculations or interacting with dynamic particle as air or water jets. Its remote API enables the simulator to be controlled by C/C++, Lua, Java, Matlab/Octave and Python. In most of the simulations presented on this thesis, Python [109] was used, together with the remote API, to interact with V-REP controlling a virtual version of the Baxter Robot. In the Figure 2.5 it is possible to visualize one the experiments where the robot draws a line using a simulated felt pen. A full version of V-REP for educational use is available and it was the one employed in this work.

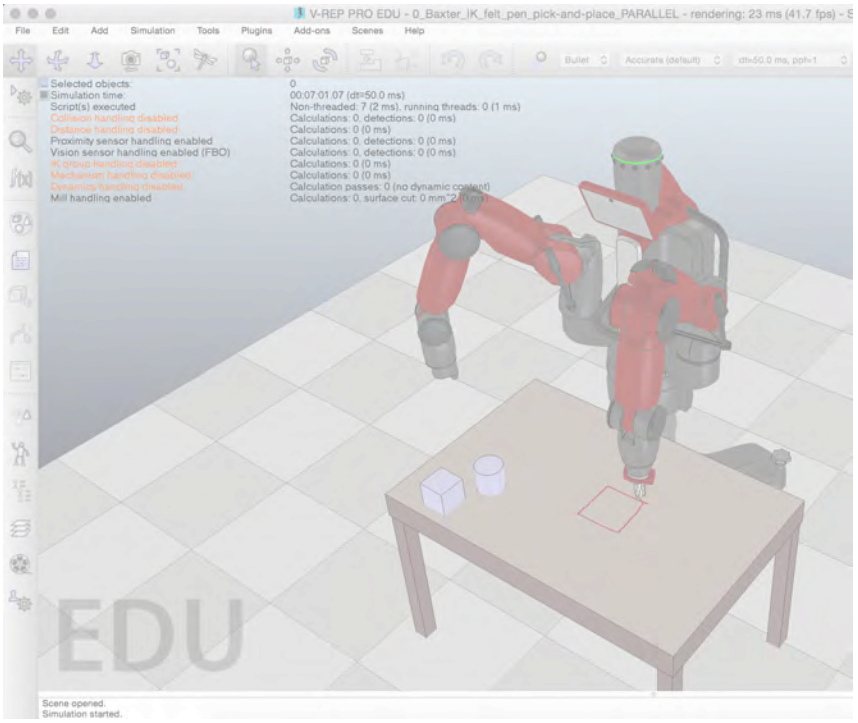


FIGURE 2.5: V-REP setup used in this thesis. A simulated felt pen was employed to draw the shapes. The shape drawn on top of the table corresponds to the square generated by the novel system presented in Chapter 5.

2.3.6 Spiking Neural Network Simulators

The experiments presented in this thesis were developed using mainly two SNN simulators: Brian and Bee. Initially, Brian was employed for the simulations (Chapters 3 and 4) because it is fully based on Python and easily portable to different systems. The simulations presented in this thesis were executed in two different systems running Apple OS X Yosemite (v.10.10.5) and Ubuntu (v.14.04.5), so portability was a big constraint.

However, with new experiments (Chapters 5, 6 and 7) demanding a bigger number of trials, and also the experiments using the real BAXTER had to run simulations with a delay compatible to the physical behaviour of the robot, the drawbacks of a system written in Python as Brian started creating problems. To solve those problems and also in order to let the author experience and learn about the internal details of a SNN simulator, it was decided to write a specialised SNN for LSMs. This effort resulted in Bee (see Section 2.3.6.3), a multiple thread, C based and Python wrapped open source simulator.

2.3.6.1 Brian Simulator

Brian [110] is a general purpose SNN system well known by the neuroscience community. It is freely available, open source and based on Python running on several different operating systems.

As Brian, at least until the time the work presented in this thesis was developed, didn't have a module specialized for Liquid State Machines, in this work it was also developed all the necessary infrastructure to automatically generate objects, using the methods available in Brian, in order to simulate the neural network according to what was presented by Maass et al. [21].

2.3.6.2 Brian Step-by-Step extension

In an online robot control task, it is mandatory to have a system that can deal with spike trains created on demand. Although Brian is a great software when one needs to simulate off-line systems, the version used here (V1.4) cannot easily receive and output a continuous spike train that uses values generated after the simulation had started. To solve this problem an extension was created to the Brian simulator making it possible to simulate online systems sending and receiving spike trains.

The extension facilitates the injection of very long spike train in a more efficient way. At every simulation step, input spikes were sent and output ones received preserving the current simulation state. All the variables used inside the simulation became encapsulated

into the extension decreasing the possibility of bugs. Additionally the extension properly reinitializes the simulator even when it is used inside the IPython [111] environment and the start-up of the Brian Simulator is performed only once every time the extension is initialized, saving time.

2.3.6.3 Bee - A Liquid State Machine Simulator

The Bee simulator is an open source, freely available online on Github [112], SNN specialised LSM system with its core functions implemented in C. It was developed together with this thesis exclusively to solve the specific problems presented here (Neurorobotics experiments). Bee uses the C library *pthreads* (POSIX threads) in order to speed up the simulation of LSMs by processing input and output spikes in a parallel way. A Python wrapper is supplied to simplify the user interaction with the software. The neuron model is hardcoded (fixed), following what was presented in Section 2.3.2.2, and the solution for the differential equations is calculated by the Euler's method according to the simulation's time step specified by the user.

The simulator has the ability to automatically generate the reservoir (liquid) according to Equation 2.3. All the parameters for the neuron model or the internal connections can be defined by the user. Also, motivated by the results presented in Chapter 3, both STP and time delays were not implemented in order to simplify and optimise the simulator since it would demand extra memory and increase the amount of calculations per simulation time step. In its current version, it supports, at least, Linux and OS X (it was never tested by the author on any version of Windows).

2.3.7 Dynamic Time Warping

The experiments presented in this thesis were focused on teaching a LSM based robot arm controller to draw using principles of action learning [34] and embodiment [6, 25]. In many situations the resultant movements had a constant value zone, a delay in time, that was not visible in the final drawings. Those time delays make it harder to apply traditional

metrics such as a simple Euclidean distance because two final shapes that look exactly the same could generate different metrics.

In order to solve this problem, from Chapter 5, data analysis was carried out applying the Dynamic Time Warping (DTW) [113, 114]. DTW is a time-normalisation algorithm initially designed to eliminate timing differences between two speech patterns. This normalisation, or correction, is done by warping the time axis of one time series to match the other. The correction (time warping) makes it easier to compare two signals in a similar way to the method human beings use [115].

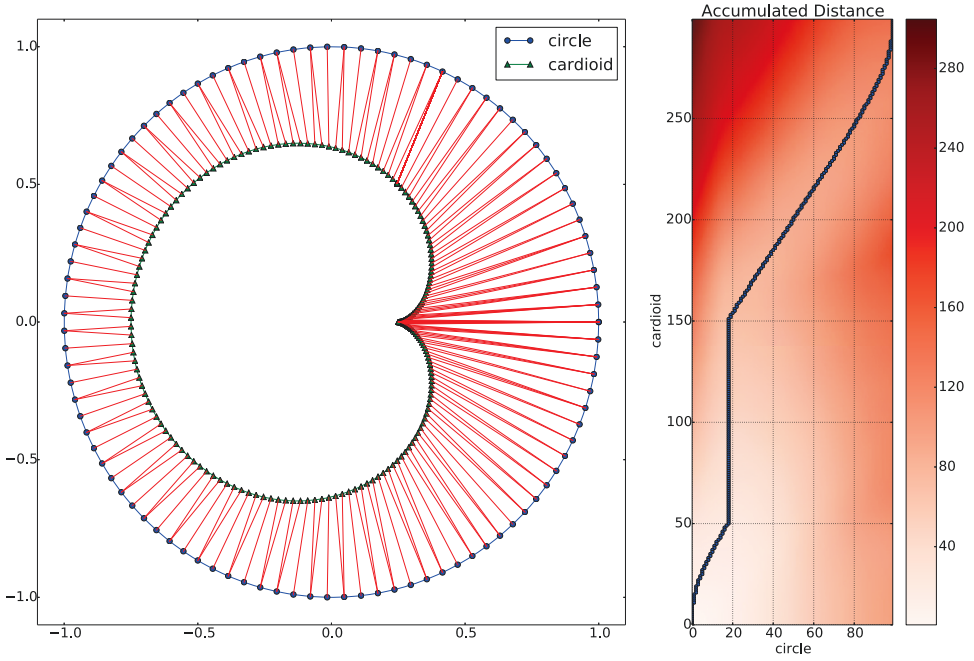


FIGURE 2.6: Example of 2D trajectory-matching generated by the DTW method. Although looking perfect in the figure on the left, the cardioid was modified to have a constant value zone from time step 50 to 150. The DTW correctly matches the values as can be seen as a straight blue line in the Accumulated Distance plot (right).

In Figure 2.6, it is presented an example where a time series that generates a cardioid is compared to a circle. The cardioid also had a time delay inserted (where values were kept constant). The DTW calculates the distance (normally the Euclidean one) between all the points of the two time series and, then, generates another matrix with the accumulated distances. The total distance defined by the path formed with the minimum values of the accumulated distance (Figure 2.6, right-hand side) can be easily applied to compare different shapes.

The DTW calculations presented in this thesis were all executed using the author's implementation of DTW. This version of the algorithm uses a C kernel, supporting multidimensional arrays and Euclidean distance, to speed up the calculations with a Python wrapper as the user interface. The library is open source and freely available online on Github [116].

2.4 Conclusions

This chapter presented the relevant literature and main tools used through this thesis. The specialised software tools developed during this PhD project and presented in this chapter (Sections 2.3.6 and 2.3.7) are available online in the author's Github account (github.com/ricardodeazambuja).

In the following chapters, a series of experiments where robot arms are controlled using Liquid State Machines (Section 2.3.3) are presented. For all experiments, the spiking neuron model (LIF, Section 2.3.2.2) always use the parameters from Table 2.1.

The experimental chapters start with a new implementation of the current state-of-the-art Liquid State Machine robot arm controller (Chapter 3) based on Brian (Section 2.3.6.1) and the extension introduced in Section 2.3.6.2. This is the only chapter where a 2 degree-of-freedom simulated robotic arm is employed.

Beginning in Chapter 4, the humanoid robot Baxter (Section 2.3.4) is controlled by the implemented systems. From Chapter 5, all the simulations are based on the Bee simulator (Section 2.3.6.3) and the results analysed using the Dynamic Time Warping (Section 2.3.7).

Chapter 3

Implementation of a Liquid State Machine Robotic Arm Controller

3.1 Introduction

In this chapter, a new implementation of a biomimetic robotic arm controller based on a SNN using the the Liquid State Machine (LSM) framework (Section 2.3.3) is presented. Moreover, a novel detailed study using almost $20,000$ simulations is analysed in order to verify the system's robustness, precision and accuracy, generalization and influence of some network parameters as the Short-Term Plasticity (STP), but always considering the whole trajectory instead of only the final position as the benchmark.

The neurorobotic controller presented here (Figure 3.1) uses an LSM in a feedback loop for the control of a simulated two-joint robotic arm (Figure **??**) to perform a total of four straight line trajectories (Figure 3.4) where each trajectory has a duration of $500ms$. This work started based on a previous controller presented in [26] where the input system had 300 LIF neurons (see Section 2.3.2.2) distributed in 6 individual layers of 50 units (each layer corresponding to one input variable), a liquid (or reservoir) with 600 LIF neurons distributed in a cuboid shape ($20 \times 5 \times 6$) and a linear readout that received inputs from all the liquid's neurons (after a low-pass, or neuron membrane, filter) generating the future torque values for the two joints. The LSM output produced the necessary joint torques

and the arm physics simulator converted those values into velocities and joint angles. The reference for the joint control were the endpoint Cartesian movements that followed a velocity profile that mimicked what is done by the human arm according to [117]. The main aim of the authors in [26] was to prove that an LSM with fast dynamics could control an arm where the movements would take several milliseconds using only linear readouts and, doing that, benefit from a biologically plausible proprioceptive delay of 200ms.

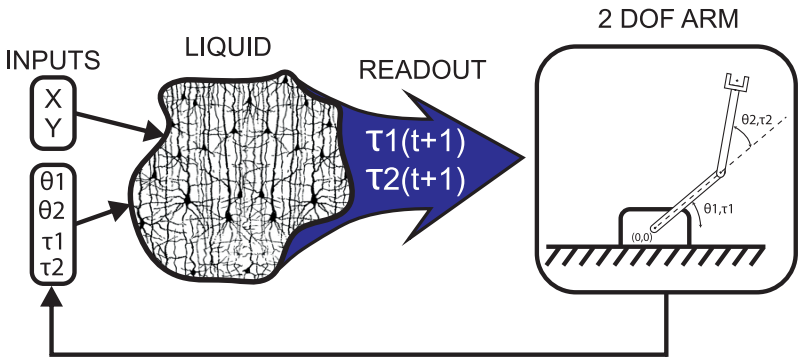


FIGURE 3.1: Illustrative representation of the arm controller.

While reviewing the work presented in [26], several open questions were found. They investigated more critically only the final positions reached by the system instead of focusing on the trajectories analysis. STP (Section 2.3.2.3) was modelled into the neural network, but results concerning the controller's performance in relation to it were not presented. The robustness of the system to the decimation of the SNN or to different levels of noise was not investigated. Several applications would benefit from controllers able to withstand high levels of noise or being partially damaged as discussed in Chapter 1 and also explored in Chapters 6 and 7. In addition, the initial system had an input layer made of extra 300 LIF neurons, while here, to optimise the simulations by reducing the total number of neurons, this layer was abstracted and the values are injected directly into the liquid. Still, the inputs followed a Gaussian distribution of weights working like a receptive field.

Using the new results from the simulations presented here, this chapter explores the open questions mentioned above, extends generalisation capability tests and verifies the effects of trajectory longevity. Besides, it introduces a new system, fully implemented in Python, based on Brian (Section 2.3.6.1), where in the future other researchers could use, reproduce the experiments and extend more easily.

3.2 Methods

The following sections present the methods applied in a robotic task comprised of learning to reproduce four distinct trajectories using a biomimetic robot arm controller. They start with the development of a simulated 2 DOF robotic arm based on an energy based method (Euler-Lagrange) and go through the details of trajectory generation, Liquid State Machine implementation and necessary experimental set-up to reach all objectives presented earlier.

3.2.1 Arm physics simulation

In this chapter, for all the simulations, a simulated two-joint planar robot arm (Figure 3.2) was used, ignoring friction and gravity to simplify the system. In order to promote possible future comparisons, the simulation parameters follow what was presented by Joshi and Maass [26] with link lengths (l_1, l_2) equal to $0.5m$, centres of mass located in the middle of each link $(l_{c1}, l_{c2} = 0.25m)$, moments of inertia (I_1, I_2) equal to $0.03kg.m^2$ and both link masses (m_1, m_2) equal to $1.0kg$. The position of the first joint - what in a humanoid robot would be the equivalent to the shoulder - was moved to the Cartesian position $(0, 0)$ in order to shorten the calculations (Figure 3.2). The generated trajectories (Figure 3.4) had a total duration of either $500ms$, $1000ms$ or $2000ms$, always using a time step of $2ms$. Consequently, each trial produced 250, 500 or 1000 individual values for each variable. The choice of the baseline values (total duration of $500ms$ with time step of $2ms$) was based on the literature [26] and more details about the experimental set-up employed here can be found in Section 3.2.5.

The forward and the inverse kinematics were calculated with a mix of analytical and numerical methods (available on the author's Github repository) to guarantee always smooth transitions of the joint angles as it is well known that the inversion of trigonometric functions in most systems produces always answers only in a specific quadrant. Movements were generated (computation of the state variables) solving the resultant differential equations of the Euler-Lagrange method in a numerical way using *odeint* from the Python package SciPy [118]. This method generates the next position based on the time step (Δt) and

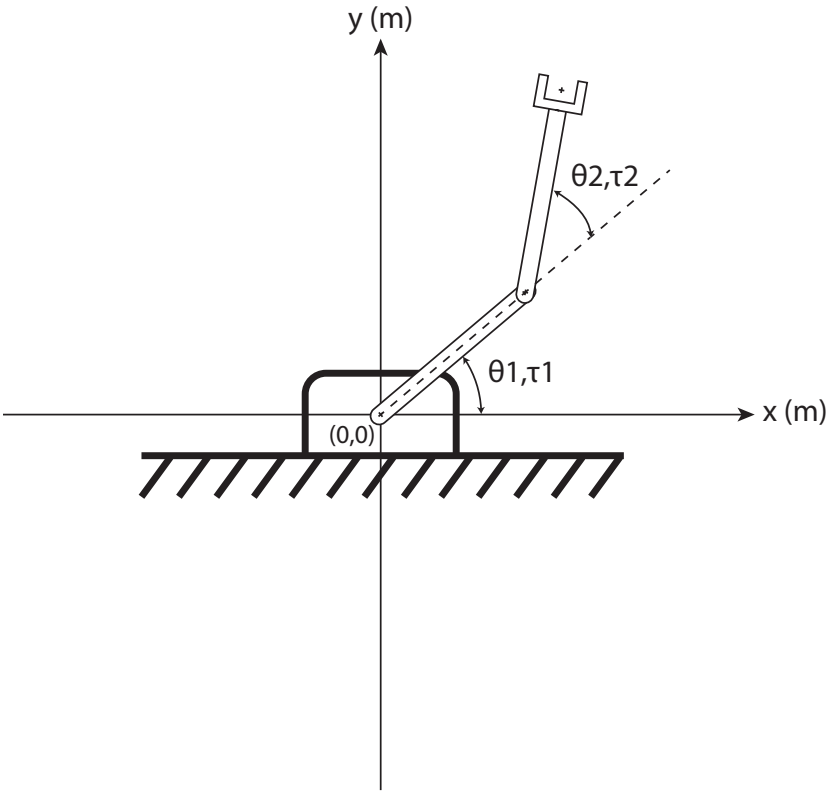


FIGURE 3.2: Two-Joint arm used for the experiments in this chapter.

in the last state variables (position and velocity). From the velocities, the accelerations are calculated and finally the necessary torques to generate the movement. So the system always needs to know the current state and the precision is based on the size of the time step.

3.2.2 Trajectory generation

A robot arm can only reach positions inside its workspace and, in the case presented here, that means the points seen in the Figure 3.3. The joint ranges were roughly inspired by the ranges from a commercial humanoid robot (Baxter Robot, from Rethink Robotics Inc.). A human arm also has limited joint ranges (try to scratch the centre of your back and you will notice them), and that gives extra support to the design choices made here. Therefore, it was not possible to simply reuse the trajectories from [26] in order to have a rich set of

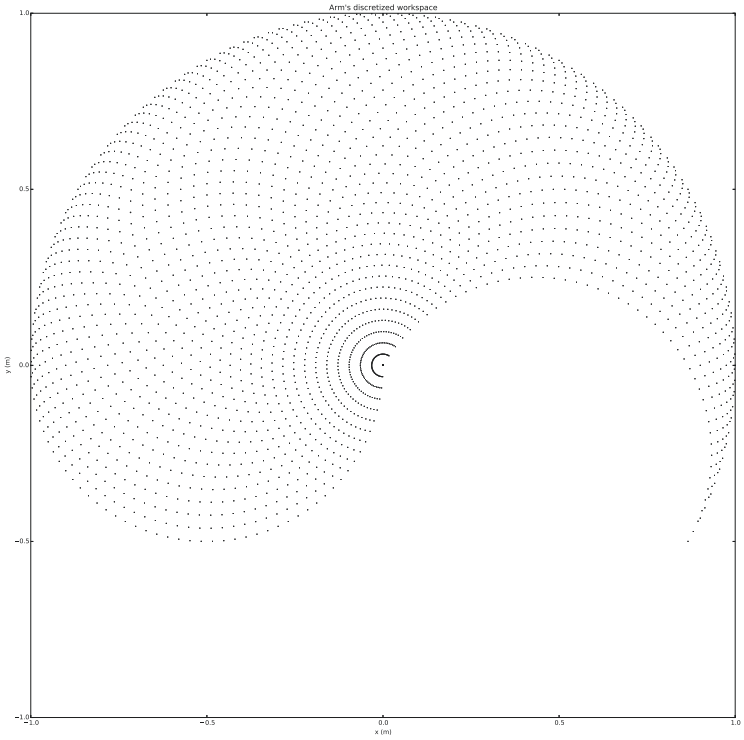


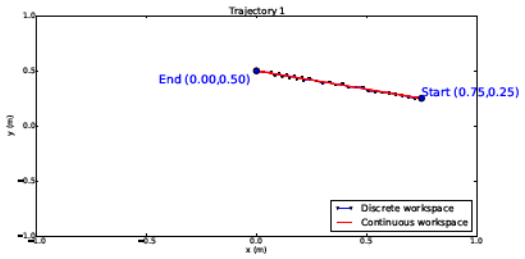
FIGURE 3.3: Workspace of the arm presented in Figure 3.2.

movements that would fit the workspace of the simulated arm presented here. Moreover, in the next chapters it will be presented new controller implementations for the arm of the Baxter Robot using the same range restrictions and workspace.

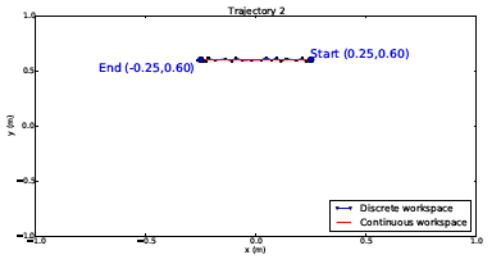
In total, four 2D trajectories were created (Figure 3.4):

- ***Trajectory 1*** - Start $\Rightarrow$ End positions: $(0.75, 0.25) \Rightarrow (0.00, 0.50)$
- ***Trajectory 2*** - Start $\Rightarrow$ End positions: $(0.25, 0.65) \Rightarrow (-0.25, 0.60)$
- ***Trajectory 3*** - Start $\Rightarrow$ End positions: $(-0.10, 0.75) \Rightarrow (-0.10, 0.25)$
- ***Trajectory 4*** - Start $\Rightarrow$ End positions: $(-0.75, 0.50) \Rightarrow (-0.40, 0.00)$

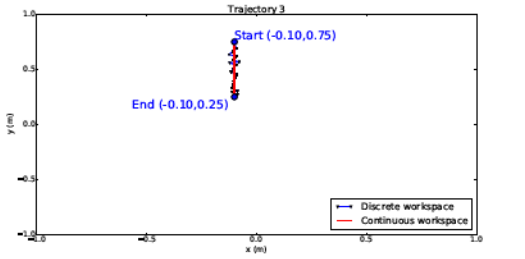
At first, the joint angle discretization was generated creating a linear distribution of values that goes from the lower to the upper joint limit. The result was an array with 50 positions



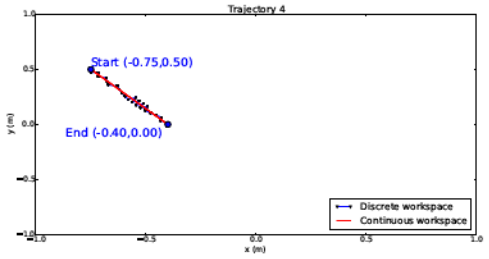
(A) Trajectory 1 - Total distance 0.79m



(B) Trajectory 2 - Total distance 0.5m



(C) Trajectory 3 - Total distance 0.5m



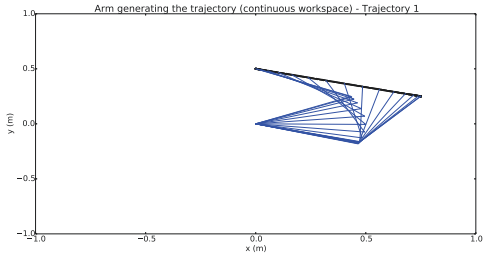
(D) Trajectory 4 - Total distance 0.61m

FIGURE 3.4: Trajectories (continuous and discrete) used to train the LSM. The discrete version was generated by dividing the joint range into 50 possible values.

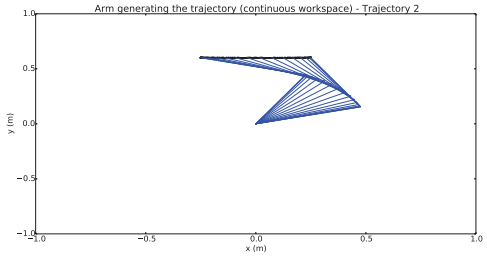
where each position was mapped to a possible floating point value following the input configuration based on a simplified population code. Array indices behaved as individual input neuron. To find the index yielding the nearest input value, the absolute value of the difference between the input and all the possibles ones available in the array was calculated and the index found by the Numpy method *argmin*. After the visualization of resultant discrete and continuous trajectories (Figure 3.4), it started to become clear that the generation of the state variables would not work using these discrete trajectories because sharp transitions would generate huge acceleration values.

Trajectory generation starts with endpoint velocity Gaussian shaped curves (calculated as presented in [117]) that must be converted to the continuous joint angle space. These angles can be seen in the arm movement necessary to follow the trajectories (Figure 3.5). In the case of the discrete version, the resultant joint angles (not shown here) were not smooth.

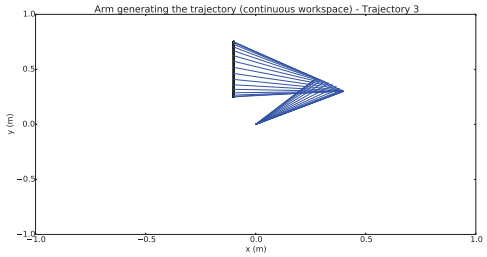
Using simple derivatives, it is possible to calculate the joint velocities and accelerations



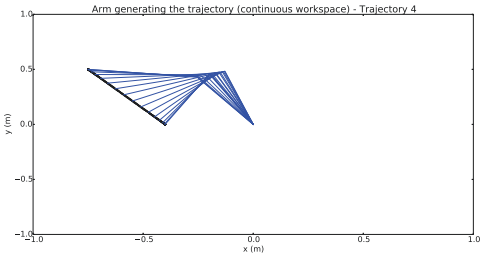
(A) Trajectory 1



(B) Trajectory 2



(C) Trajectory 3

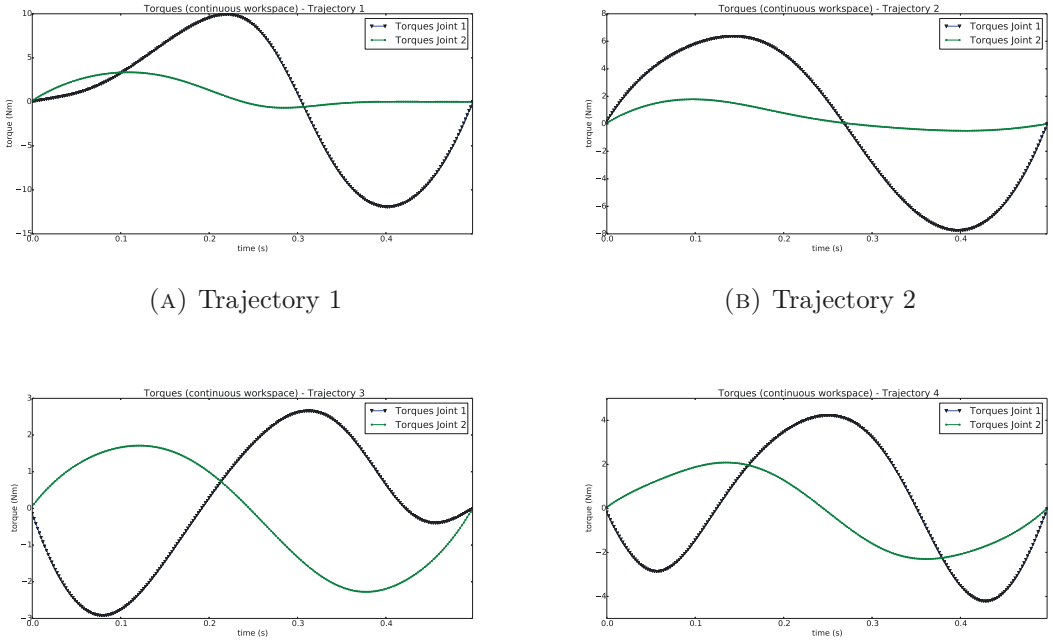


(D) Trajectory 4

FIGURE 3.5: Arm movements necessary to generate the continuous trajectories seen in the Figure 3.4

(not shown here) necessary to generate the torques for the continuous and discrete trajectories (Figure 3.6). Since the calculated torques using the discrete trajectories (not shown here) yielded infeasible values because of abrupt changes in directions (see discrete workspace trajectories in Figure 3.4) and step functions have a infinite derivative on the transition, continuous ones were used being discretized into fifty values to match the number of neurons available in the implemented LSM input layer.

Additionally, as an adaptation to increase the input resolution, the values used in the training and the testing phases were discretized based in the minimum and maximum values generated by the four trajectories. While this range selection improves the resolution, it limits the possible trajectories to be generated as well.



(A) Trajectory 1

(B) Trajectory 2

(C) Trajectory 3

(D) Trajectory 4

FIGURE 3.6: Joint torques (continuous) necessary to recreate the Cartesian trajectories from Figure 3.4

3.2.3 Liquid State Machine Simulation

One very important difference in the LSM arm controller implemented here is related to the input layer. Instead of an LSM defined as normal a three layer system with input, liquid and readout layers, here the input layer is abstracted and the values are injected into the liquid as if they were spikes generated by a *virtual* input layer. In previous works, it was not provided a strong technical necessity of having a separate input layer, which in a simulation adds an extra time delay and uses more computational resources. Moreover, the input layer was implemented using Brian Simulator's *SpikeGeneratorGroup* with the help of Brian Step-by-Step extension (see Section 2.3.6.2). This simplifies the system avoiding the simulation of around 300 LIF neurons.

For reservoir computing in general, in a feedback system the use of noise is crucial during the learning phase to make the system robust to small variations in the feedback values during the testing phase [25]. Therefore, here a discrete noise source was used directly in the input neurons (depending on the experiment set, some inputs don't receive noise). A value drawn from a discrete uniform distribution (Numpy method *randint*) going from

$min = -1$ to $max = 1$ was used. That way, it is possible to have a better control over the input spikes when compared with a situation where an analogue noisy value is used (remember the input is made of 50 discrete values). Also, because the feedback during the testing phase is made of spikes instead of analogue values, this discrete noise mimics the real testing situation.

The final LSM configuration consisted of three layers: Input, Liquid and Readout. The input layer was composed of 300 virtual neurons subdivided into 6 groups. The liquid was created using 600 LIF neurons. As a sub-layer after the liquid, there is a low-pass (neuron membrane) filter that processes the spikes before they reach the readout. The use of a low-pass filter makes it possible to convert spikes into analogue values while filtering noise below its cutoff frequency (value directly related to its RC constant). The final configuration of the liquid used through all the experiments was already presented in the Figure 2.2. Since the liquid has a small-world connection pattern, it is important to highlight the formation of short and long range connections and the way they are distributed. The formation of islands are more easily seen through the Figure 2.3 where contour plots are made summing all the connections on each plane.

With 300 virtual neurons in the input and 6 variables, there are only 50 unique neurons available to represent each variable (simplified population code). A lookup table was created to convert each variable into neural code. Hence, to facilitate the conversion, an offset was added to the final tables to adjust them to one of the 6 available neuron groups. The use of an offset creates the same effect as a lookup table, but it can use an unidimensional array instead of a table. Each input group was associated to one of the input variables as following:

1. X-Coordinate of the end position: linear distributed values between -1 and 1 and no offset.
2. Y-Coordinate of the end position: linear distributed values between -1 and 1 with an offset of 50.
3. Joint 1 - proprioceptive angle: linear distributed values between $-\frac{\pi}{6}$ and π with an offset of 100.

4. Joint 2 - proprioceptive angle: linear distributed values between 0 and π with an offset of 150.
5. Joint 1 - proprioceptive torque: linear distributed values between the maximum and minimum ones that occurred during the training phase with an offset of 200.
6. Joint 2 - proprioceptive torque: linear distributed values between the maximum and minimum ones that occurred during the training phase with an offset of 250.

The conversion of the analogue values to the discrete ones associated with each input neuron was made using the nearest available in the respective range.

Connections between the input layer and the liquid were created using a Gaussian distribution as a receptive field. This curve modulates the values of the weights between pre-synaptic and post-synaptic neurons with a certain standard deviation (the default value is 3 neuron positions, see Figure 3.7). Therefore, each connection is spread to the nearest ones creating a certain redundancy. Also, in order to increase the separability, each input group is directly connected only to an unique slice of the liquid. An example of the resultant weights, generated to follow a Gaussian distribution, for the second variable (Y-Coordinate of the end position) can be seen in the Figure 3.7. The neuron index position inside the liquid in this situation is presented in the Figure 3.8.

3.2.3.1 Least squares linear regression (LSLR)

A linear regression is always optimal, fast and naturally diminishes, smooths, information that is not relevant. Furthermore, a system using linear regression inherits the ability to average over conflicting information making it more robust to a noisy environment.

When the LSM framework is employed, usually a linear regression is implemented to train the weights connecting the readout to the liquid (after the low-pass filter). The generalised linear model (Ordinary Least Square - OLS) from Scikit-learn [119] Python's package was used here with the following settings: *fit_ intercept=True, normalize=True, copy_ X=True*. This method solves the following problem:

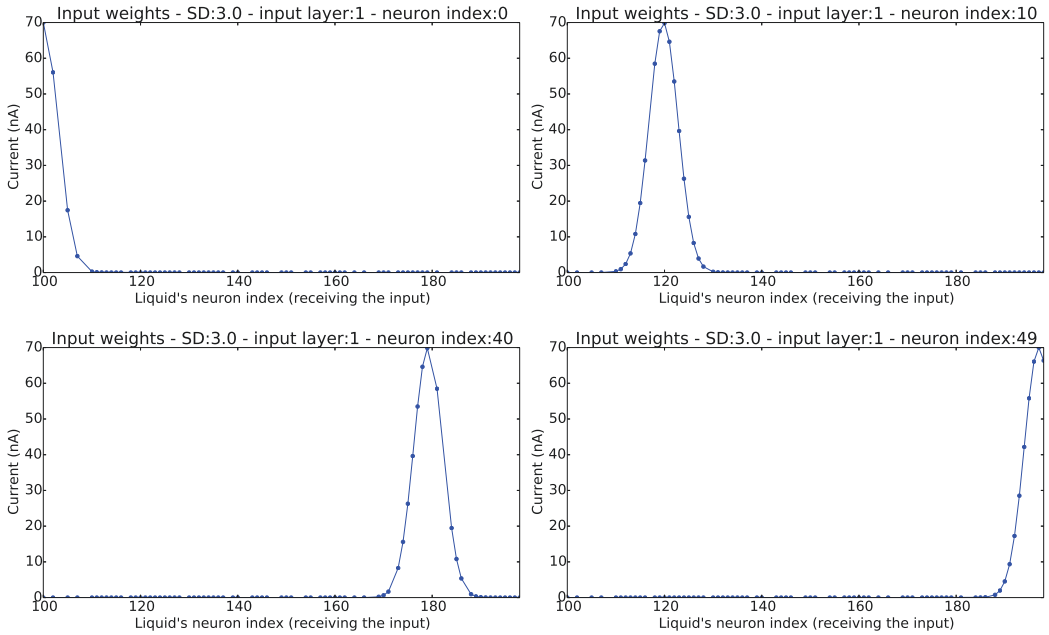


FIGURE 3.7: Example of resultant weights values connecting the input to the liquid (standard deviation equals to 3 neuron positions). The abscissae axis has the liquid's neuron index.

$$\min_{w} \|Xw - y\|_2^2 \tag{3.1}$$

In this work, the matrix X was composed by the membrane low-pass filtered values of all liquid spikes generated during a particular movement ($\tau_m = 30ms$), y the joint torque values necessary to generate that trajectory and w the readout weights. In order to improve the results by making the OLS less prone to generate weights with outliers [120], and after pilot experiments (not presented here), Gaussian White Noise (GWN) with $\mu = 0$ and $\sigma = 0.1$ was added to the X matrix and $\sigma = 0.01$ to the y. For the sake of comparison, Figure 3.9 shows the resultant weights of one experiment where noise was used with the readout training and Figure 3.10 shows the same situation without the use of noise. The weight distributions can be seen on Figures 3.11 and 3.12. During pilot experiments, without the noisy version of the LSLR, the weights were not evenly distributed and only a few outliers were responsible for the final value. In this situation, the system became very sensitive to noise, generating abrupt changes in the output, and incapable to reproduce the results during the testing phase.

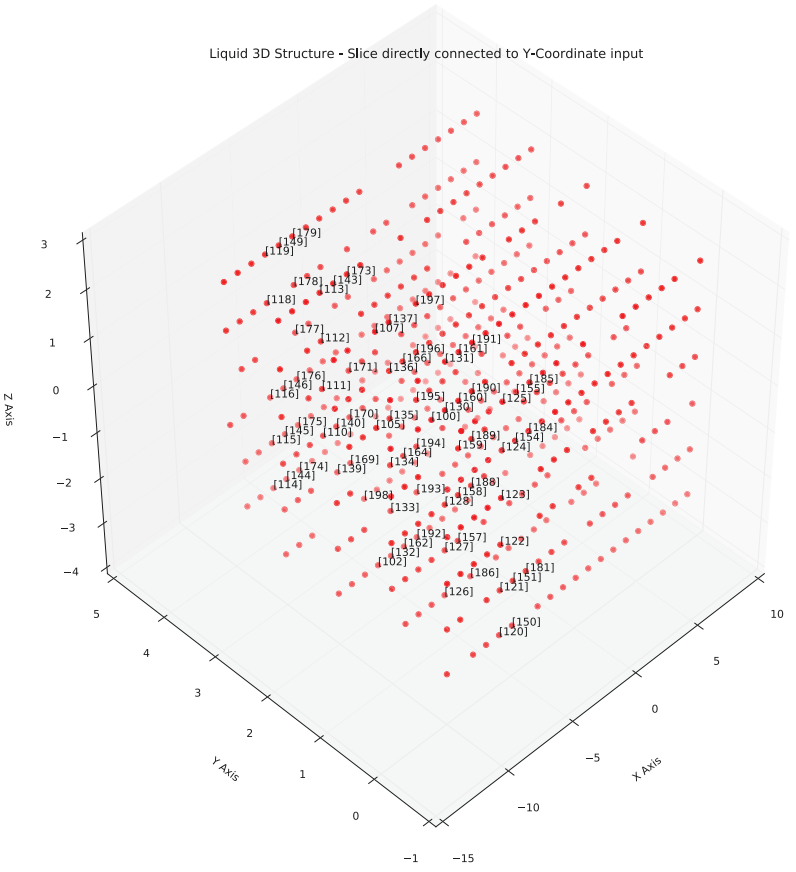


FIGURE 3.8: Liquid's neurons indicating the location of the indices seen in Figure 3.7. Because the inputs are connected only to excitatory neurons, the figure shows only the red (excitatory) instead of all neurons as in the Figure 2.2.

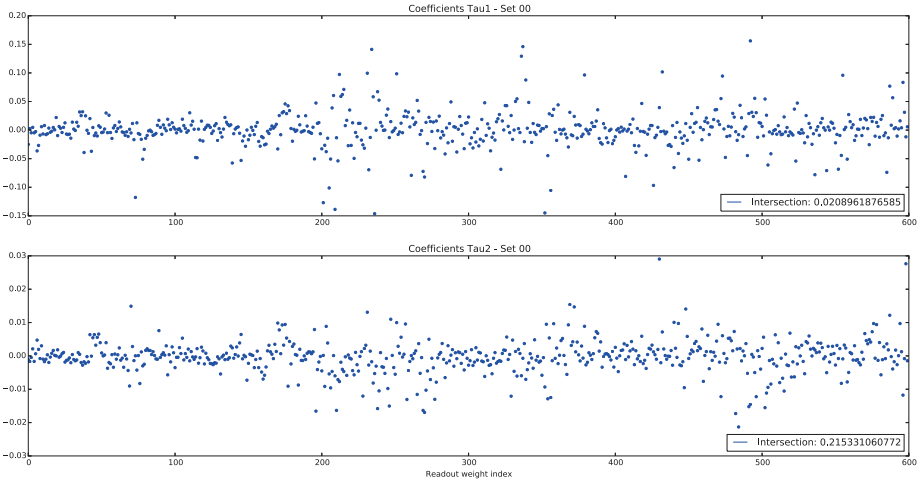


FIGURE 3.9: Example of readout weights generated by the LSLR *with* the injection of noise.

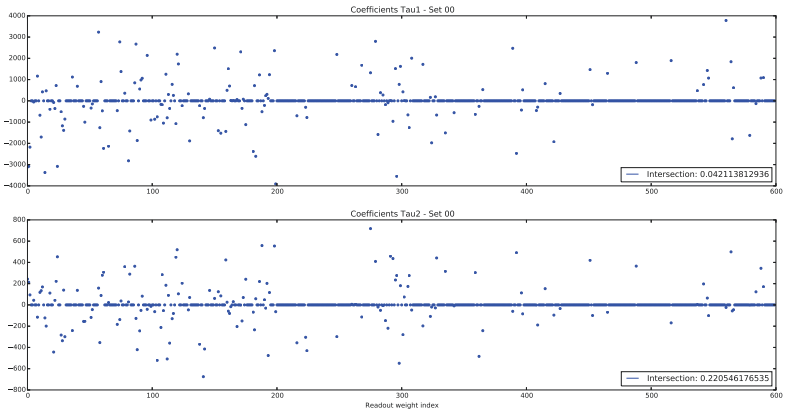


FIGURE 3.10: Example of readout weights generated by the LSLR *without* the injection of noise. When it is compared to Figure 3.9, becomes clear the heavy generation of outliers in the noiseless situation (the *y–axis* max. values from Figure 3.9 are more than 1000× smaller).

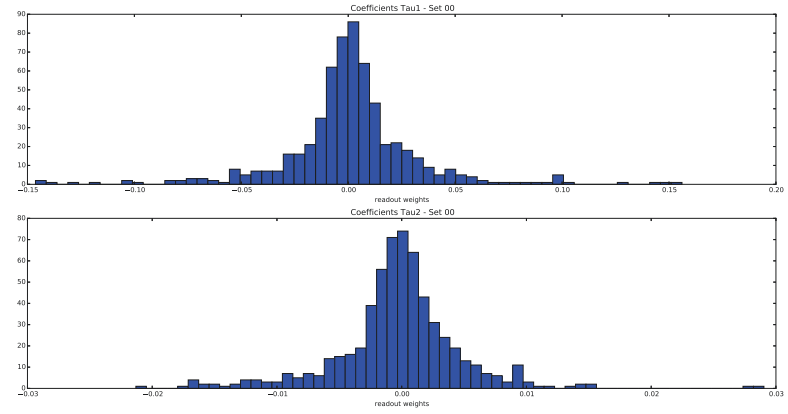


FIGURE 3.11: Distribution of the weights (with noise) seen in the Figure 3.9.

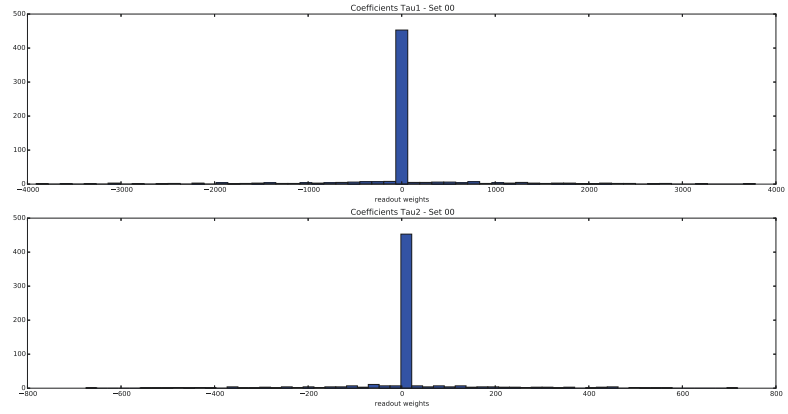


FIGURE 3.12: Distribution of the weights (without noise) seen in the Figure 3.10.

3.2.4 SNN Simulations

As discussed in the chapter's introduction, all the SNN simulations used in this work were performed by the Brian simulator (see Section 2.3.6.1) together with a specialised extension (Brian-Step-By-Step, Section 2.3.6.2) instead of PCSIM [121].

Moreover, all the simulations in the training phase were executed in parallel to speed-up the process and save time. Because IPython Notebooks were used to develop all the code and generate the data in this chapter, its built-in parallel processing module was the chosen one to run the simulations.

The IPython notebooks also helped to speed-up tasks during the testing phase even without the direct use of the parallel processing module by simply opening multiple notebooks (kernels) and manually executing them simultaneously. The use of IPython notebooks made the development process self-documenting and repeatable.

3.2.5 Experimental set-up

Two groups of experiments (Group 1 and 2) were carried out here, both using the same four trajectories (Figure 3.4) following a Gaussian endpoint Cartesian velocity profile [117]. Twenty trials were used for the training of the readout weights for each configuration. In Group 1, the total time spent during the trajectories was the same as presented by Joshi and Maass [26], where the endpoint moved from start to end lasting $500ms$ using 250 steps. A higher spatial resolution was used for the trajectories in Group 2 and the total time went to $1000ms$ using 500 steps and $2000ms$ using 1000 steps. This last set was necessary to study the effects of the increase in the trajectory time resolution.

The basic LSM simulation parameters, taken from the literature [26], were employed for both groups of experiments (Group 1 and 2). Random variables for the generation of the liquid's basic structure (excitatory and inhibitory neurons, connections and weights as is fully explained in [21]) were seeded always using the same value (using a Numpy RandomState with seed value equal to 93200) in order to keep the same liquid across the simulations. The only parts of the liquid that were not kept the same along the simulations

were the variable noisy currents (i_{noise} from Equation 2.1), the initial membrane voltages and the noisy offset initial currents (i_{offset} from Equation 2.1). The mentioned variables were seeded randomly (Numpy's default) and drawn from their respective distributions [26] for all simulations. In the case of the noise source represented by the variable i_{noise}, a new value was drawn for each simulation's time step.

All experiment sets consisted of two phases: training and testing. During the training phase, a forced teaching approach (supervised learning) was used and the data collected was exclusively employed to train the readout weights to generate the trajectories. Because the robot arm model used here has two joints, the system was divided into two readouts - one for each joint - trained individually to generate the next (future) value of the joint's torque based on liquid's current state. Each readout was connected to all 600 liquid's neurons through individual membrane low-pass filters and the response from those filters then connected to the readout by 600 individual weights. Those weights were calculated using the Equation 3.1 during the training phase. During the testing phase the weights do not change.

One last important point to highlight here is that all trials are always unique. This fact comes from noise injected into the system at every simulation time step in addition to the ones during the initialization. The networks presented here are always changing as initialization voltages and offset noises are changed also during the testing. Only the readout weights are kept fixed after the training phase. This is a quite different situation when compared to the way the testing is done with neural networks when there's no noise involved, besides the initial one for the training phase.

3.2.5.1 Experiment Group 1

The training and testing phase experiments were chosen to help in the understanding of the system behaviour when different strategies (parameters sets) are used. The noise levels were varied because initial pilot experiments showed that too much noise can break the readout learning ability. This is the reason why all experiments noise levels are specified in relation to the base (default) ones [26].

Two types of noisy offset current (see Equation 2.1) had their values varied, a normally distributed i_{noise} and an uniformly distributed i_{offset}. The i_{noise} default values were $\mu = 0$ and $\sigma = 1nA$ and i_{offset} default ones were from $13.5nA$ to $14.5nA$ [26]. As an example, when the values of i_{noise} and i_{offset} appears divided by 100, it means i_{noise} is generated using $\mu = 0$ and $\sigma = \frac{1}{100}nA$ and i_{offset} from $\frac{13.5}{100}nA$ to $\frac{14.5}{100}nA$. During the testing phase only i_{noise} was varied.

The spread of the input weights into the liquid (standard deviation of a Gaussian - Figure 3.8) is known to help in the robustness when neurons are decimated since it creates a redundant code and could also improve the generalization potential. The range tested here was based on the value manually adjusted during pilot experiments (not presented here).

In a SNN simulation, STP is computationally very expensive factor and consequently it is important to verify if the system could work as well as without it in a robotic control implementation as presented here. Neuromorphic systems as SpiNNaker [30] can have the maximum number of neurons simulated greatly reduced when extra rules as the one closely related to STP (as up-to-date this has not been implemented in SpiNNaker code yet), the Spike-timing dependent plasticity (STDP), are used.

In addition to the experiments explained above, the proprioceptive delay was varied from zero to a value bigger than the total time of the trajectory in order to verify its real role. Thus when the joint angle values are fed back from the beginning of the simulation, the system has enough information to know where the endpoint is and this situation could increase the generalization potential. The antagonistic situation, when the delay is bigger than the total simulation time, verifies if the proprioceptive delay was improving anything or just adding more noise.

Lastly, the way noise was injected in the inputs was varied during some experiments. Again based on the pilot tests, noise was only necessary in the fed back torque values, but in order to verify other effects (as the improvement in the generalization) some of the sets had noise also in other input variables.

Training Phase: Nine sets of experiments were used varying the noise levels, use of STP

TABLE 3.1: Training Phase Parameters - Experiment Group 1

Set	i_{noise}	i_{offset}	STP	Proprioceptive Delay	Input Noise
A	Default/100	Default/100	YES	Default	Only at torques.
B	Default/10	Default/10	YES	Default	Only at torques.
C	Default	Default	YES	Default	Only at torques.
D	Default	Default	NO	Default	Only at torques.
E	Default	Default	YES	$2000ms$	Only at torques.
F	Default/10	Default/10	YES	$0ms$	Only at torques.
G	Default/10	Default/10	YES	$0ms$	At torques and angles.
H	Default/100	Default/100	YES	Default	At all input variables.
I	Default/100	Default/100	NO	Default	At all input variables.

and proprioceptive delay (see Table 3.1). The default values are from [26]. In total, 720 simulations were executed ($9 \times 4 \times 20$) only for the training of the readouts. Because during the first step of the simulation the liquid's response was only related to the noisy initialization (the input spikes take one step to propagate in Brian), those initial states were always ignored (*washout*). According to Joshi and Maass [26], the initial noisy states were not a problem because 20 trials were used during training phase. However, in this thesis, it was opted to ignore the initial noisy spikes (the first simulation time step) since the simulation takes one time step to propagate the inputs and those initial spikes were not only noisy, but pure noise.

Testing Phase: First, the trained readouts were tested through new simulations using the same trajectories from the training against three levels of noise (1×, 2× and 10× the amount of noise - variable i_{noise} from Equation 2.1 - used during training for each set) and also the resilience against the random decimation of the liquid's neurons (0%, 1% and 10% neurons). With 9 sets, 3 noise levels, 3 decimation levels, 4 possible trajectories (324) and 50 trials for each combination, a total of $16,200$ simulations were carried for the testing phase. Those results were used to verify which configuration set was able to better reproduce the trajectories and if the implemented systems had the ability to degrade gracefully. In order to verify if the readout had redundant information, the Set A (the one with the lowest level of noise) was tested against the decimation of the readout weights

where a total of 0, 6 and 60 connection weights had their value switched to zero, for 5 randomly chosen trials.

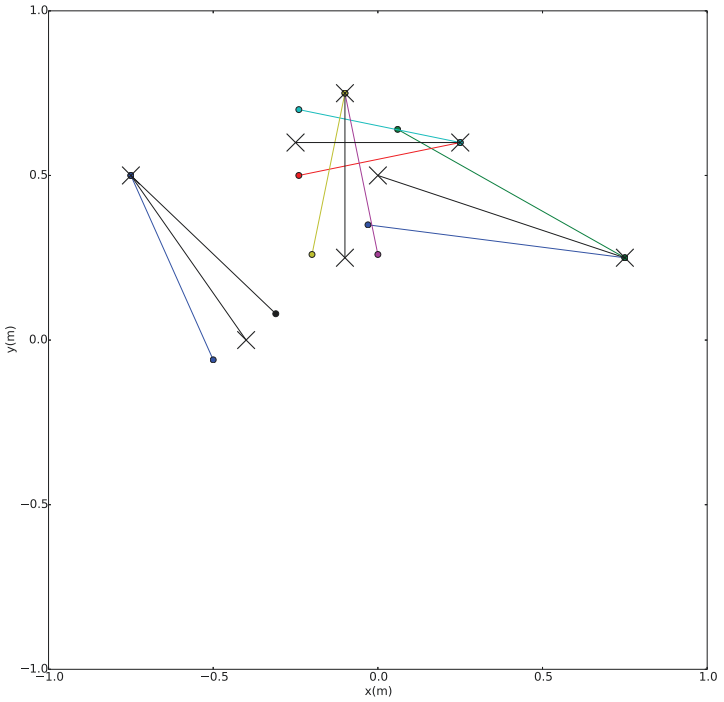


FIGURE 3.13: Original trajectories (from Figure 3.4) are showed here with a "X" and the new ones used for the generalization tests with a dot.

To verify the generalization potential, eight new trajectories were employed using always starting positions from the training set by only rotating the final position by $+\pi/16$ and $-\pi/16$ to generate a total of 8 small movements with multiple directions (Figure 3.13). The sets F and G were used because in these experiments there was no proprioceptive delay and therefore it is the unique situation where the system would have enough information to generate a line equation connecting the start to end. The sets H and I were used because they received noise into all the input variables during the training phase and it is a good way to verify if the noisy inputs can improve the generalization even though having a smaller total SNR through the liquid. Set A was used as it has the smallest total noise level (considering the injected noise at the inputs) and sets C and D were employed to verify, together with sets F and G, the influence of STP on the generalization.

3.2.5.2 Experiment Group 2

This group verified if the time spent in the trajectories could be extended (therefore slowing down the movements) and yet readout weights still properly trained. Also the proprioceptive delay was analysed as a percentage of the total trajectory time. Because in this group the trajectories are generated with a higher spatial resolution (more simulation steps), the results could show the ability of the system to deal with a bigger number of trajectory points without increasing the number of neurons inside the liquid.

Training Phase: It was composed of four base set-ups (see Table 3.2) where twenty trials for each of the trajectories were executed (4x20x4=320). The main difference here is that the total time spent in the trajectories was twice and four times as big as in the Group 1 - instead of $500ms$ was used $1000ms$ and $2000ms$. This data was used to train the readout weights.

TABLE 3.2: Training Phase Parameters - Experiment Group 2

Set	i_{noise}	i_{offset}	STP	Proprioceptive Delay	Input Noise	Total Time
J	Default/100	Default/100	YES	$200ms$	Only at torques.	$1000ms$
K	Default/100	Default/100	YES	$400ms$	At torques and angles.	$1000ms$
L	Default/10	Default/10	YES	$400ms$	At torques and angles.	$1000ms$
M	Default/100	Default/100	YES	$800ms$	At torques and angles.	$2000ms$

Testing Phase: Tests to verify the resilience against noise and decimation of the liquid or the generalization skills were not carried out here because the increase in the number of simulation steps also increases the total time spent making those tests too time consuming. Here the resultant systems from the training phase were tested (5 trials for each set) to verify the consequences of the parameters ranges used.

3.2.5.3 Analysis tools

Normalized Curve Error (NCE): The metric used to verify how close to the ideal curve was the response of the system during the testing phase was the Normalized Curve Error (NCE). The Equations 3.2, 3.3 and 3.4 introduce the method in detail.

$$NCE = \frac{MeanDistance}{TotalLength} \tag{3.2}$$

$$MeanDistance = \frac{1}{N}\sum_{n=1}^{N}\|P_{Ideal}(n) - P_{Test}(n)\| \tag{3.3}$$

$$TotalLength = \sum_{n=1}^{N}\|P_{Ideal}(n) - P_{Ideal}(n-1)\| \tag{3.4}$$

The NCE used in this work was defined as the mean Euclidean distance between all the points of both curves (Equation 3.3) normalized by the total length of the ideal one (Equation 3.4). Using this formulation the results are unit-less and proportional to each curve.

Apart from NCE, Euclidean distance between current position produced by the LSM and desired one was used to compare the temporal evolution of the error during the trajectory generation.

3.3 Results and Discussion

3.3.1 Experiment Group 1

The experiments included tests made to analyse the LSM against different noise levels, effects of the STP use, decimation of the liquid and readout. This is novel because, in the author's knowledge, in the literature don't exist experiments where the robustness of LSM or the influence of dynamical synapses using STP are tested using a robotic application as the benchmark.

In [26], they have confirmed the LSM ability to learn the trajectories and even showed some small generalization capability, but the system presented here is not simply reproducing

their experiments. As explained in the methods section, several aspects are different as, for example, the SNN simulator, the way the input variables are discretized and injected in the liquid, the use of a virtual input layer and even the readout training. Therefore, it was necessary to test the LSM again in order to verify if trajectories were learned and to some extent the generalization potential of the system.

3.3.1.1 Precision and Accuracy of the trajectories generated and the joint curves learned

Since all the parameter sets were able to accomplish the training phase reproducing the most important characteristics of the joint torque curves, here the exposition of the results is focused in the testing phase only. Just to give the reader an idea of the type of output verified during the training phase, the readout comparison of the joint torque curves for the Set C (the one using only the default values) is shown in the Figure 3.14. As the noise levels are lowered in most of the other parameter sets, the output curves standard deviation decreases too.

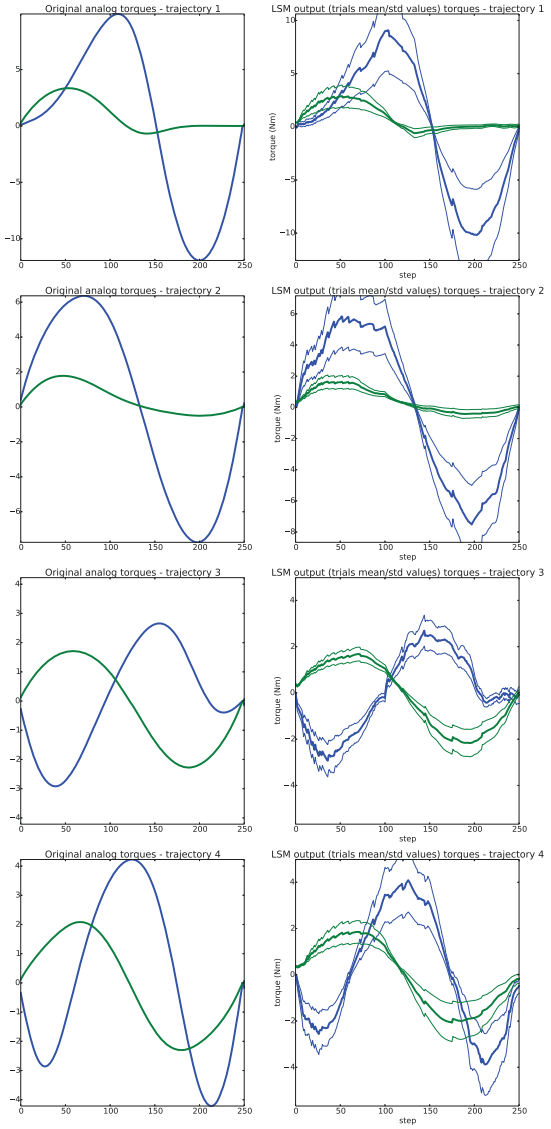


FIGURE 3.14: Training curves and output during the readout training - Set C (see Table 3.1) - Experiment Group 1. The curves in the right are readout outputs for the training inputs where the thicker lines show the mean and the thinner ones the standard deviation for all twenty trials. The curves on the left are the same from Figure 3.6.

Due to the huge amount of data generated during the testing phase (16, 200 experiments in total), first a simpler analysis was made using the NCE (Equation 3.2) keeping constant the liquid's amount of noise and decimation for each trajectory. The results can be seen in the Figure 3.15. With a fast visual inspection, it is possible to verify that some parameter sets had results that are significantly better than the others (smaller Normalized Error).

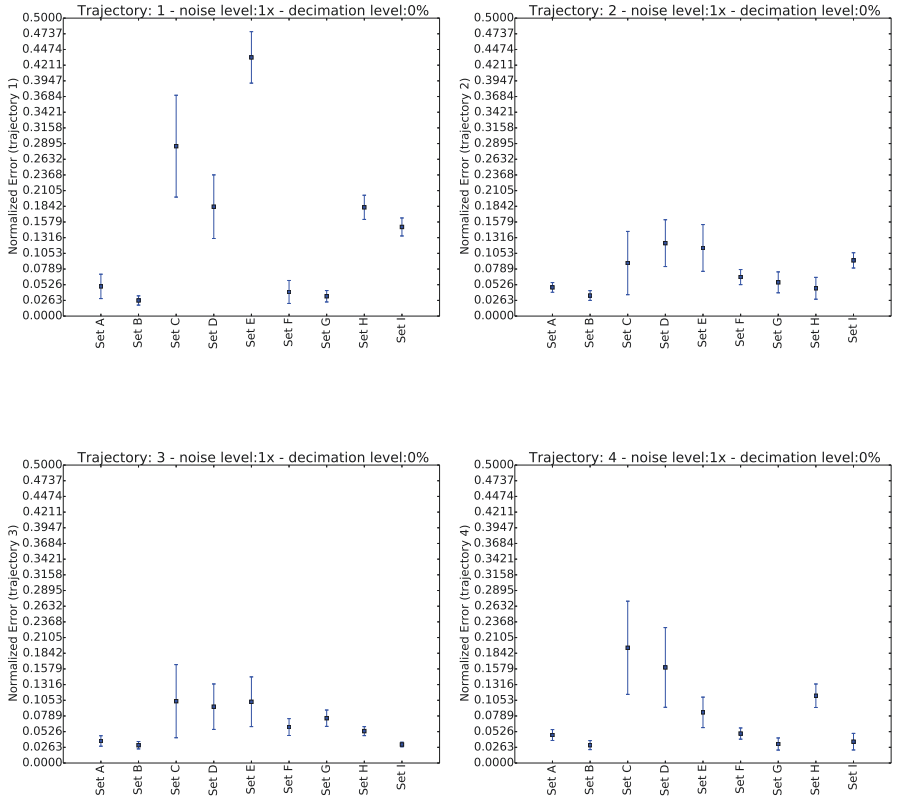


FIGURE 3.15: Results for the *individual* NCE (Equation 3.2) considering the four trajectories from Figure 3.4 individually for all parameters sets (see Table 3.1) in this experiment (Group 1) where the noise levels and decimation are kept constant. Mean values and standard errors are shown.

Ignoring the robustness analysis (where initial noise and decimation levels are varied, to be presented in the following subsections), the results must be compared to when the parameters are the default ones (values presented at Table 3.1). Consequently the starting point is the situation when there's no increase in the noise levels and the liquid is kept intact (noise level 1x and decimation level 0%, see Section 3.2.5.1, *Testing Phase* explanation). Among the results, when all the trajectories are analysed together, the smallest NCE average value comes from the Set B (Figure 3.16). This is an interesting result as default noise levels (used to calculate NCE values in Figure 3.16) in Set A were 10x smaller than in Set B. Between Set B and Set C there is the same noise level increase (10x), but in this second case the NCE increases too. As results are not taking into account generalization abilities at this point, maybe some kind of stochastic facilitation [98] could be happening here when the noise background is between the ones used in Set A and Set C.

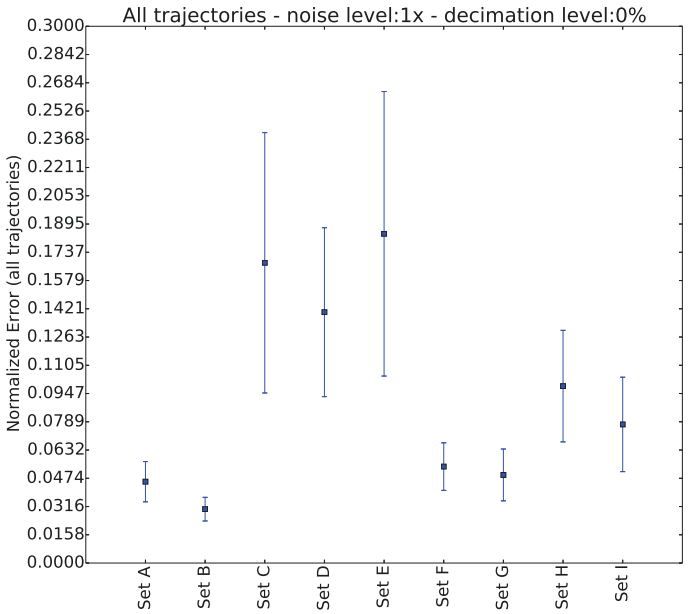
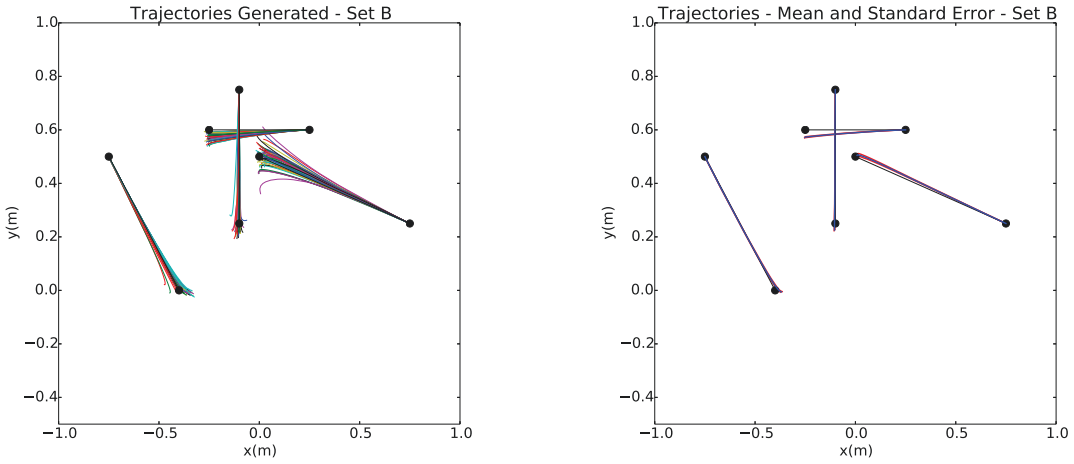


FIGURE 3.16: Analysis of all the trajectories together (*cumulative*) in respect to the NCE (Equation 3.2). Mean values and standard errors are shown. See Table 3.1 for parameters.

Besides being a good metric when analysing the results of many experiments together, the NCE (Equation 3.2) averages along the whole trajectory giving as the result only one single number. Therefore, it is necessary also to use another type of plot in order to have a better idea of the system's behaviour. Trajectories generated by parameter set B used for NCE calculation (Figure 3.16) are presented in Figure 3.17. Inspecting the mean values in the Figure 3.17b only trajectory 2 (the unique horizontal line) had a systematic error, but this type of error is easily corrected with a simple calibration procedure as a bias correction.

The task solved by the LSM in this work was not only to command a simulated 2 degree of freedom robot arm to generate straight trajectories (Figures 3.4 and 3.5), but also to follow a human inspired smooth gaussian shaped velocity profile during the trajectory [117]. The curves calculated based on the generated trajectories for the best arrangement (Set B) can be seen in the Figure 3.18.



(A) Resulting trajectories for the Set B. The coloured lines are the trials and the ones with black circles are the ideal ones.

(B) Mean values (blue lines) and standard error (red lines) of the trajectories generated by all the trials from Figure 3.17a. The ideal trajectories are shown with small black circles.

FIGURE 3.17: Trajectories generated by the Set B (see Table 3.1 for parameters).

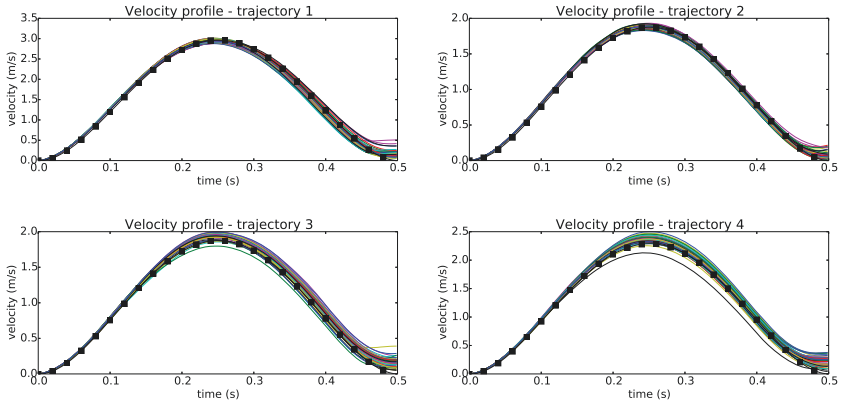


FIGURE 3.18: Gaussian shaped velocity curves for the Set B (Table 3.1). Coloured curves represent the 50 trials while black squares the ideal one.

Starting with the resultant trajectories (Figure 3.17), even with quite a good amount of variability in the results (Figure 3.17a) the average value and the standard error (Figure 3.17b) show a good fit to the ideal curves. This is one of the characteristic of systems based in noisy neurons: they may not always produce good results when it is seen in a deterministic way, but averaged (or statistically) they do.

In order to analyse the trajectories presented in the Figure 3.17, the error distance (the

distance between the ideal position and where the endpoint should be in a particular time step) was calculated. The results for the trajectories seen in the Figure 3.17 are shown in the Figure 3.19.

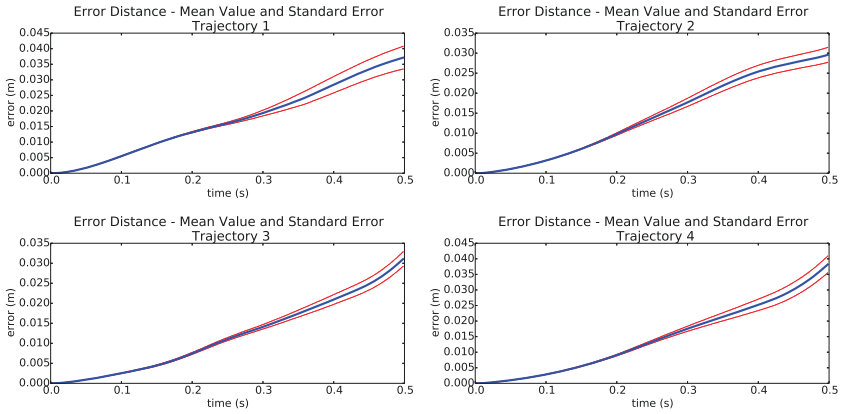


FIGURE 3.19: Evolution of the error distance between the ideal and the current endpoint positions for all the 50 trials (Set B) shown in the Figure 3.17. Red lines: standard error. Blue thick lines: mean value. The error curves follow a well known human error pattern where accuracy declines with increase of speed [18].

The inspection of the Figure 3.19 shows the system's performance using the parameter set B in a different perspective. It condenses in one curve the results from the Figures 3.17 and 3.18. Now the variation in the error starts to appear with more force after the $250ms$ (half the total trajectory) or after the endpoint had reached its maximum velocity. Before that point, the maximum error is below $2cm$ for all trajectories. Yet, the curves have a distinct linear growth while the velocity decreases after half the trajectory. It could suggest the error grow is being generated by the sum of an almost fixed bias during each simulation step.

So far, the examination of the data was done only in a higher level with what was generated after the readout output - no spikes were directly involved. Figure 3.20 shows the averaged output spikes (parameter set B) for all trajectories. Visually comparing them with the original joint torque curves (Figure 3.6) it is easy to recognise some of the shapes. At first, these figures could give the idea that there is almost no interaction between the neurons inside the liquid as the readout is feeding back the values and reading them in sequence. Although, the Figure 3.8 shows that, because of the way the inputs are distributed inside

the liquid, a raster plot cannot properly express where those spikes are happening inside the 3D volume.

Since the connections between neurons are created following a small-world configuration (Figure 2.2) actually all the variables interact to each other according to the connection distributions in the Figure 2.3.

At the first simulation step, all the membrane values are drawn randomly from a distribution as explained in the methods section. Such a random initialization creates a wave and can be seen in the Figure 3.20 as the vertical white lines. As the time passes, the simulation frequency slows down until it reaches approximately $16Hz$. Coincidently beta frequency goes from about 12.5 to 30Hz [122] and according to [123] near to 20Hz occurs one important type of oscillatory occurrence in the motor system related to tonic contractions and voluntary movements.

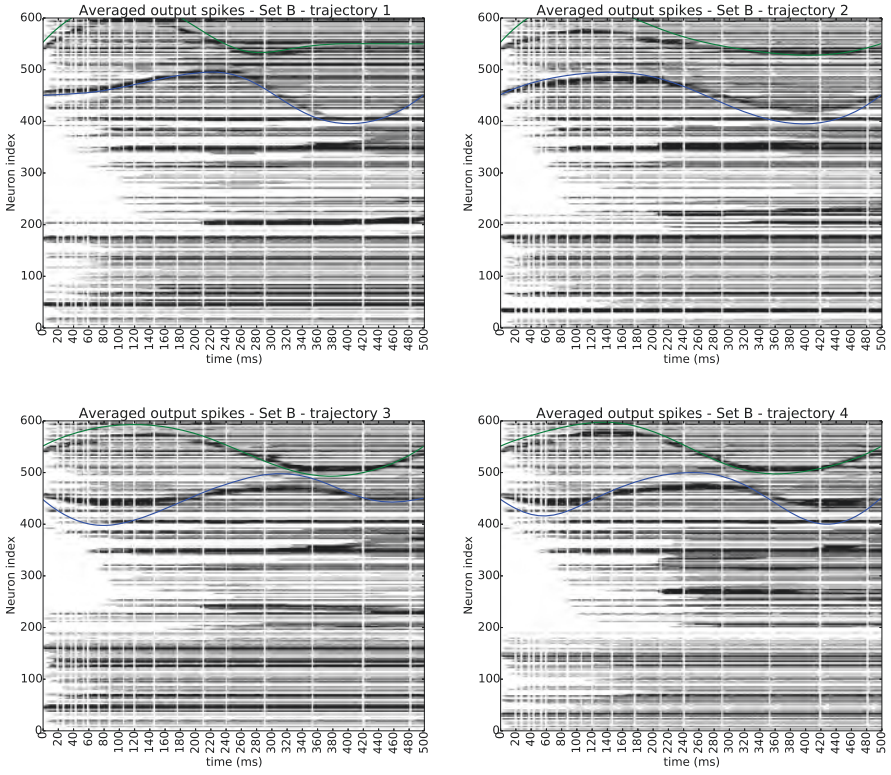


FIGURE 3.20: Averaged values of the output spikes for all the 50 trials (Set B) shown in the Figure 3.17. Values towards one (spike received) becomes darker. Green and blue curves show the original torque values after translation and rescale.

3.3.1.2 STP influence in the LSM performance

Some of the parameter sets were developed specifically to verify the effects of STP within the liquid. The set pairs C/D and H/I contrast only by the use or not of STP. The Figure 3.21 shows at a more detailed level the results of the summed NCE (3.2) for all trajectories within those four sets. The visual inspection of the results helps to conclude the STP had no effect as the differences in NCE between the sets falls within the standard error bars.

The results of the STP tests can be seen as explained in [12] where it's stated STP is only one of the time-dependent neuronal properties available. Therefore, STP does not seem to be an irreplaceable source for more complex network dynamics. Maybe even the organization of the network, with its multiple internal feedback loops, can generate behaviours like STP or even suppress it.

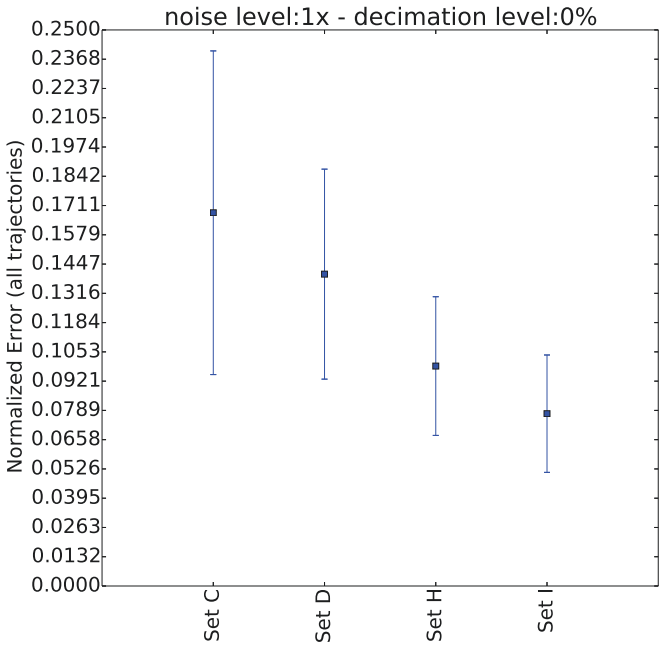


FIGURE 3.21: Results from the Figure 3.16, but showing only the parameter sets (see Table 3.1) created to test the whole of the STP in the liquid's robustness. Sets D and I don't use STP.

3.3.1.3 Robustness of the LSM controller

Robustness was verified through the use of three different levels of noise (i_{noise} from Equation 2.1) and liquid's neurons decimations. The combination of those possible configurations generated a total of nine scenarios. All parameter sets were subjected to fifty trials for each one of those scenarios. The Figure 3.22 exhibits the results of all simulations using the NCE (Equation 3.2) mean value and standard error.

Some of the parameter sets seem to be almost immune to i_{noise} increase. During the training phase, parameter set C used a noise level (i_{noise} and i_{offset}) 10x bigger than parameter sets B, E, F and G. Compared to the Sets A, H and I, parameter set C was 100x bigger. Analysing the first column of Figure 3.22 where only i_{noise} level is varied, it would be expected that NCE would increase for the Sets B, E and F to the same levels of Set C as the last value is 10x the initial noise. The results suggest that the noise source i_{offset} acts in a non-linear way deteriorating the ability to generate trajectories when the i_{noise} is increased. Another possibility is connected to the readout training. As i_{offset} is kept constant after the start of the simulation, it actually generates slightly different neurons each trial. A too low value for i_{offset} makes the liquid less sensitive to inputs (as it is necessary more inputs/s to drive the neuron to a spike), but a value too high could also make it too noisy impeding the readout to be properly trained.

The first row of Figure 3.22 shows NCE's evolution according only to liquid decimation. Visual inspection already shows the LSM's NCE values increase as the liquid is decimated, but not abruptly as would be expected in a common digital system. In order to make completely clear the system presented in this work degrades gracefully as some of its parts fail, the way the trajectories are influenced by the decimation of the liquid is presented in the Figure 3.23.

As a proof of concept, Figure 3.24 shows the evolution of both the noise and decimation levels for the parameter set B (see Table 3.1). Again, the system shows a graceful degradation behaviour that could be useful for operations in extreme environments. Space

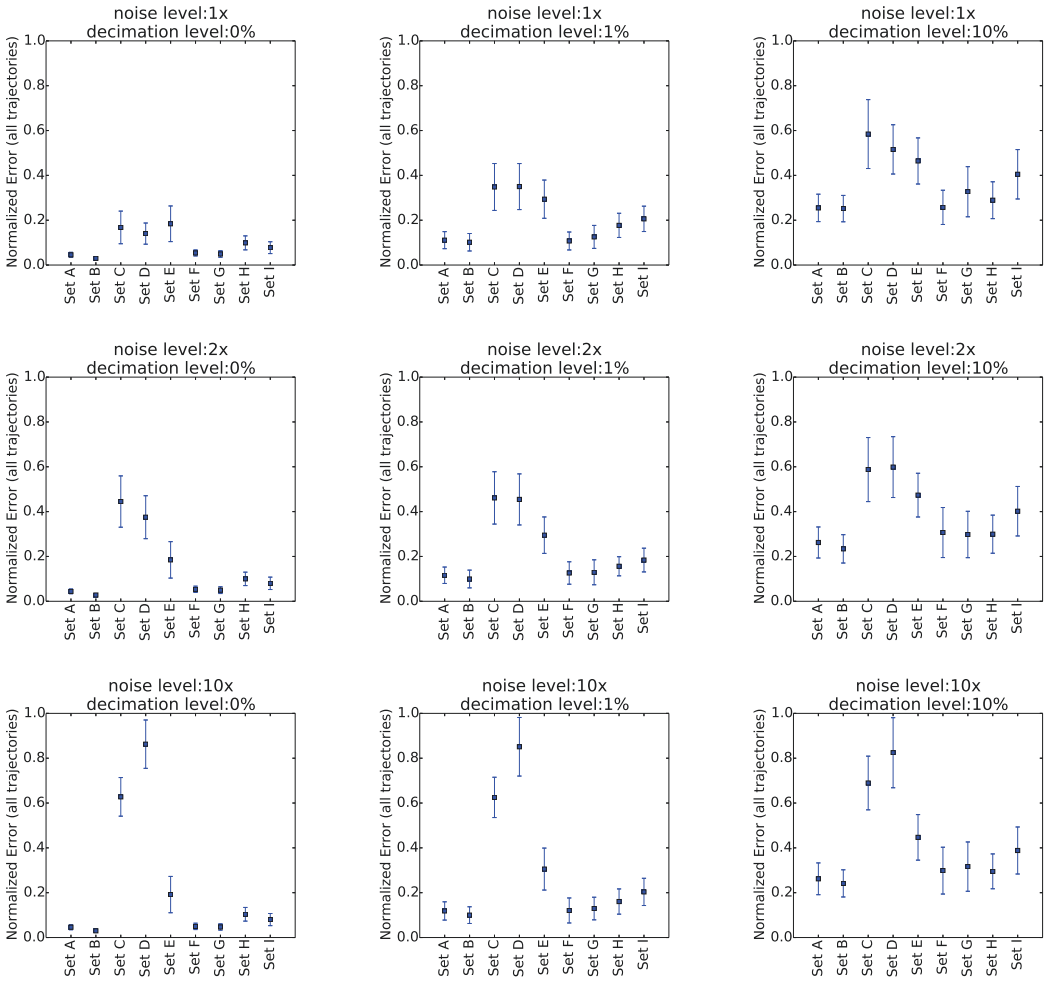


FIGURE 3.22: Robustness tests *cumulative* NCE results (all trajectories). Blue squares indicate the mean values while the blue whiskers show the standard error. Table 3.1 presents the parameter sets.

exploration is a good example of harsh conditions that could benefit from such a robot controller as the one presented here.

Figure 3.25 shows the results of experiments where the readout was decimated instead of the liquid. The parameter set A was used as it was the one with the lowest level of noise during training. When compared to the Figures 3.23 and 3.24 the similarities are evident. The LSM presented here shows a graceful degradation behaviour for the liquid as well readout decimation.

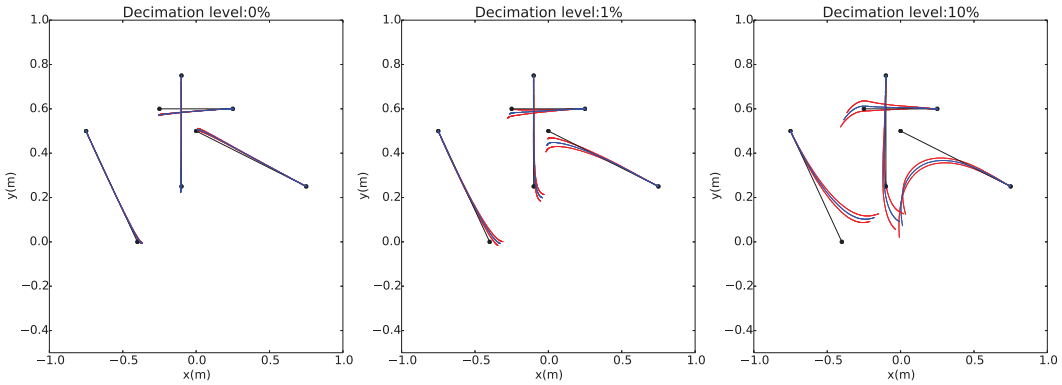


FIGURE 3.23: Evolution of the trajectories (50 trials for each decimation level where mean value is showed in blue and standard error in red) according to the liquid's decimation (Figure 3.22). The parameter set B was used (Table 3.1).

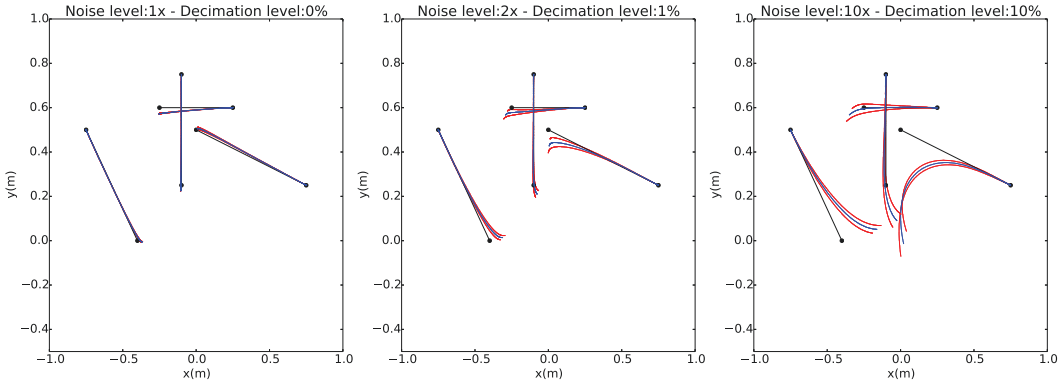


FIGURE 3.24: Evolution of the trajectories (50 trials for each noise/decimation level where mean value is showed in blue and standard error in red). It is the equivalent to the diagonal of Figure 3.22. The parameter set B was used (Table 3.1).

3.3.1.4 Proprioceptive delay

In an LSM, the liquid has the potential to integrate information from all its inputs over more than one time scale [21]. Because of its fading memory [23], in order to have an active liquid it is necessary to have input spikes distributed over time and space. Also the readout can average and be more robust if it has as many as possible inputs receiving values.

For the experiments about the proprioceptive delay influence, parameter sets E, F and G were used. The same noise level was employed during the training phase. Set E had it proprioceptive delay set to $2000ms$ so it never received the feedback during the simulations.

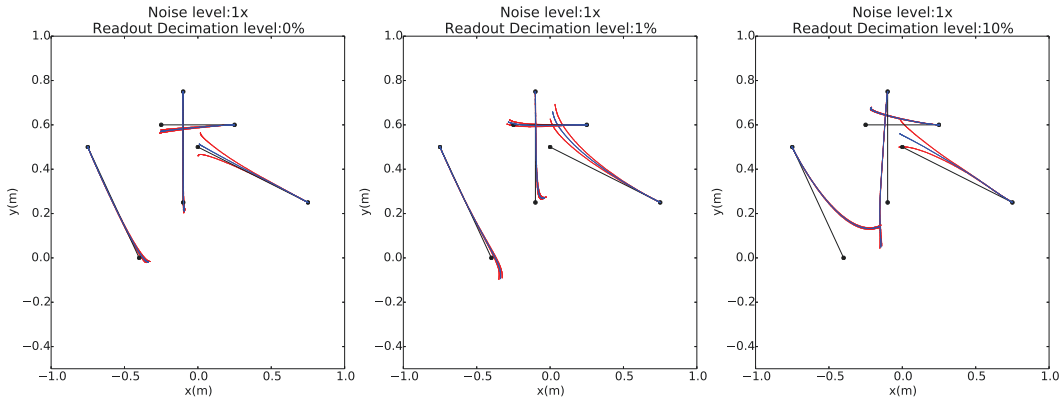


FIGURE 3.25: Evolution of the trajectories (5 trials for each readout decimation level where mean value is showed in blue and standard error in red). The parameter set A was used (Table 3.1).

Both sets F and G use the proprioceptive feedback since the start of the simulation, as the delay was set to $0ms$. In the case of the Set G, the proprioceptive feedback received noise as described in the methods section. Since the Set E has no proprioceptive feedback, the liquid receives less spikes and the readout has not as much information as in the other parameter sets. The results from Figure 3.16 confirms that Set E had a worse performance when compared to sets F and G and also when compared to Set B since it has the same parameters as Set E but the proprioceptive delay. Yet, Sets F and G had a slightly worse NCE when compared to Set B (it has exactly the same parameters as Set F, besides the proprioceptive delay - see Table 3.1) and maybe this could be seen as a confirmation that a proprioceptive delay of $200ms$ generates better results as stated in [26].

3.3.1.5 Generalization capability

Once again the NCE was used to verify the performance of the system (Figure 3.26). Parameter sets C, D, F, G, H and I were chosen for the generalization tests. Sets F, G had proprioceptive feedback from the beginning of the simulations (proprioceptive delay equal to $0ms$). According to the results from Figure 3.26 only the Set G had a significantly worse performance considering all the trajectories used during the generalization verification, but all the other sets had similar results with the values of standard error merging among them.

The variability suggests the results for individual trajectories varied. Trajectories were presented in the results section (Figures 3.28 and 3.29) and in the methods (Figure 3.13).

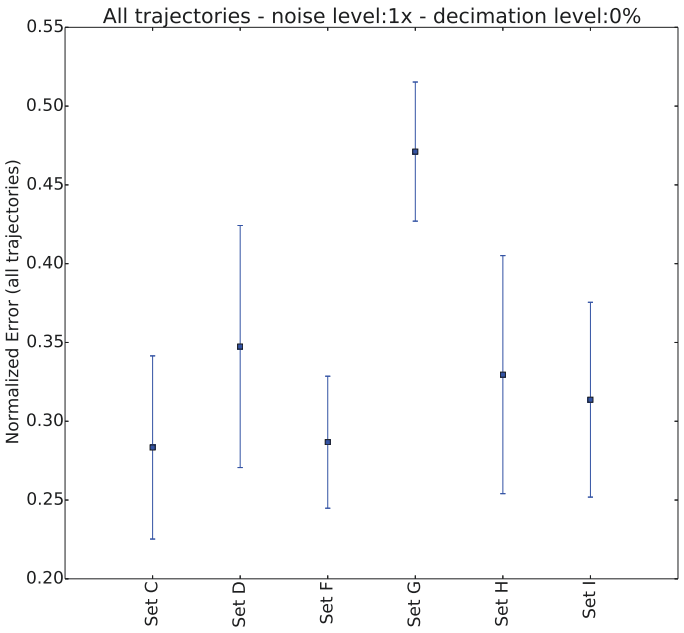


FIGURE 3.26: Comparison of the *cumulative* NCE (all trajectories) used for the generalization tests (see Table 3.1 for parameter sets).

An analysis of the NCE for the individual trajectories can be seen in the Figure 3.27. The plots confirm none of the sets was able to perform well in all trajectories and even Set G was not a clear loser in all them. Also Set C has the biggest variability (represented by the blue whiskers). Looking back the Figure 3.16, Set C had one of the worse results when the original training trajectories were tested.

To visualize what NCE represents in the generated trajectories, sets C and F were chosen. The Figures 3.28 and 3.29 depicts the eight trajectories used during the generalization tests as well the original base one. In several situations the system was able to correctly choose the direction at the beginning of the movement losing the control after a while. Among all the results presented, the Set F - during the trajectory 4 - was the only one able to follow the trajectory along all its length.

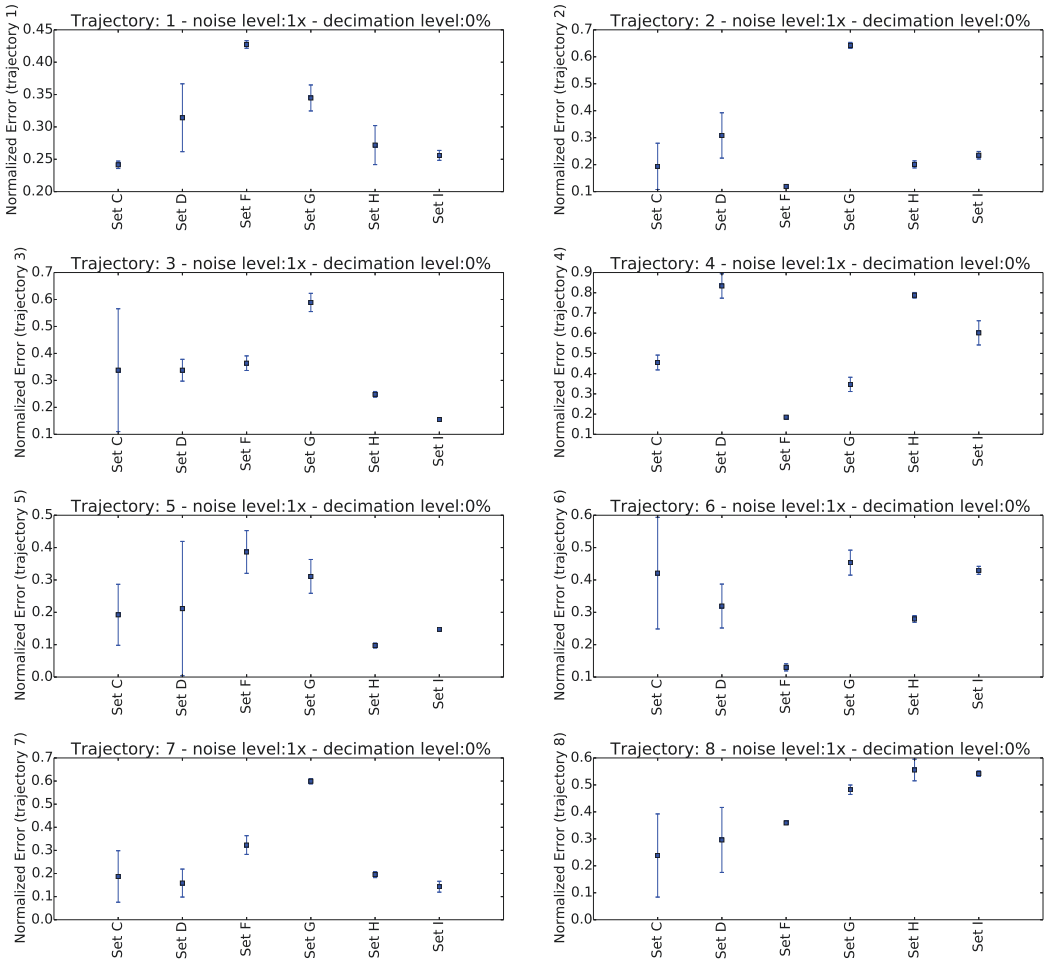


FIGURE 3.27: Comparison of the *individual* NCE used for the generalization tests (see Table 3.1 for parameter sets).

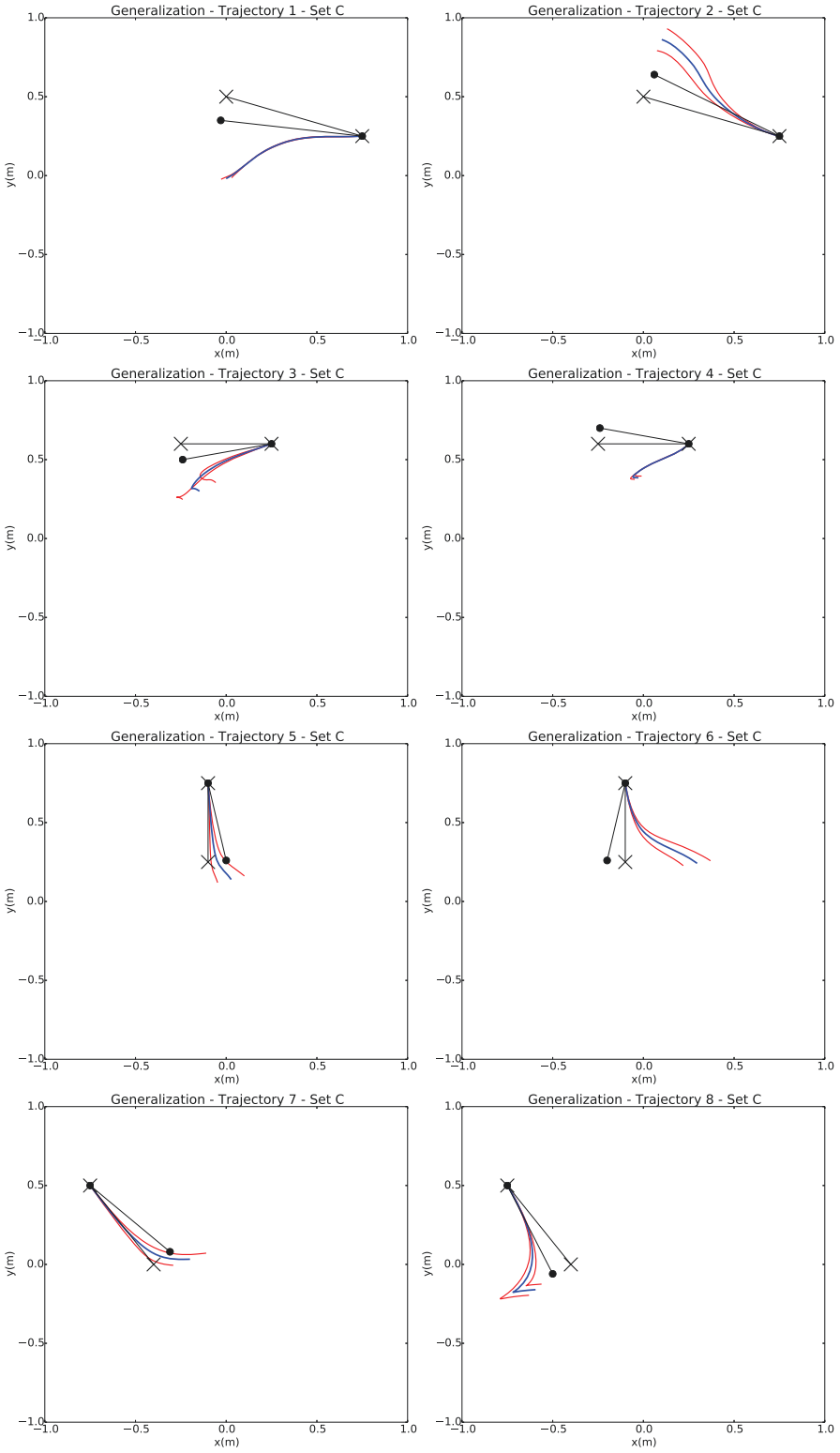


FIGURE 3.28: Generalization test results (Set C, see Table 3.1 for parameter sets). Mean values in blue and standard error in red. Original trajectory (from the training set) uses a "X" marker whilst the new one a black dot.

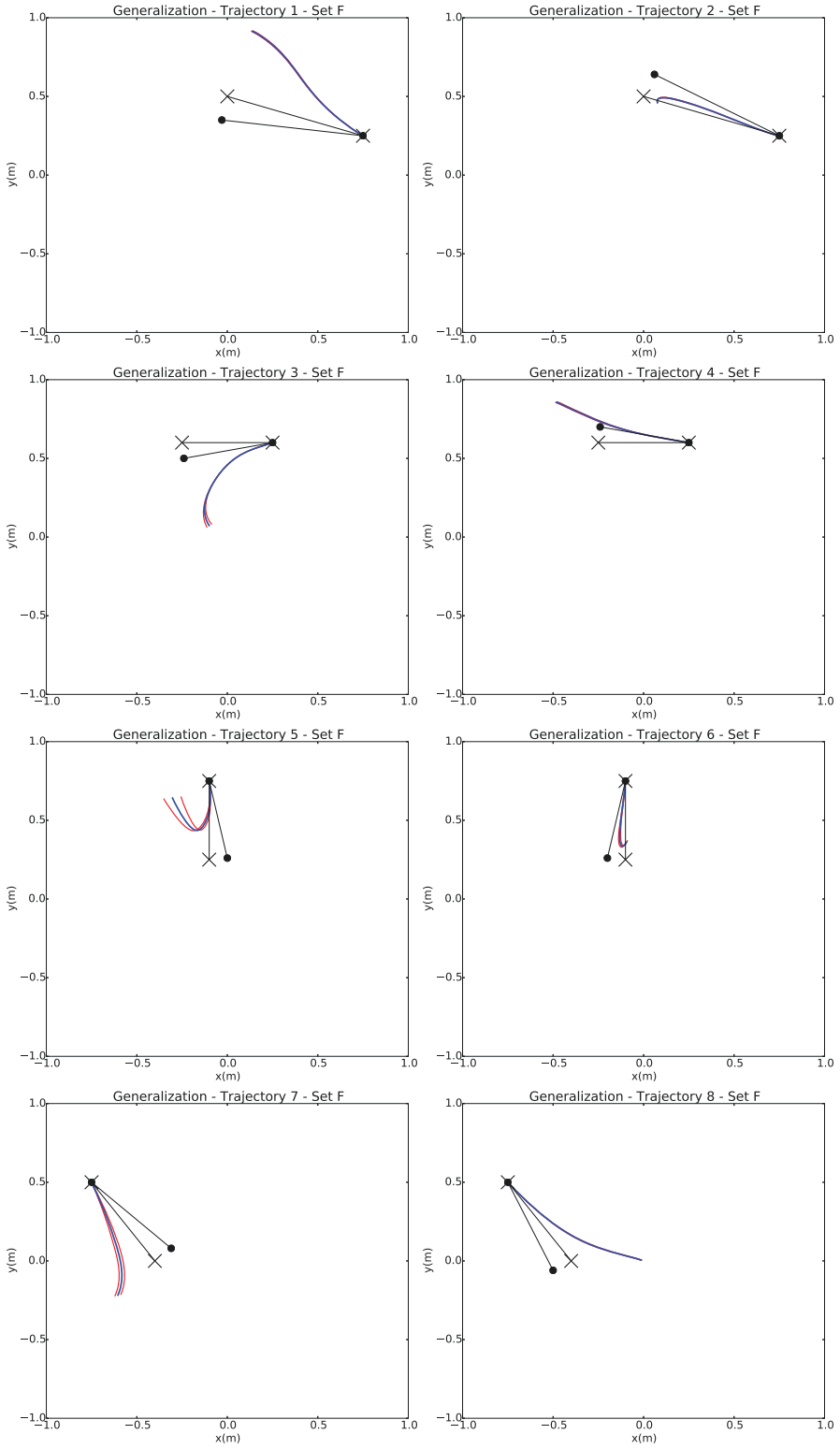


FIGURE 3.29: Generalization test results (Set F, see Table 3.1 for parameter sets). Mean values in blue and standard error in red. Original trajectory (from the training set) uses a "X" marker whilst the new one a black dot.

3.3.2 Experiment Group 2

3.3.2.1 Effect of the increase in the time spent (resolution) in the trajectories

The most important verification done within the experiments in this group was to confirm that the same LSM basic set-up could still generate trajectories using four times as many points as in the Group 1. This means that the LSM potential to store information was not saturated opening the possibility that more than four trajectories could be learned by the same system.

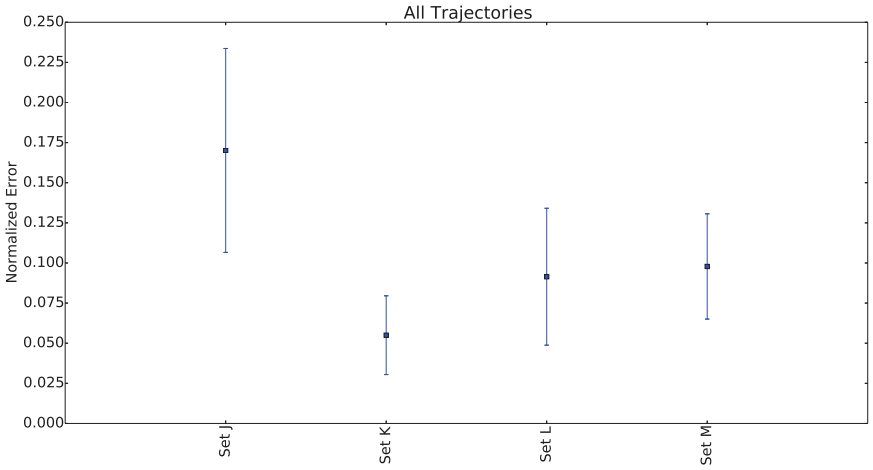


FIGURE 3.30: Comparison of the *cumulative* NCE (all trajectories) - Experiment Group 2. See Table 3.2 for parameter sets.

Overall, the parameters from the simulation Set K led to the best performance (using the NCE metric - Equation 3.2) in relation to all the trajectory curves as can be seen in the Figure 3.30.

However, looking at the NCE results for the individual trajectories (Figure 3.31) the Set J reached error levels much lower than the other configurations for the trajectory 2 (description of the trajectories can be verified in the Figure 3.4).

Visually it becomes clearer that the second trajectory for both sets has a deviation to the bottom when looking at the generated trajectories using the parameters from Set J and

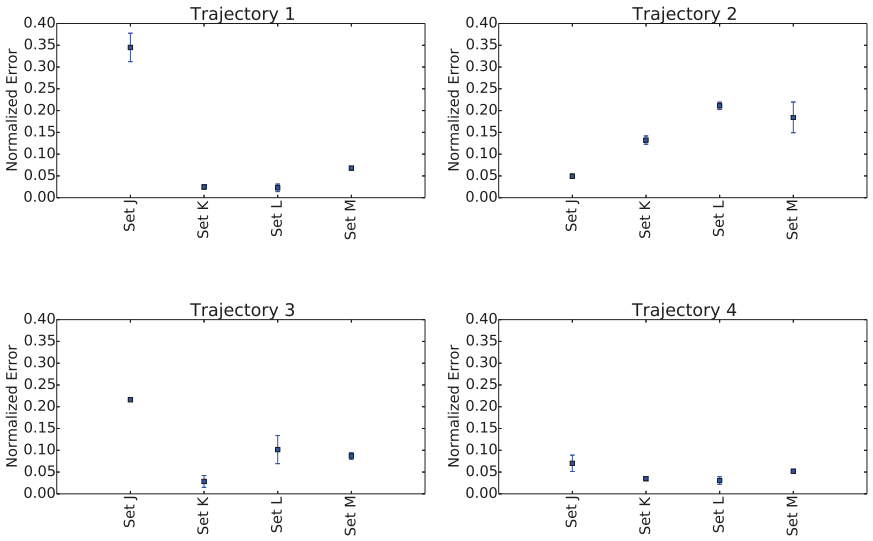


FIGURE 3.31: Comparison of the *individual* NCE - Experiment Group 2. See Table 3.2 for parameter sets.

Set K in the Figures 3.33 and 3.32 respectively. Set L is closer to the line used during training (straight horizontal line with black circles at beginning and end).

The analysis of the torque curves that actually are generated directly by the LSM (the trajectories are the result of those torques applied to the arm model) from the Set K (Figure 3.34) shows that during the trajectory 2 the joint 2 torque curve had a consistent deformation for all the 5 trials resulting in the deviation in that trajectory.

Comparing the results from Sets A and B, it can be concluded that the variation in the proprioceptive delay from $200ms$ (20% of the total trajectory time) to $400ms$ (40%) was responsible for improving the results as all the other factors were kept the same.

Notwithstanding having the biggest noise level (10× bigger than all the other sets in this group), the Set L was still able to generate mostly good results when compared with the Set J. This could be seen as further evidence to back up the theory that the proprioceptive delay of $400ms$ (40%) instead of $200ms$ (20%) was responsible for the bad performance of the Set J.

That contradicts the results from [26] where it is justified that the proprioceptive delay of

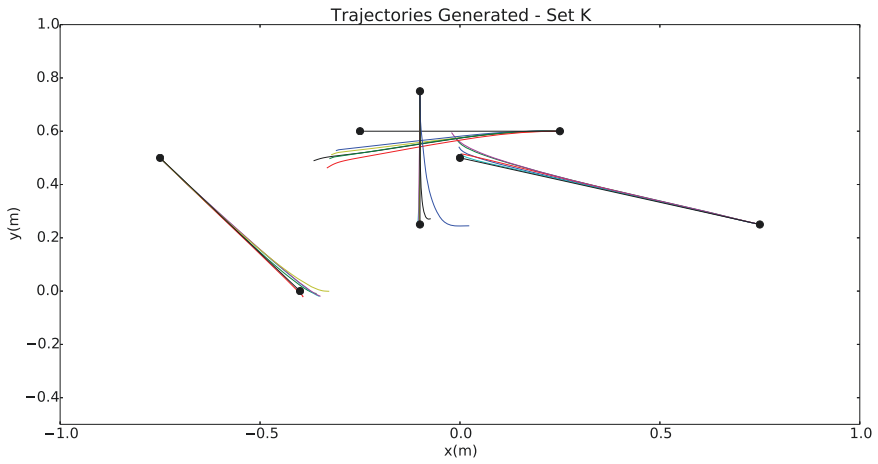


FIGURE 3.32: Generated trajectories during the testing phase by the Set K - Experiment Group 2 (see Table 3.2 for parameter sets). The original trajectories are depicted by the lines with filled circles. Five trials for each trajectory (from the Figure 3.4) are displayed using different line colours.

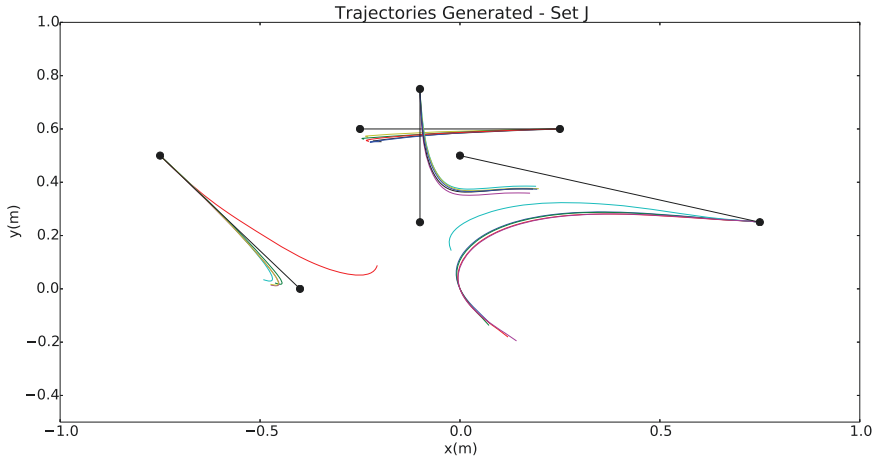


FIGURE 3.33: Generated trajectories during the testing phase by the Set J - Experiment Group 2 (see Table 3.2 for parameter sets). The original trajectories are depicted by the lines with filled circles. Five trials for each trajectory (from the Figure 3.4) are displayed using different line colours.

$200ms$ had a better performance because 200ms is biologically relevant. Maybe the percentage compared to the whole trajectory (40%) and not the value ($200ms$) was responsible for the better performance. The experiments done here are not completely the same as in [26], so more studies are necessary to confirm the true role of the proprioceptive delay.

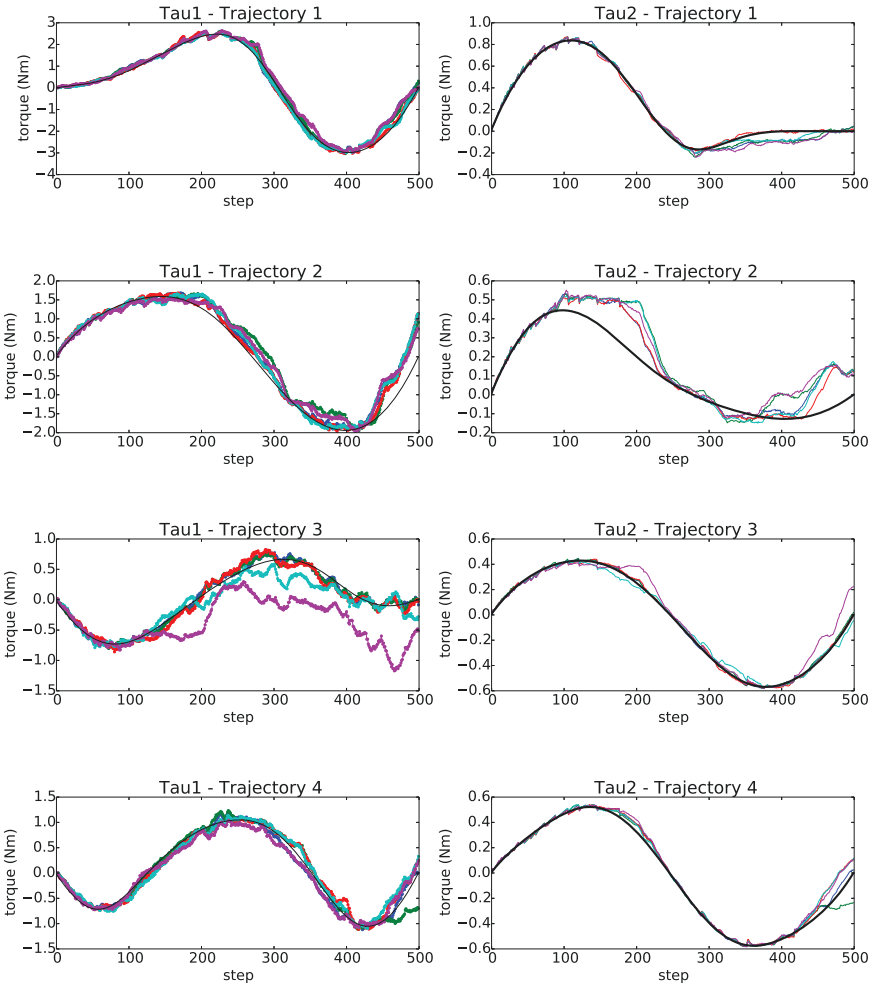


FIGURE 3.34: Resultant torque curves from the testing experiments (Figure 3.32) using the parameters from the Set K - Experiment Group 2 (see Table 3.2 for parameter sets).

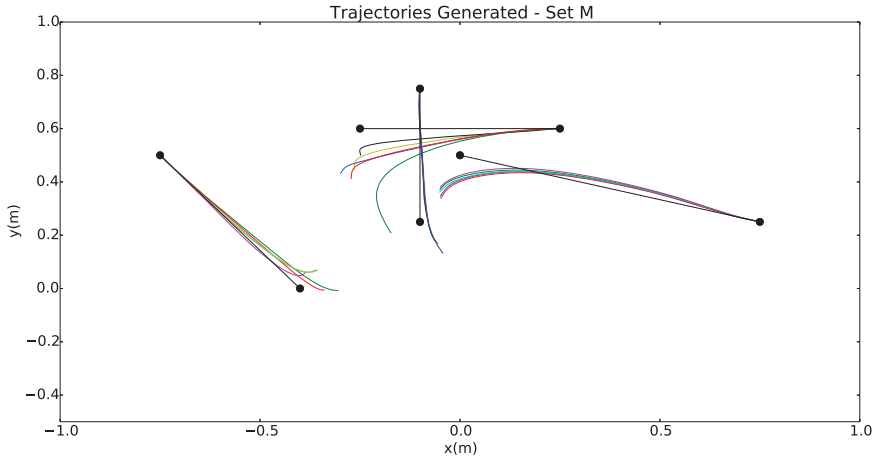


FIGURE 3.35: Generated trajectories during the testing phase by the Set M - Experiment Group 2. The original trajectories are depicted by the lines with filled circles. Five trials for each trajectory (from the Figure 3.4) are displayed using different line colours.

3.4 Conclusions

This chapter presented a new implementation of a 2 DOF arm controller based on the work presented in [26], but with the exclusive use of Python in order to make it easier to deploy in different platforms. Additionally, the Brian Simulator was extended in order to facilitate the interaction between the arm model and the spiking neural network in a step-by-step way.

Novel experiments to test the robustness of the controller in relation to the trajectory generation task were developed and executed. The results had confirmed what is stated in [24] since the increase in the noise levels led to a deterioration of the controllers ability to follow the trajectories. However, the behaviour of the LSM to the increase of noise was stable following a graceful degradation as seen in biological systems. In addition, the experiments decimating the liquid could be seen as a hint that it is not necessary to have plasticity for graceful degradation.

According to some of the experiments that took place in this work, the LSM with feedback from its own outputs behaves as a central pattern generator (see Chapter 2). The curves generated in the readout output are naturally synchronized because the liquid has internal connections and the readout reads the outputs from all the neurons for each joint value generated, hence it is influenced by all the activity inside the liquid.

The system always receives a zero torque value at the beginning as the LSM first output is always noisy without any correlation to the input commands. Consequently, the simple act of finding the right direction is quite an achievement since the system works by the prediction of the next values based on the liquid ability to integrate current and past inputs. This could justify the reason for the poor generalisation.

Experiments based on the trajectory generation tasks showed the use of a model where Short-Term Plasticity (STP) was implemented had no clear effect in the task results. This is an important outcome as STP is a computationally expensive part of the Liquid State Machine traditional implementation [21] making it possible to increase the number of neurons or decrease the time each simulation step takes to be computed.

Despite being able to learn four trajectories and partially reproduce them, the experiments that were performed in this chapter did not show a good generalisation ability when novel endpoint positions are tried. However, biologically inspired robotic systems that are supposed to learn by demonstration, i.e. learn how to reproduce a given task, are still useful without being capable to generalise to unknown endpoint positions.

Following the results obtained in this chapter and the lessons learned, the focus of this thesis shifted from trying to create a system that would generate point-to-point segments to modelling a biologically inspired system capable to learn by demonstration how to reproduce more complex movements. In addition, for certain tasks (e.g. work in nuclear disasters), robustness against external factors is a very valuable factor for robot controllers. Therefore, starting in the next chapter, this shift begins with the extension of the system presented here using the humanoid robot Baxter.

Chapter 4

Controlling Baxter Robot using a Liquid State Machine

4.1 Introduction

As a novel approach for the implementation of an LSM from Chapter 3, a 4 DOF arm controller along with a pilot experiment is presented in this chapter. While Chapter 3 presented a new implementation of a 2 DOF robot arm LSM controller based on Cartesian positions, here, instead of passing initial and final Cartesian positions as set points to the controller, the LSM only receives the first joint angle values and devises, in a feedback loop, all the necessary information to generate the commanded trajectory behaving more like a Central Pattern Generator (CPG) in this situation. With this approach, the generated patterns could be used in a future extension to produce more complex movements following, for example, the ideas introduced by [19]. The schematic representation of the controller implemented in this chapter can be seen in the Figure 4.1.

The LSM from Chapter 3 was controlling a robot arm that actually doesn't exist. However, in this chapter the research version of the Baxter Robot (from Rethink Robotics Inc.) was used to perform the same trajectories in a simulated environment (see Section 4.2.4). The new task is a real world one: drawing lines on top of a table. Following the ideas of Action Learning (Section 2.2), the tasks developed by the robot in the experiments presented in

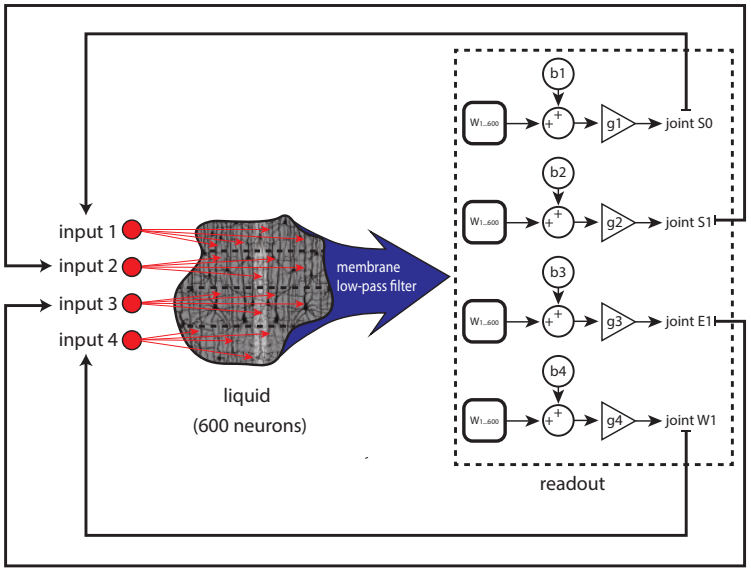


FIGURE 4.1: Illustrative representation of the trajectory generator implemented. Joint angles are fed to the LSM and the readout generates the correct joint angles for the next position. Inputs from 1 to 4 are the initial joint angles. Constants b1, b2, b3 and b4 are bias and g1, g2, g3 and g4 gains applied to facilitate the learn.

this chapter can be presented to Baxter using its special touch sensitive cuffs, enabling the user to directly teach movements to a real robot.

In this chapter, an LSM system capable to control 4 DOF of a humanoid robot arm (Baxter's arm), mimicking the behaviour of a central pattern generator, is presented. The controller will use the same four trajectories from Chapter 3, but rescaled to fit virtual environment used here. All trajectories are executed on top of a table where the robot uses a simulated felt pen to draw the result of the movements. Baxter's arm has a total of 7 joints, but only the joints S0, S1, E1 and W1 (Figure 2.4) are necessary in order to keep a 90 degree angle between the table and the pen while the drawings are created. The generated trajectories will be compared with the ones from Chapter 3 in order to verify the evolution of the error during their generation. The Virtual Robot Experimentation Platform (V-REP) simulated Baxter robot will be used for the the results presented.

4.2 Methods

In this section, all the procedures necessary to execute the experiments developed will be presented. Some of the techniques were already explained in Chapter 3 and references will be used, instead, for those situations.

4.2.1 Trajectories

In order to enable some comparison with the experiments presented in Chapter 3, the same four trajectories from Figure 3.4 were used. Although, to fit the new simulated space, the trajectories had their Cartesian position values divided by three. Baxter robot left arm was chosen to generate the trajectories during the experiments. The resultant approximated trajectories (Figure 4.2) are:

- ***Trajectory 1*** - Start $\Rightarrow$ End positions: $(0.25, 0.08) \Rightarrow (0.00, 0.17)$
- ***Trajectory 2*** - Start $\Rightarrow$ End positions: $(0.08, 0.20) \Rightarrow (-0.08, 0.20)$
- ***Trajectory 3*** - Start $\Rightarrow$ End positions: $(-0.03, 0.25) \Rightarrow (-0.03, 0.08)$
- ***Trajectory 4*** - Start $\Rightarrow$ End positions: $(-0.25, 0.17) \Rightarrow (-0.13, 0.00)$

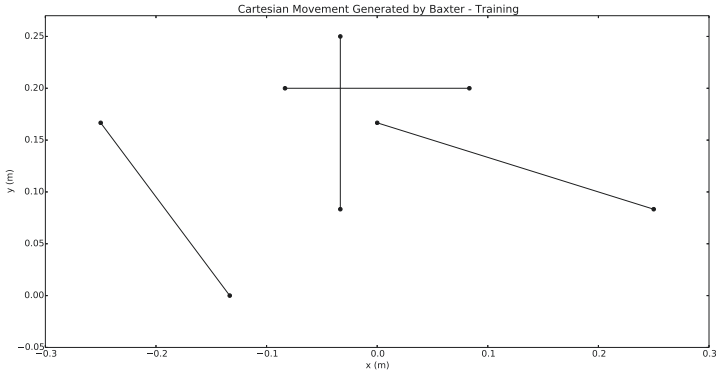


FIGURE 4.2: Trajectories used to train the LSM during the experiments with Baxter Robot.

4.2.2 Liquid State Machine Simulation

The LSM basic structure and parameters presented in Chapter 3 are employed here as well. For the 2 DOF arm, the system had a total of six input variables (Cartesian positions x_{dest}, y_{dest}, proprioceptive joint angles θ_1, θ_2 and joint torques τ_1 and τ_2) where 50 neurons where used to encode each variable using a total of 300 neurons for the input. In this chapter the LSM receives only four input variables. The joint angles S0, S1, E1 and W1 from Baxter's left arm have their values feedback to the LSM after the first simulation step.

As the results are going to be compared to the ones from Chapter 3, the total number of neurons used in the experiments are kept the same and the inputs here were still encoded using the 300 input neurons available.

Another difference between the two chapters is related to the input gain used. For the Chapter 3, the input gain was the same used in [26], but here an input gain mean value of $105nA$ was applied to increase the liquid's activity following some results during pilot experiments with the 4 DOF arm controller.

The most important change introduced in this chapter was in the readout as depicted in the Figure 4.1. When the trajectories are generated directly by controlling the system based on the original joint angles, the start values in most of the cases are different from zero. Also, the range of values have a bigger variation when compared with the experiments from Chapter 3 where joints are torque controlled. Since the output values from the liquid pass through a low-pass filter, it's impossible to generate a value that largely differs from zero at the first simulation step. Additionally, the variation at the final joint values for the four different trajectories make it harder to the readout to fit distributed weights that work smoothly for all situations. The solution found was to include a bias and a gain term between the readout and the output of the low-pass filter. Instead of only training the readout weights, a bias and a gain are calculated and used for each trajectory. The bias forces the LSM output values to start at zero (because of the low-pass membrane filter the system is not designed to generate starting values far from zero) and the gain normalizes the maximum value to one. That way, all trajectories always generates curves that start at

zero and go up to one. This idea was inspired by the signal conditioning techniques used in electronic instrumentation where analogue to digital converter normally have a range of values they can receive as input.

Moreover, as the experiments in Chapter 3 show the use of STP is not clearly necessary, here all the simulations are developed without the implementation of the dynamical synapses.

The LSM used in this chapter made use of two types of noisy offset currents (see Equation 2.1). One was normally distributed (i_{noise}) and the other uniformly distributed (i_{offset}). The i_{noise} default values were $\mu = 0$ and $\sigma = 1nA$ and i_{offset} default ones were from $13.5nA$ to $14.5nA$.

A summary of the parameters can be seen here:

- no STP.
- i_{noise} was made ten times smaller than default value.
- i_{offset} was made ten times smaller than default value.
- input gain mean value was made equal to $105nA$.
- all the four input variable received noise during training phase.
- each trajectory had calculated an unique bias and gain.

4.2.3 BRIAN Simulator

The SNN simulations used during the training and testing phase were all executed using BRIAN Simulator (for more details, see Section 2.3.6.1). The same set up and tools presented in Chapter 3 were applied for the experiments in this chapter.

4.2.4 Virtual Robot Experimentation Platform - V-REP

All the robot simulations done in this chapter were developed using the Baxter robot (Section 2.3.4) simulated by the Virtual Robot Experimentation Platform (V-REP) presented

in Section 2.3.5. Python was used, together with V-REP's remote API, to interact with it controlling Baxter. Using the simulator, the robot was capable of writing on top of a table (an equivalent setup can be seen in Figure 2.5).

4.2.5 Central Pattern Generator (CPG)

Central Pattern Generators are groups of neurons found in biological systems that are capable of producing basic patterns necessary to the composition of automatic movements. Examples of those movements seen in animals are locomotion, respiration, swallowing and defence reactions [124]. Another important characteristic in those neural oscillators is the absence of external feedback.

The CPGs are supposed to generate rhythmic activity without any rhythmic inputs, but only simple signals. Also the motor cortex, cerebellum, and basal ganglia are considered as high-level controllers and said to be responsible for the pattern interlocking in agreement with the external senses [125].

The authors in [126] suggest that both arms and legs are regulated by CPGs. The sensory feedback regulates the CPG activity, as in the inter-limb coordination, but it is not necessary for the primary pattern generation. CPGs are modulated by sensory input and brain commands in such a way to cooperate dynamically with the animal's environment and can achieve high level goals [29].

4.2.6 Experimental set-up

For the experimental set-up used here, the trajectory generation presented in Chapter 3 (2 DOF arm) was adapted to a real world situation where a line is drawn on top of a table using a felt pen. The task demands that the pen is kept at 90 degrees from the table and the height is held the same until the end of the trajectory (3D trajectory). The arm was controlled by angle values using the joints S0, S1, E1 and W1 (Figure 2.4).

In Chapter 3, the arm model had a bigger role in the computation because its reactions were dependent on the physics involved (it could be seen as a simple kind of morphological

computation [127]). One positive characteristic of that situation was a natural low-pass filtering of the commands received from the LSM making the movements smoother than they would be. But it also had an integrator effect summing the error during all the trajectory. As explained in the preceding sections, the only way to control Baxter keeping all the software safety procedures is by using joint angles. This control mode is known in the Baxter software development kit as Joint Position Control Mode (Figure 4.3). In this control mode, the robot's internal controller receives the angle set point and converts it to motor torques necessary to reach and keep that position. Therefore, here the LSM controls the robot by only sending angle values.

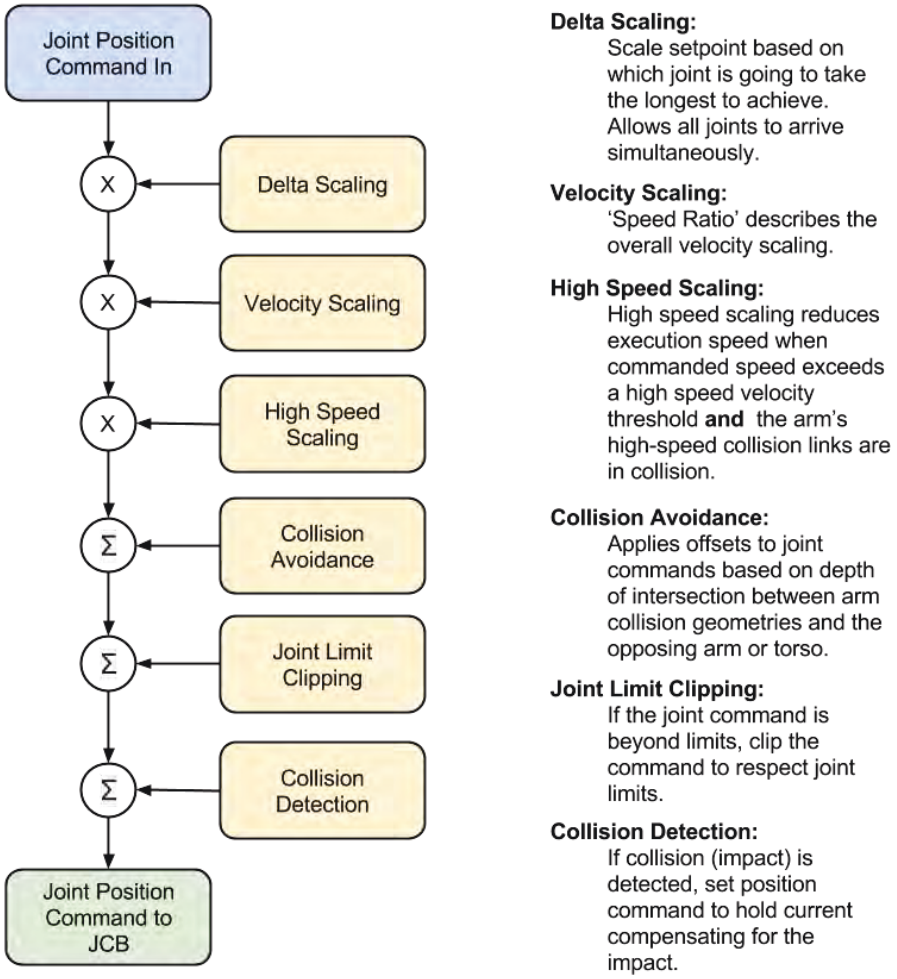


FIGURE 4.3: Chart explaining how the Joint Position Mode works internally in the software development kit. JCB stands for Joint Controller Boards - the internal motor actuators (from sdk.rethinkrobotics.com).

The training set could be generated directly by the user giving directions to the robot's arm. In the case of Baxter, it has special sensors in its wrists that enable a mode called Zero-G where the arm can be moved around almost without opposing the torques applied by the user's hand. Using the simulated robot inside V-REP, the Zero-G mode role was replaced by calculating the inverse kinematics using a special V-REP object called *dummy*. With the use of this set-up, it was possible to drag (move) the dummy object attached to the robot arm to record a new trajectory. During the experiments presented in this chapter, end effector Cartesian points were sent from an IPython notebook using the V-REP remote API.

During the training phase, 200 trials were used for each trajectory from Figure 4.2 totalling 800 simulations. All the data collected was used to generate the readout weights as well the bias and gain values.

The testing phase was composed of 10 trials for each trajectory. The input variables didn't receive noise in this situation, but the offset noisy currents (Equation 2.1) were kept with the same values as in the training phase.

As the controller only receives the initial joint angle values, without any information about the final position, generalization experiments were not tried. Therefore, in the next section only results related to the central pattern generator behaviour will be tested and the trajectories compared against the best result from Chapter 3.

4.3 Results and Discussion

The experiments with the 4 DOF LSM arm controller started with the analysis of the joint curves. The result of the ten trials were plotted against the original ones from the training set. Figure 4.4 presents the curves of all the trajectories from Figure 4.2.

The results from Figure 4.4 show that in most of the cases the LSM was able to follow the joint curve very closely. Only in the second row, where the trajectory 2 curves are shown, the LSM had a constant bias in the form of a time delay for all joints, but as the bias appears for all joints, it would appear in the final generated line as a shift in time.

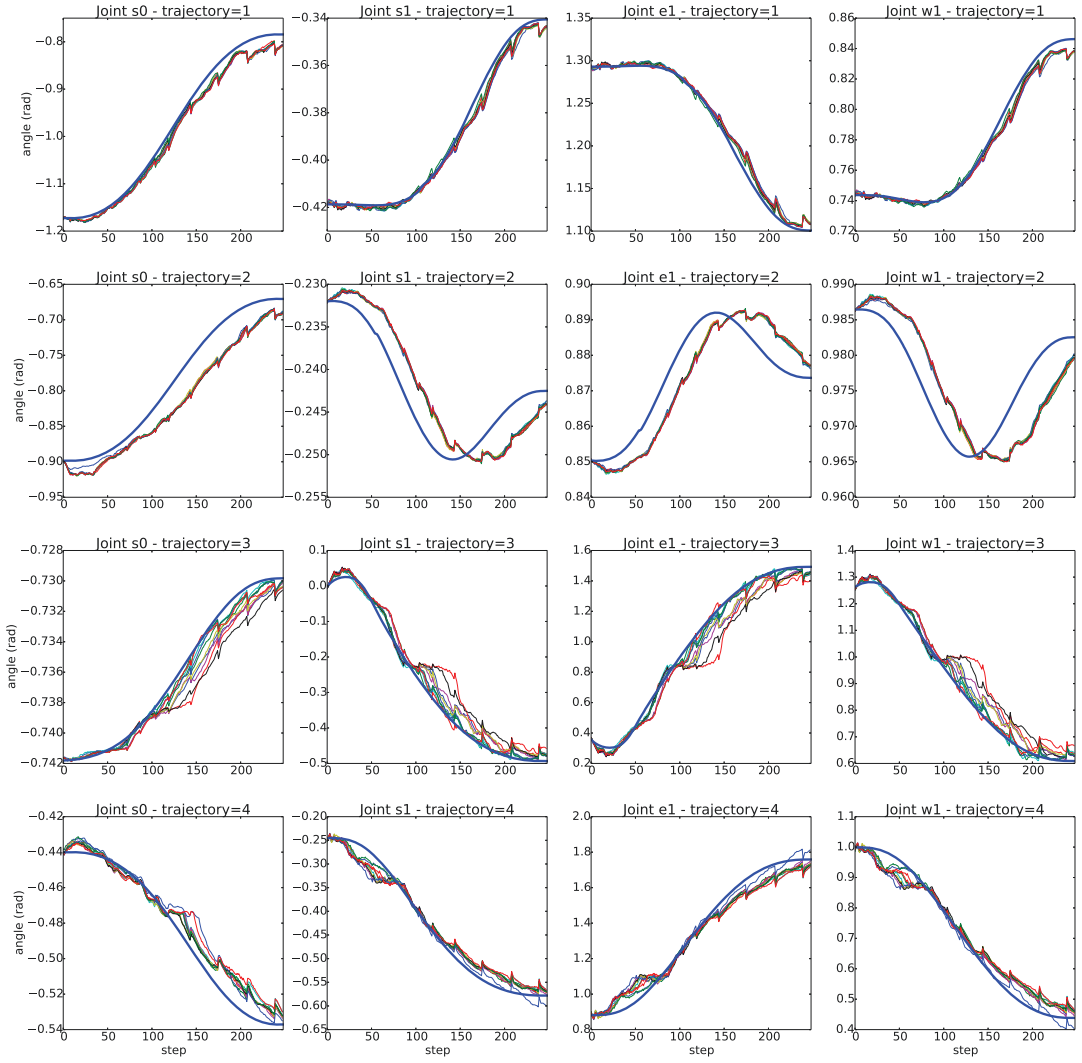


FIGURE 4.4: Joint angle curves resulted from the testing phase (multicoloured lines) are plotted against the training set (blue lines). Each row represents a different trajectory.

Trajectories generated during the testing trials clearly show that the error is not adding up in this situation because even with jerky movements in the middle of the trajectories the system is able to recover (Figure 4.5. Looking back to the trajectories generated in Chapter 3, Figure 3.17, the lines always start at the correct position (as the torque starts from zero and the arm needs to be positioned manually at the start of the trajectory) and along the trajectory they deviate.

The behaviour of the error during the trajectory is better understood with the help of Figure 4.6. As the times passes during the simulation the error does not accumulate. In the case of trajectory 1, it presents almost the same initial and final value. If the Figures 4.5

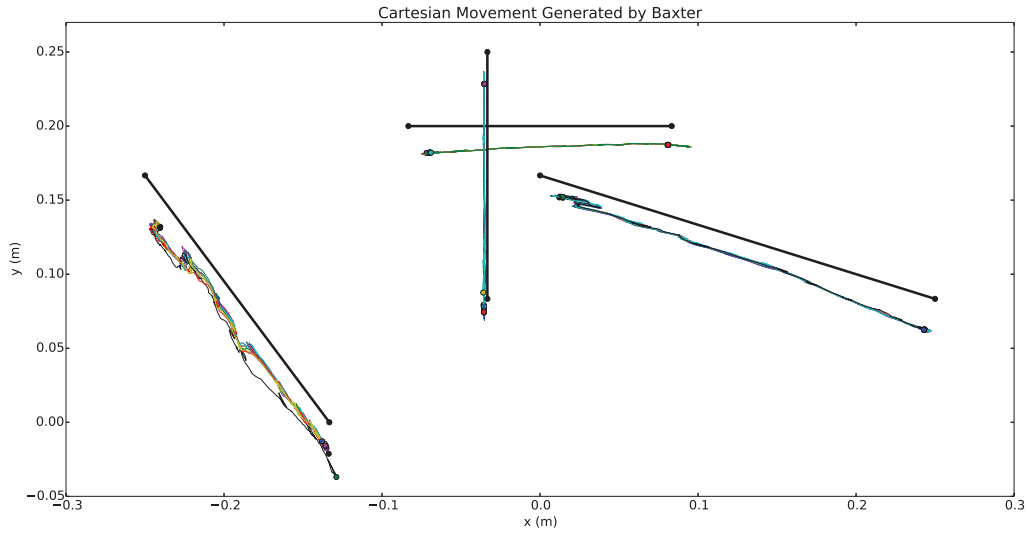


FIGURE 4.5: Original trajectories (Figure 4.2) are shown in thick black lines while the result from the 10 testing trials have multicoloured lines.

and 3.19 are compared, it is easy to see the steady increase in the error when the LSM controller from Chapter 3 was used.

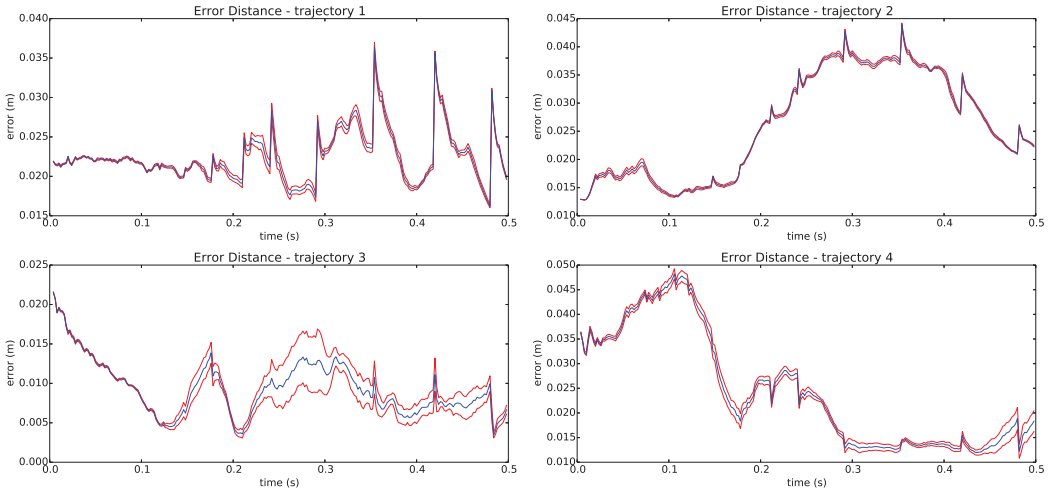


FIGURE 4.6: Evolution of the error (distance) between the original trajectory and the one generated by the LSM during 10 trials. Standard error is shown in red lines while mean value in blue.

4.4 Conclusions

This chapter presented a novel implementation of a humanoid robot arm controller based on a biological inspired neural network. For the first time, according to the author's knowledge, the Liquid State Machine framework was used to control a robot arm with more than 2 degrees-of-freedom. On top of that, for the first time the humanoid Baxter Robot had one of its arms controlled by a spiking neural network.

The results show that the system robustly generated trajectories with a good repeatability as could be seen in the results of ten trials presented in the Figure 4.5. Moreover, compared to the results from the experiments in Chapter 3 and the work presented in [26], here the error did not accumulate as the trajectory was generated.

Also, the controller developed here was able to generate four simultaneous time-series (Figure 4.4) receiving only the four initial angle values of the joints. It was able to generate those curves for all the four different trajectories. These results, alone, are of great interest as they could be extrapolated to other fields as biomimetic reproduction of central pattern generators or even new neural based computers considering the trajectories together with the joint curves as the result of a computer instruction and the initial joint angles, bias and weights as the calling method.

Besides promising results, the robot was still only learning how to reproduce simple straight lines. In addition, the simulator employed here was totally based on Python, an interpreted language, creating a bottleneck for testing more complex trajectories. The next chapter solves this problem by using a C based simulator (see Section 2.3.6.3) and a new approach to increase the computational power of the final implemented system by the use of ensembles.

Chapter 5

Improving Liquid State Machines Controllers by the use of Ensembles

5.1 Introduction

This chapter presents the experimental results of a novel humanoid robot control framework based on parallel, diverse (each liquid was randomly generated) and noisy (using random injected currents and initial membrane values), sets of biologically inspired LSM. Some of the motivations to apply the LSM paradigm are related to the idea that the neuron model and the inherent network connectivity could have an influence in the results of the computations as suggested in [128]. Also, the concepts of movement decomposition from [9, 18, 20] were used as inspiration for the parallel system as their results are averaged and composed together to generate the final movement.

The chosen task was based on the principles of action learning [34] focusing on the ability to learn from a teacher how to draw on top of a table. This is part of a wider approach to robot learning and development with embodied and situated interaction [6, 25, 35]. The trajectories start and end with zero velocity and acceleration following a smooth human inspired profile [117]. A total of four joints were necessary to draw the shapes (square, triangle and circle) while the distance and angle between the pen and the table had to

be kept constant and, therefore, the final task is much more complex than the drawings themselves.

Two different techniques, associated to the way LSMs are stacked together, were employed here: parallel and serial. Both are more extensively presented in the Section 5.2. The performance and analysis of the learned movements using the parallel and the serial approaches were done using a model of the Rethink Robotics Inc. industrial humanoid robot Baxter inside the Versatile and Scalable Robot Simulation Framework (V-REP, see Section 2.3.5). At the end of the chapter, as a proof of concept, one of the shapes was also tested using the real Baxter robot and the results presented.

5.2 Methods

This section details the procedures adopted in this chapter to analyse the parallel system (Section 5.2.1) formed my multiple LSM (Section 5.2.2), introduced here, using a robotic experiment[1].

5.2.1 Parallel and Serial Approaches

When working with a stochastic system, as in the case of the LSM approach, each time an experiment is done a new unique value will be generated. Considering the random process as stationary, traditionally an averaging of the results from multiple simulations, i.e. the mean value or first moment, is used.

In the work of Maass et al. [24], LSM systems with an added feedback connecting the output to the input were employed in order to increase their computational power. Each LSM received individual feedback and its outputs were averaged between trials to improve the results. This method is called here the *serial* approach and a diagram is depicted in Figure 5.1.

[1] Source code available at github.com/ricardodeazambuja/IJCNN2016

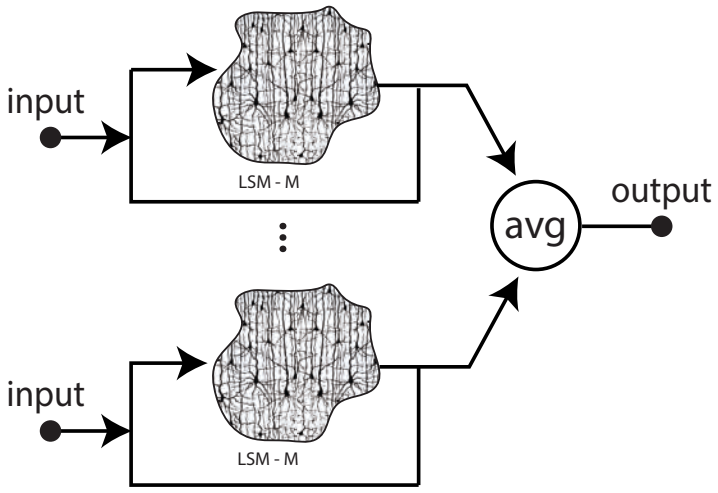


FIGURE 5.1: Simplified diagram for the *serial* approach. Contrasting with the parallel system (Figure 5.2), here each LSM receives its own output as feedback instead of the averaged values received from the robot. All LSMs were randomly initialised.

The alternative framework proposed in this chapter follows a slightly different approach. Inspired by the idea of a parallel noisy brain model, multiple LSM, randomly created and initialized, are used in parallel and the feedback each individual LSM receives is the average of all the readout outputs. A diagram of the *parallel* method is presented in Figure 5.2.

5.2.2 Liquid State Machine

The *parallel* approach (Figure 5.2) is compared to the *serial* one (Figure 5.1) through a robotic task. Both systems employed during the experiments had the same number of LSMs (with 600 artificial neurons each, totalling $3,000$ per trial) in order to allow a comparison between them.

An LSM usually is composed of Leaky Integrate-and-Fire (LIF) spiking neurons [13] connected in a recurrent pattern as suggested in [21] forming what is known as Small-World Network (SWN). The authors of [97] suggest SWN presents an appealing way to model the brain connections based on empirical and theoretical motivations.

Non-spiking neuron model based applications traditionally do not alter the network after the learning process is finished. Noise is only applied during the initialization as stated by [25]. However, stochastic processes seem to be an important part of brain computational

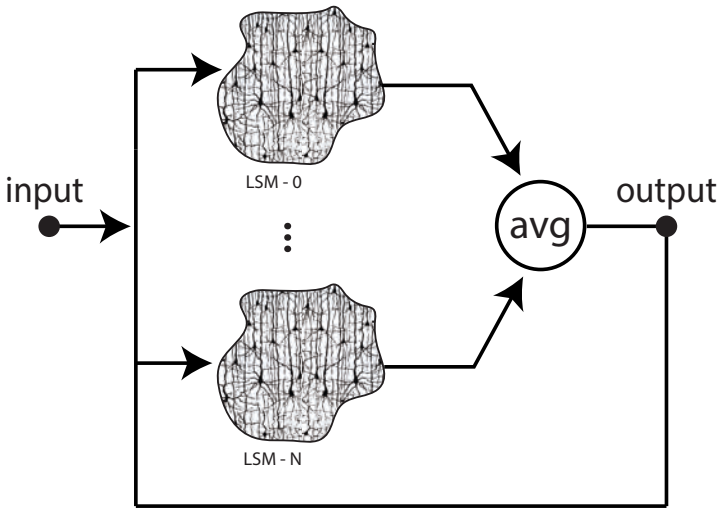


FIGURE 5.2: Simplified diagram for the *parallel* approach. Complete LSMs are arranged in parallel and their outputs (joint angles) are averaged before sending them to the robot. Additionally, the values fed back to the input are the ones read from the robot encoders. All LSMs were randomly generated and initialised.

strategy [28] and the LSM technique implements noise levels compatible with what was found in *in vivo* recordings [12].

In order to make it possible to benchmark the performance of the framework proposed here, the same neuron model and LSM parameters from [26] were applied with a few modifications. The system implemented in this chapter does not make use of Short-Term Plasticity (STP) or forced transmission delays. According to [12], STP is only one of the properties generating hidden network states. Therefore, as they slow down simulations by demanding extra variables and calculations, STP and delays were not used. The diagram of the implementation of one individual LSM can be seen in Figure 5.3.

A variant of the Leaky Integrate and Fire (LIF) neuron model with exponential currents is used in this chapter (Equation 2.1). The basic LIF model behaves as a capacitor-resistor circuit with an added circuitry in order to generate the spike (action potential) and also to keep it discharged during the refractory period [13]. It can be partially represented by the set of differential equations as seen in the Equation 2.1 where c_m is the membrane capacitance (in F), τ_m the membrane time constant (in s), τ_{syn_e} and τ_{syn_i} decay time of the excitatory and inhibitory synaptic current respectively (in s), v_{rest} the membrane resting potential (in V), i_{offset} a fixed noisy current and i_{noise} a variable noisy current (in A).

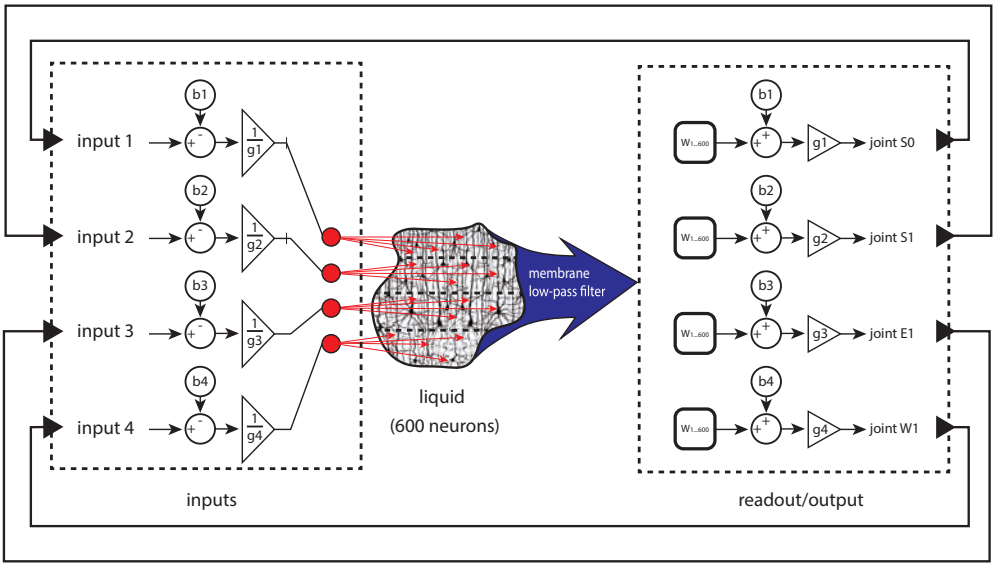


FIGURE 5.3: Illustrative representation of one individual LSM using only its own feedback. Each joint input/output has its unique bias (b1 to b4) and gain (g1 to g4). Readout weights sets (W1 to W600) are trained individually for each joint before the biases and gains are applied. As the weights are directly connected to the output of the low-pass membrane filter, they are trained using the normalised values (see Section 5.2.4 and Figure 5.6).

All simulations employed IPython and a custom software entirely written in C for the SNN simulations (see Section 2.3.6.3 for more details). The parameters used for the neuron model were: $\tau_{syn_i} = 6ms$, $\tau_{syn_e} = 3ms$. $c_m = 30nF$, $\tau_m = 30ms$. Each LSM had the i_{offset} randomly drawn (see [26] for details about the distributions) during its creation, but the values were kept constant (by the use of the same random seed) after that. The liquid's internal structure (connections) was also kept after the initialisation. It was necessary to keep those values otherwise each trial would have a different liquid instead of one with added noise. The initial membrane voltage and the current i_{noise} were randomly drawn during learning and testing phases where new i_{noise} values were drawn every time step. For all simulations, a time step of $2ms$ was used.

5.2.3 Definition of the $2D$ shapes

Three shapes were used in this chapter as a teaching task for the robot: square, triangle and circle (Figure 5.4). The joint angles generated by the inverse kinematics to draw a

triangle on top of the table are presented in Figure 5.5. The system needs to obey not only the individual joint curves but also the synchrony between them to accomplish its task.

Based on the human inspired model reported by [117], the velocity profile was not constant all over the trajectories and the effect can be seen by the concentration of points in Figure 5.4. A time step of $2ms$ was adopted with the whole trajectory taking $2s$. Shapes containing sharp bends (square and triangle) were designed using several straight trajectories where the velocity reached zero at the corners. The idea here was to simulate a human teacher guiding the robot's arm.

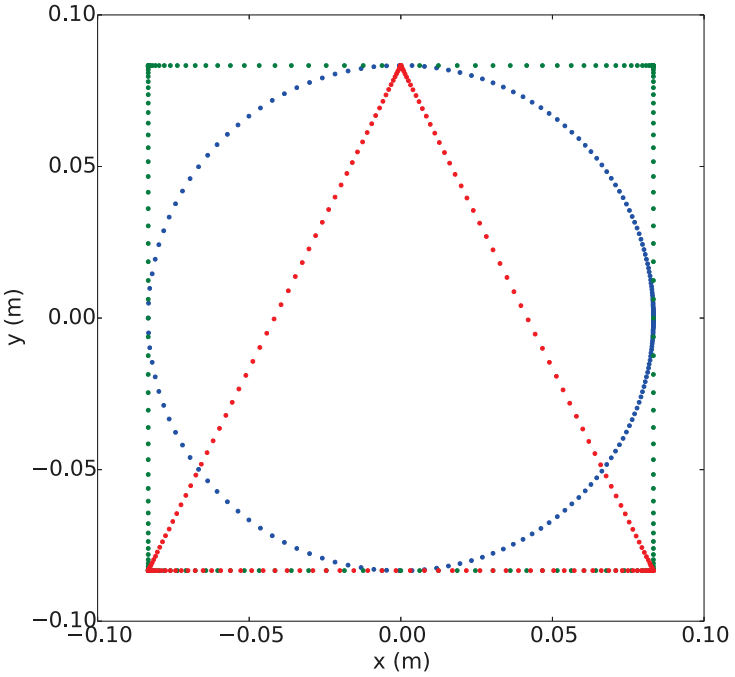


FIGURE 5.4: Shapes used to teach the robot (green:square, blue:circle and red:triangle). The Z axis is not presented here as it was kept constant (non zero) for all the shapes. The increase in the concentration of points is a consequence of the human inspired velocity profile [117]. Only one fifth of the points are presented to help the visualization.

5.2.4 Input and Output Code

The input code uses a simplified population code inspired by what was presented in [26] to discretize analogue values. A VLSI friendly implementation of an LSM based system, where discrete inputs were used as well, was presented in [129]. Therefore, this setup is appealing for future conversion of LSMs to a VLSI digital binary based system. Despite its simplicity, it needs a large number of neurons if a fine scale is necessary.

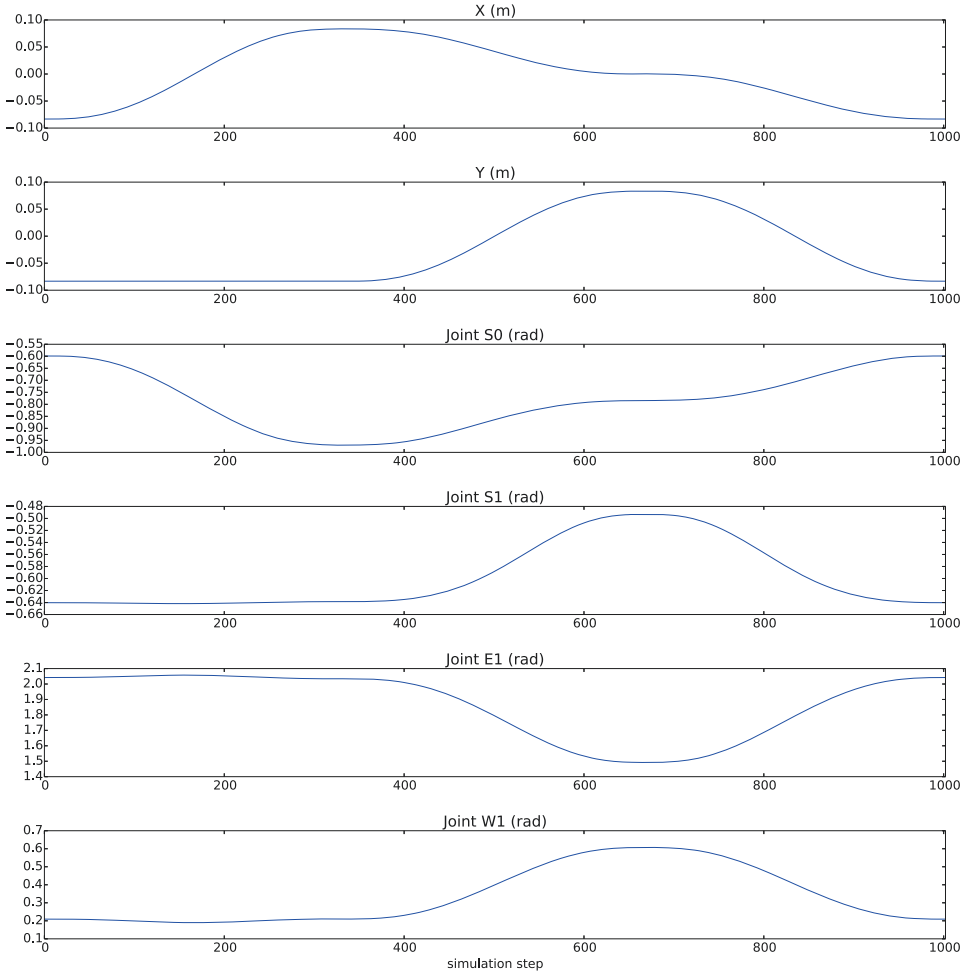


FIGURE 5.5: Resultant individual Cartesian X and Y movements as well as joints S0, S1, E1 and W1 (see Figure 2.4) necessary to draw the triangle (Figure 5.4) on top of a table.

Inspired by electronics instrumentation signal conditioning, here analogue values suffer a translation (*bias*) making them start at zero and compression/elongation through a *gain* that fits its total range to one (normalisation, see Figure 5.3). Using this method, and with only two extra variables (*bias* and *gain*), the system normalises the inputs and uses as much of the population code range as possible producing a higher resolution (Figure 5.6) than a system that does not make use of the normalisation presented here.

The pseudo-code used to convert from analogue values to neuron indices is presented as Algorithm 1. Both *bias* and *gain* are unique for each shape and could be seem as a form of mapping from the desk space (where the shapes were drawn) to the LSM space.

For the injection of the input spikes, the liquid was divided into four slices. Each slice

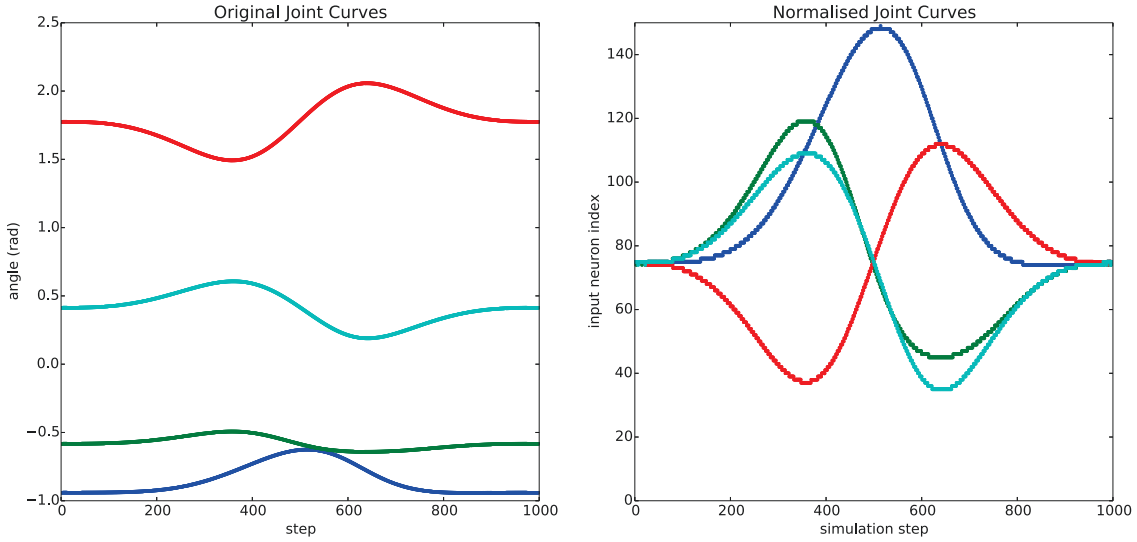


FIGURE 5.6: Input normalisation example. The figure on the right presents the normalised version of the joint angle values necessary to draw the triangle on top of the table (figure on the left). The normalised values go from 0 to 149 as they are related to the input neuron index offset.

receives spikes from only one input (joint angles). The weights connecting the liquid's neurons and the input spikes are formatted to create a redundant code with a Gaussian shape where its mean value lays on the input neuron index (same idea was already presented in Chapter 3, Figure 3.7).

Algorithm 1 Input code normalisation

procedure INPUTNORM($J\theta$)
  Receives joint angle $J\theta$ ▷ $J\theta$ in *rad*
  Extracts the *bias* ▷ the curve starts from zero now.
  Divides by the *gain* ▷ *gain* makes the range unitary
  Discretizes ▷ closest value on the input neuron array
  return *index* ▷ value passed to the Gaussian input
end procedure

The output joint values use the same *gain* and *bias* calculated for the inputs. It works as the opposite calculation of the Algorithm 1, but having as inputs the analog values from the membrane low-pass filter (time constant $30ms$) output.

5.2.5 Linear regression

Reservoir computing systems, as the LSM, need a readout to extract, or translate, information from the liquid (reservoir) to the output. The standard method is the use of a

linear regression. In this chapter, the weights are represented by variables W1 to W600 (see Figure 5.3) connecting the *readout* to the liquid through the membrane filter. Previous chapters employed the Ordinary Least Squares (OLS) as this method solves the minimization problem expressed by the Equation 3.1.

The Ridge Regression [130] solves a slightly different version of the OLS problem as can be seen on Equation 5.1 (λ is the regularization parameter). Differently from the OLS, the Ridge Regression is still solvable even if $\mathbf{X}$ is not full rank. The implementation from Scikit-learn (0.16.1) [119] was used here with its default settings.

$$\min_{w} \left(\frac{1}{2} \|y - \mathbf{X}w\|^2 + \frac{\lambda}{2} \|w\|^2 \right) \tag{5.1}$$

In this chapter, the matrix $\mathbf{X}$ was composed by the membrane low-pass filtered values of the liquid spikes ($\tau_m = 30ms$), y the normalised joint angle values (before the *bias* and *gain* being added back) necessary to generate the shape and w represents the readout weights.

5.2.6 Baxter Robot

Since Baxter is a humanoid robot designed to be safe and operate among humans and, in order to keep all its safety mechanisms activated the researcher must use the Joint Position Control mode (see Figure 4.3), the experiments using the framework proposed here always command the robot using that control mode.

It is important to emphasize that although it is drawing $2D$ shapes, the robot moves in the $3D$ space, keeping an angle of 90 degrees between the felt pen and the table, and it needs to keep the Z axis constant (table top height). This made necessary the use of four joints S0, S1, E1 and W1 as presented in Figure 2.4.

For the training phase, the translation from Cartesian space to joint space was made using the available Damped Resolution Method (damping:0.10 and max.iterations:3) for inverse kinematics in V-REP (version 3.2.2. rev.1) after the arms were positioned in their default *untucked* pose[2]. In addition, the V-REP remote API was employed to control the

[2]https://github.com/RethinkRobotics/sdk-docs/wiki/Tuck-Arms-Example

simulator from IPython making it easier to run several trials using the V-REP headless mode. Figure 2.5 presented the simulation result of one trial using the framework presented here.

As a proof-of-concept only, the square shape using the novel parallel framework proposed here (Figure 5.2) was tested using the real Baxter robot (Figure 5.28).

5.2.7 Testing and Analysis tools

An analysis was carried out to verify if the novel framework proposed here (Figure 5.2) would perform better or worse than a single LSM after several trials where the final results were averaged (Figure 5.1). Pilot experiments, not presented here, suggested that the positive effects of multiple LSMs in parallel could be more clearly seen with at least five of them together. Therefore, five different LSMs (600 neurons each) were randomly generated to test each individual shape. Here, the term *randomly* is employed in respect to the initialization of the random seeds used during the definition of the liquid structure's main parameters.

Readout weights were trained for each one of the five LSMs for a total of five hundred trials (one hundred trials each LSM). The testing phase employed all five LSMs created for each shape in a batch of ten trials for the proposed parallel framework (with five LSMs running in parallel - Figure 5.2) and ten trials for the traditional serial averaged one (where each trial was composed of five simulations of identical LSM with the results averaged at the end. In order to have the same number of final shapes, each LSM was employed twice during the simulations using the serial approach - Figure 5.1).

Having the same total number of trials for both systems and using the same five different LSMs generated made it easier to compare the two approaches. Instead of simulating only $2s$ (1000 simulation steps), during the testing phase the system was subjected to twice the number of steps. The use of more simulation steps helped to verify if the systems had the ability to keep the end position.

The experiments presented many results where the movement had a constant value zone creating a delay in time, and this makes it harder to apply traditional metrics such as a simple Euclidean distance. Instead, the data analysis was carried out applying the Dynamic Time Warping (DTW) (see Chapter 2, Section 2.3.7) to the time series obtained from the experiments together with visual inspections.

5.3 Results and Discussion

The main task proposed in this chapter as a benchmark was the ability to teach a robot, using principles of action learning [34], embodiment [6, 25] and controlled by a SNN to draw three $2D$ shapes (Figure 5.4). The analyses were made by comparing the resultant curves from the parallel and serial methods using the DTW method as well as visual inspection.

Since the task involved the control of a pen in the $3D$ Cartesian space with the added time dimension, and in order to simplify the figures, the comparisons were divided into the analysis of: $2D$ Shape (5.3.1), Time (5.3.2), Space-Time (5.3.3) and Z Axis (5.3.4).

5.3.1 Final $2D$ Shape Analysis

For the analysis of the resultant $2D$ shape, all the ten trials were plotted together with the parallel and serial approaches side-by-side. Figure 5.4 shows the square, circle and triangle shapes as green, blue and red dots, respectively.

Starting with the square, the comparison between the two methods, parallel and serial (Figures 5.7 and 5.8, respectively), showed the latest was not able to complete the task even after all the trials. As the square has four sides, it is easy to see the serial approach could not reach much more than one half of the total trajectory.

In the case of the triangle, the serial method (Figure 5.10) was able to draw a shape resembling the final triangle shape on only one out of ten trials. In spite of that, the serial approach still failed in the final $1/6th$ of its best trial confirming how unreliably the system

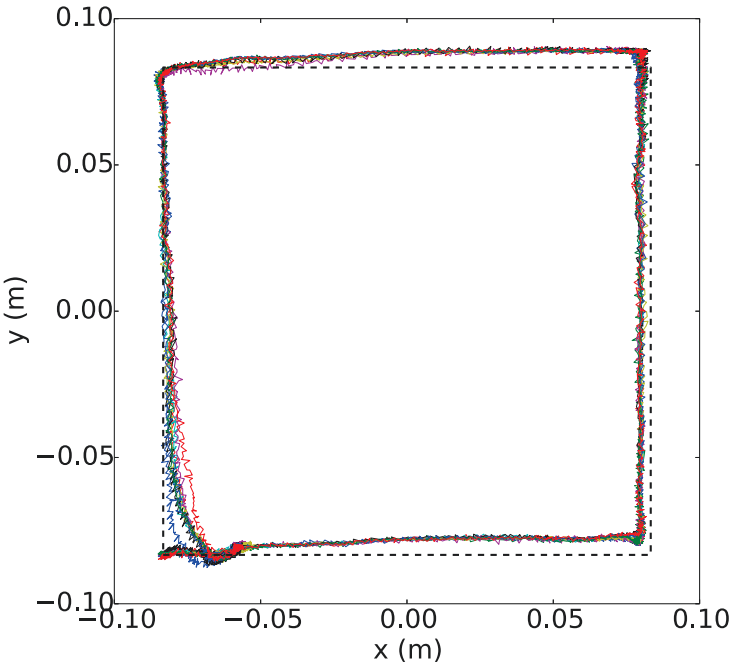


FIGURE 5.7: Results of the ten trials plotted overlaid in multiple colours (square). The dashed line represents the original shape (Figure 5.4) - using the framework presented here (*parallel* method).

behaved during the experiments. This strongly contrasts with the parallel method results (Figure 5.9) where for all trials the triangle shape almost matches the original one.

The circle was the only shape with results where the traditional serial approach (Figure 5.12) was able to follow the trajectory for more than one trial.

Although not a perfect fit to the dashed line (the original shape), the resultant curves from the parallel method presented here (Figure 5.11) were able to match the original ones with just some small errors when compared to the serial one. The only shape with slightly worse results for the parallel case was the circle.

5.3.2 Time Series Analysis

The teacher signal used in this chapter contained more than spatial information, as it also had time information in the form of a velocity profile (Section 5.2.3, Figure 5.5). Consequently, using only the analysis of the final $2D$ shape it is not possible to verify the behaviour in relation to time.

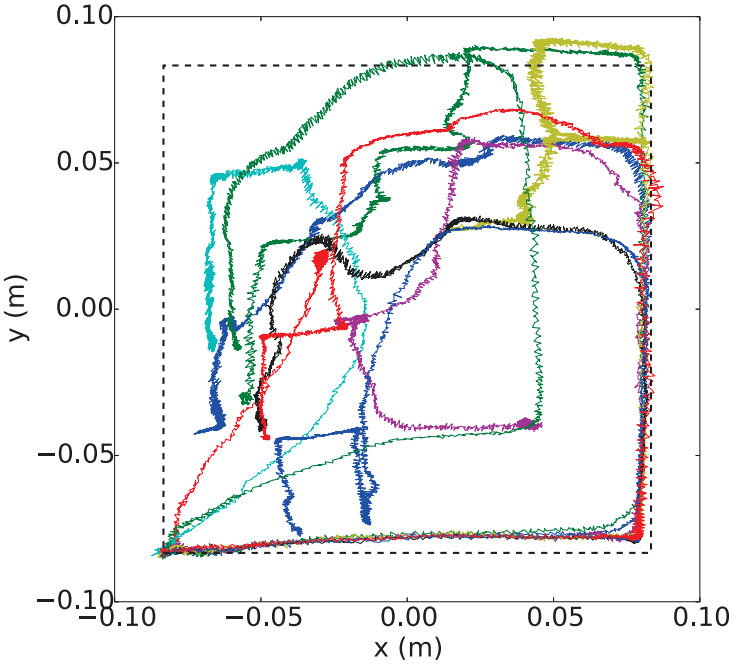


FIGURE 5.8: Results of the ten trials plotted overlaid in multiple colours (square). The dashed line represents the original shape (Figure 5.4) - using the *serial* averaged method.

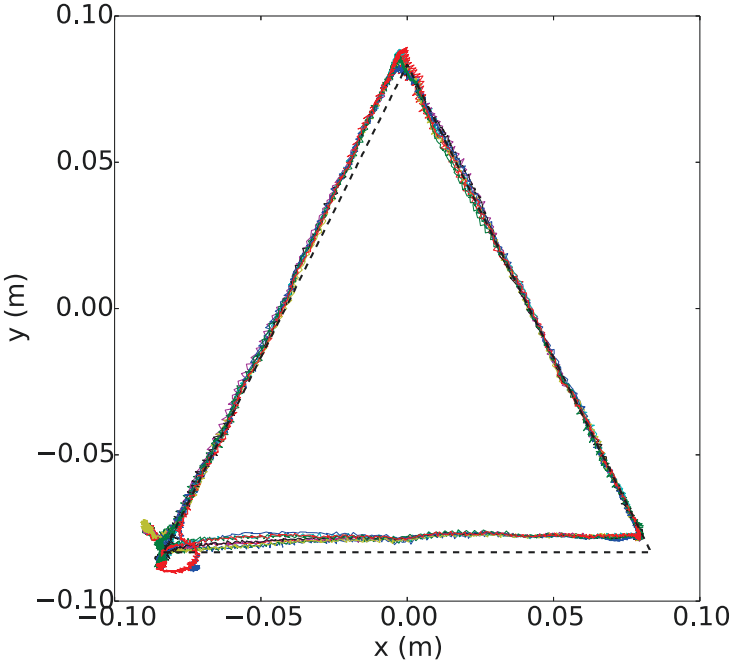


FIGURE 5.9: Results of the ten trials plotted overlaid in multiple colours (triangle). The dashed line represents the original shape (Figure 5.4) - using the framework presented here (*parallel* method).

Through a visual inspection of the Figures 5.13 and 5.14 (square), Figures 5.17 and 5.18 (triangle) and Figures 5.15 and 5.16 (circle) it is possible to realise that the solution proposed

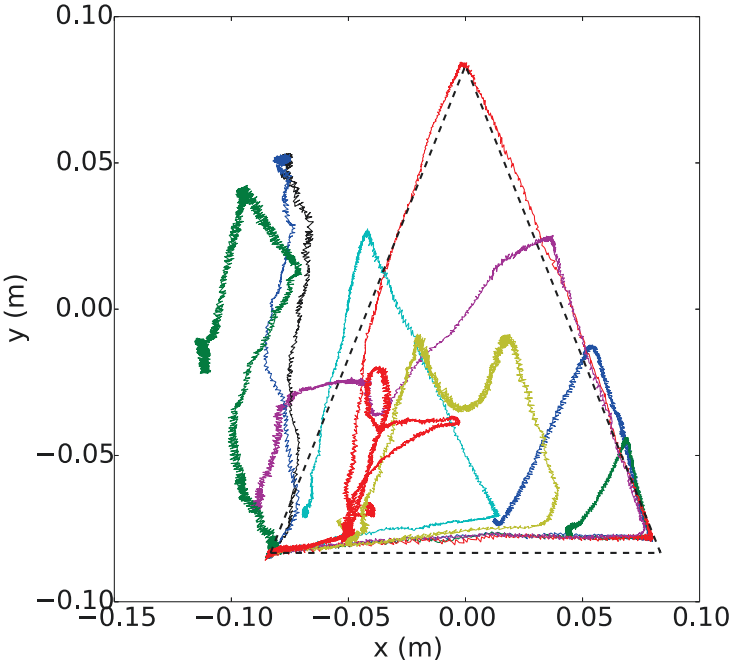


FIGURE 5.10: Results of the ten trials plotted overlaid in multiple colours (triangle). The dashed line represents the original shape (Figure 5.4) - using the *serial* averaged method.

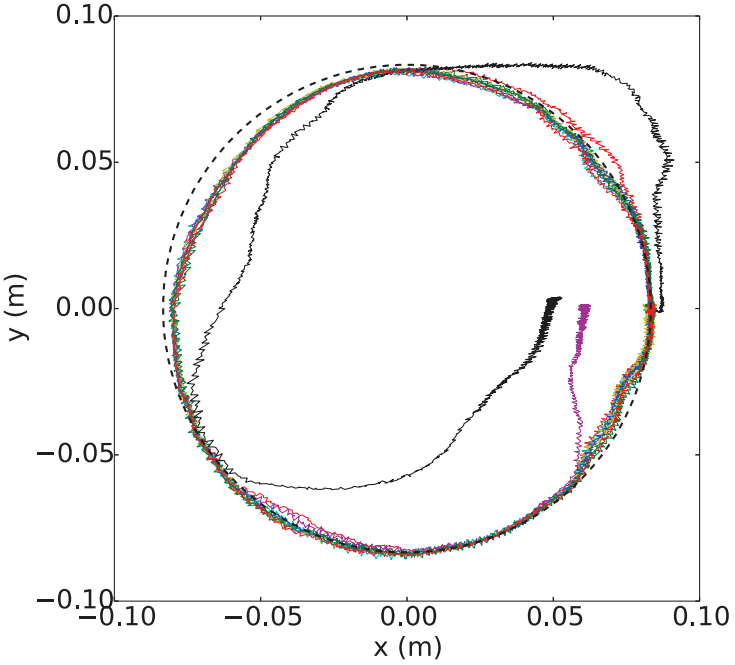


FIGURE 5.11: Results of the ten trials plotted overlaid in multiple colours (circle). The dashed line represents the original shape (Figure 5.4) - using the framework presented here (*parallel* method).

here was able to follow more closely the original X and Y time series.

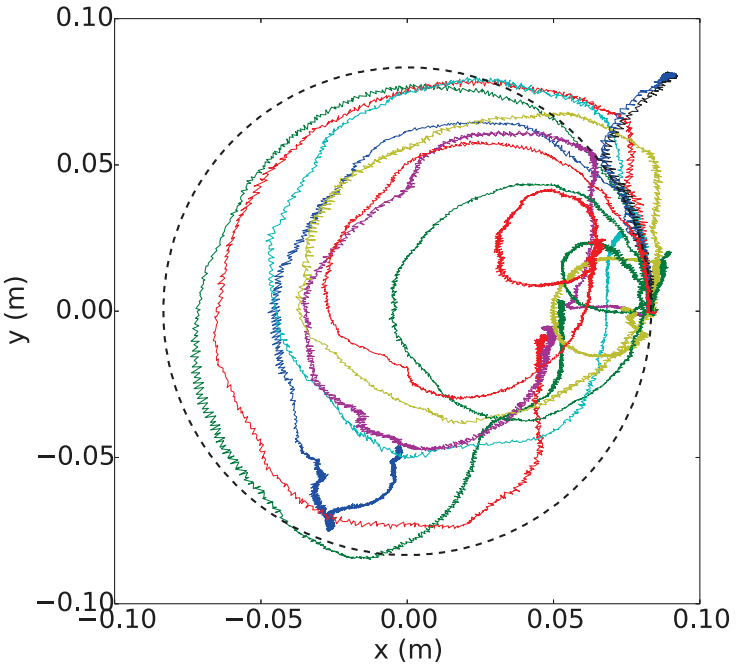


FIGURE 5.12: Results of the ten trials plotted overlaid in multiple colours (circle). The dashed line represents the original shape (Figure 5.4) - using the *serial* averaged method.

However, in some of the trials, phase delays or translations in time (represented by simulation step in the presented figures) occurred mostly within the parallel approach (Figures 5.13, 5.15 and 5.17). The serial approach (Figures 5.14, 5.16 and 5.18) clearly could not take advantage of the diversity of the LSMs as most of the curves had a very erratic shape.

5.3.3 Space-Time Analysis

During the time series analysis (Section 5.3.2), phase delays, disturbances and constant value zones were observed in some of the signals generated. The DTW algorithm was applied generating a final metric closer to what a visual inspection of the final $2D$ shapes could detect.

The results from the comparisons using the final cost generated by the DTW are presented in the Figures 5.19, 5.20 and 5.21 for the square, circle and triangle respectively. For each of the ten trials, the normalised cost (bottom) and the resultant shape (top) are depicted.

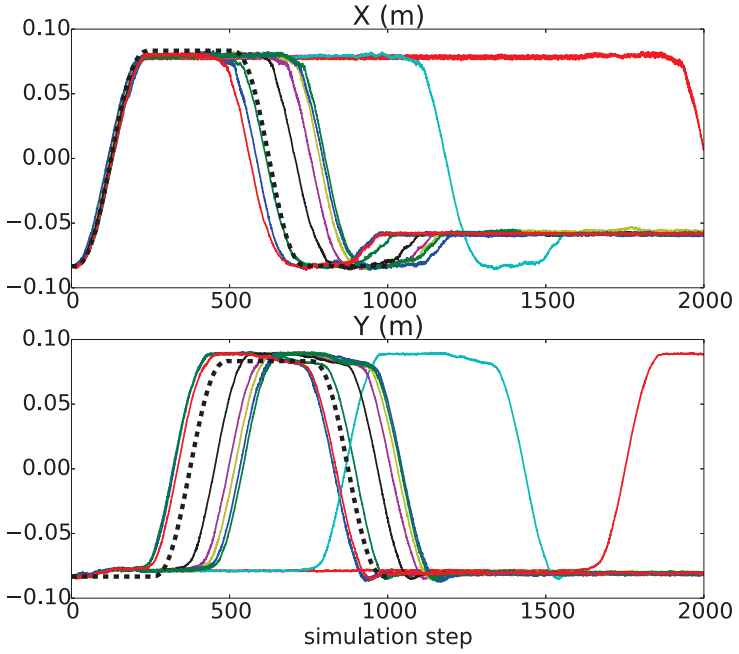


FIGURE 5.13: Visualization of the X and Y resultant curves (square) in time (all ten trials). The dashed line represents the original shape (Figure 5.4) - using the framework presented here (*parallel* method).

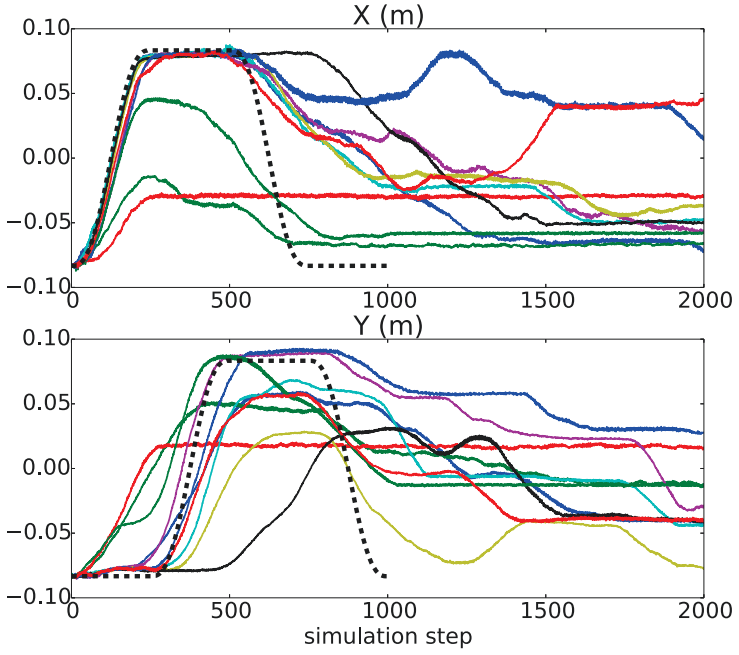


FIGURE 5.14: Visualization of the X and Y resultant curves (square) in time (all ten trials). The dashed line represents the original shape (Figure 5.4) - using the *serial* averaged method.

In only one situation (square shape, Figure 5.19, trial number 2) the serial method had approximately the same cost as the parallel one. This happened because the parallel

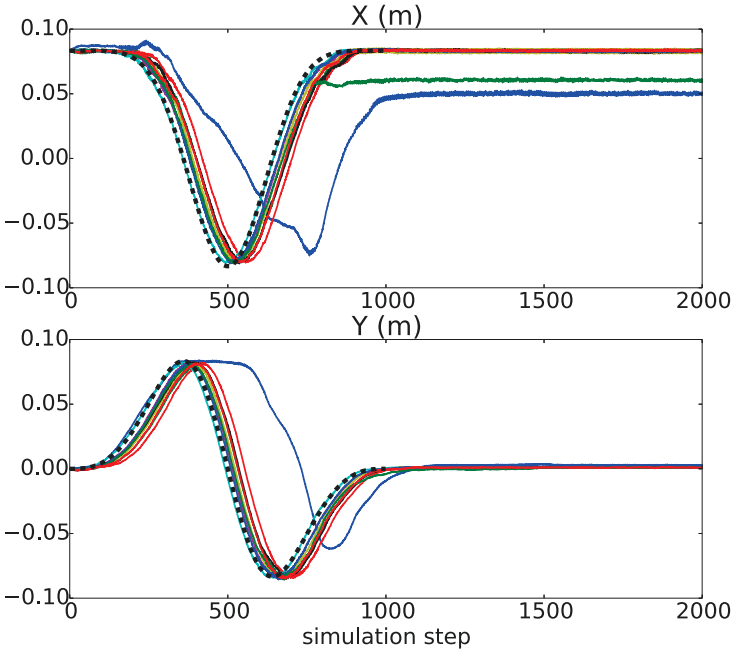


FIGURE 5.15: Visualization of the X and Y resultant curves (circle) in time (all ten trials). The dashed line represents the original shape (Figure 5.4) - using the framework presented here (*parallel* method).

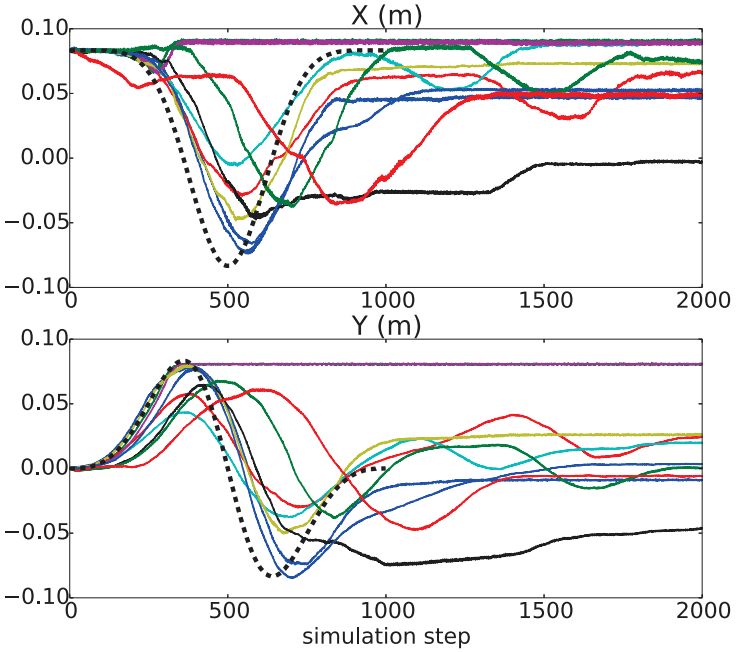


FIGURE 5.16: Visualization of the X and Y resultant curves (circle) in time (all ten trials). The dashed line represents the original shape (Figure 5.4) - using the *serial* averaged method.

method got stuck in a constant value (see Figure 5.13) and the simulation finished before it could get unstuck.

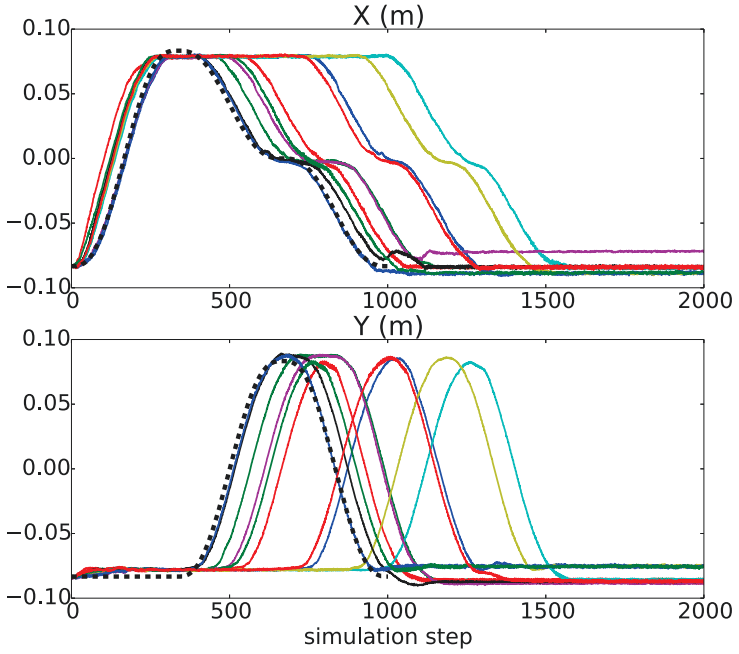


FIGURE 5.17: Visualization of the X and Y resultant curves (triangle) in time (all ten trials). The dashed line represents the original shape (Figure 5.4) - using the framework presented here (*parallel* method).

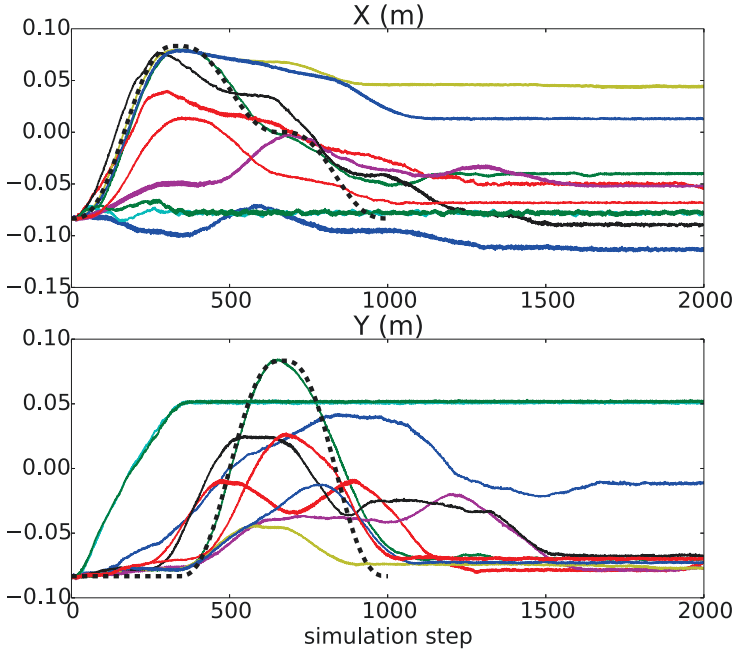


FIGURE 5.18: Visualization of the X and Y resultant curves (triangle) in time (all ten trials). The dashed line represents the original shape (Figure 5.4) - using the *serial* averaged method.

Besides the particular situations were the parallel system got stuck on an intermediate position, its cost was always smaller than the serial one. Therefore, the DTW method

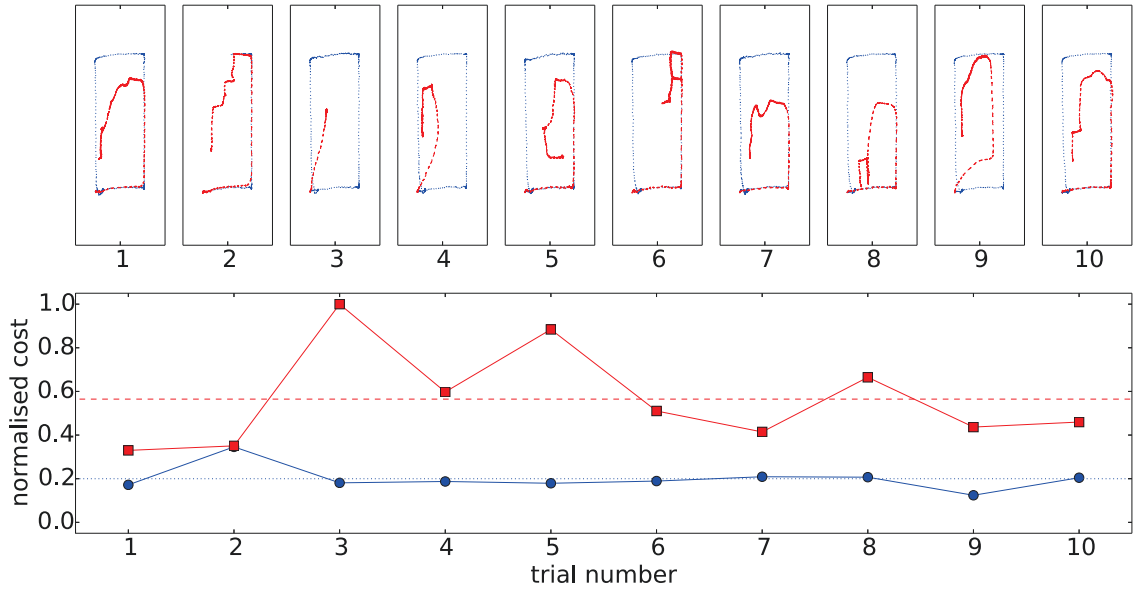


FIGURE 5.19: Resultant curves (square) for all trials - top figure. The DTW generated cost is presented at the bottom figure. The serial (dashed line, small squares - red) and parallel (dotted line, small circles - blue) approaches had a normalised mean cost value of 0.56 and 0.20 respectively.

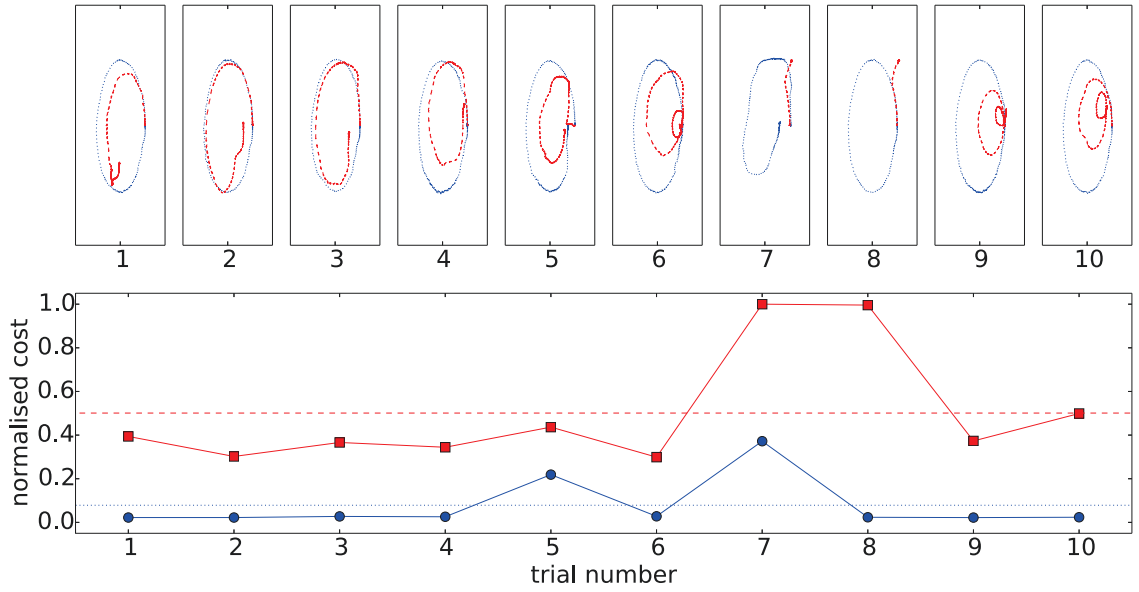


FIGURE 5.20: Resultant curves (circle) for all trials - top figure. The DTW generated cost is presented at the bottom figure. The serial (dashed line, small squares - red) and parallel (dotted line, small circles - blue) approaches had a normalised mean cost value of 0.50 and 0.08 respectively.

confirms what the visual inspection of the Figures 5.13, 5.14, 5.17, 5.18, 5.15 and 5.16 had already suggested and becomes clear the parallel method was able to control the simulated robot to generate better final shapes than the serial one.

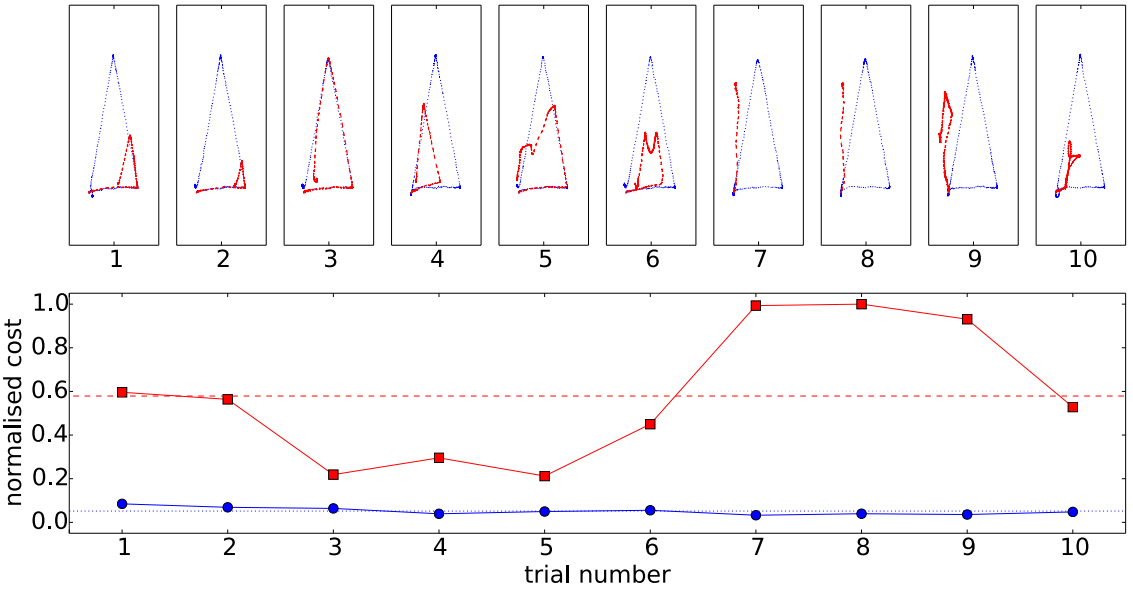


FIGURE 5.21: Resultant curves (triangle) for all trials - top figure. The DTW generated cost is presented at the bottom figure. The serial (dashed line, small squares - red) and parallel (dotted line, small circles - blue) approaches had a normalised mean cost value of 0.58 and 0.05 respectively.

5.3.4 Final Z Axis Analysis

During the simulations done in V-REP, the table did not exert any kind of reaction against the pen. Therefore, values where the Z height is below the table surface were generated. In order to draw the shapes, the Baxter robot should keep the Z height approximately constant.

The Figures 5.22 and 5.23 present the results obtained during the testing phase of the square shape for the Z axis. The maximum *delta* (absolute distance from the original Z value) for the parallel method was $1.12mm$ for the square, $2.10mm$ for the circle and $0.71mm$ for the triangle while the serial averaged one had respectively $7.92mm$, $5.09mm$ and $6.35mm$. These numbers show the novel approach was on average more than six times better controlling the Z axis when compared with the serial.

Equivalent results can be seen for the circle and triangle shapes (Figures 5.24, 5.25, 5.26 and 5.27). Again, the parallel system obtains a better control of the Z (or height) during the movements necessary to draw the shapes on top of the simulated table.

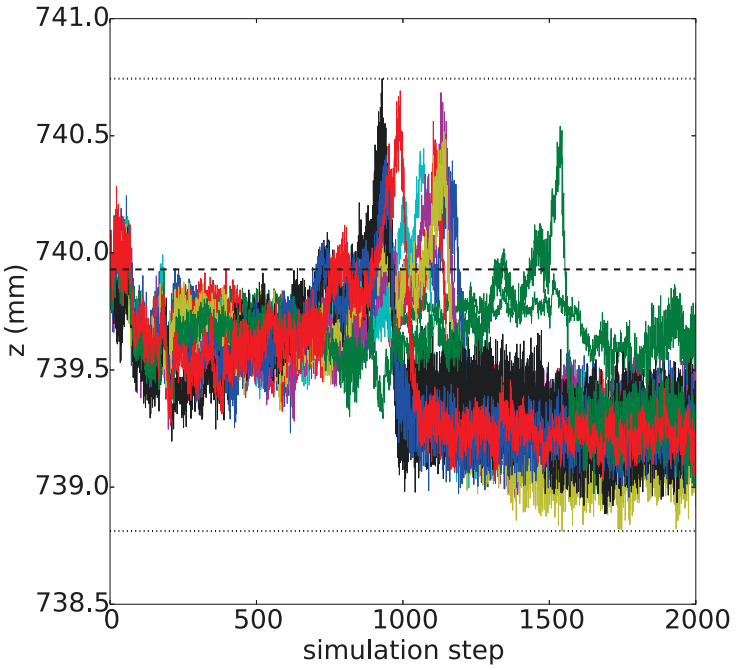


FIGURE 5.22: Visualization of the Z curves (square) in time (all ten trials). The dashed line represents the original curve ($Z_{original} = 739.93mm$) and the dotted ones the maximum/minimum values $Z_{max} = 740.74mm$ and $Z_{min} = 738.81mm$ - using the framework presented here (*parallel* method).

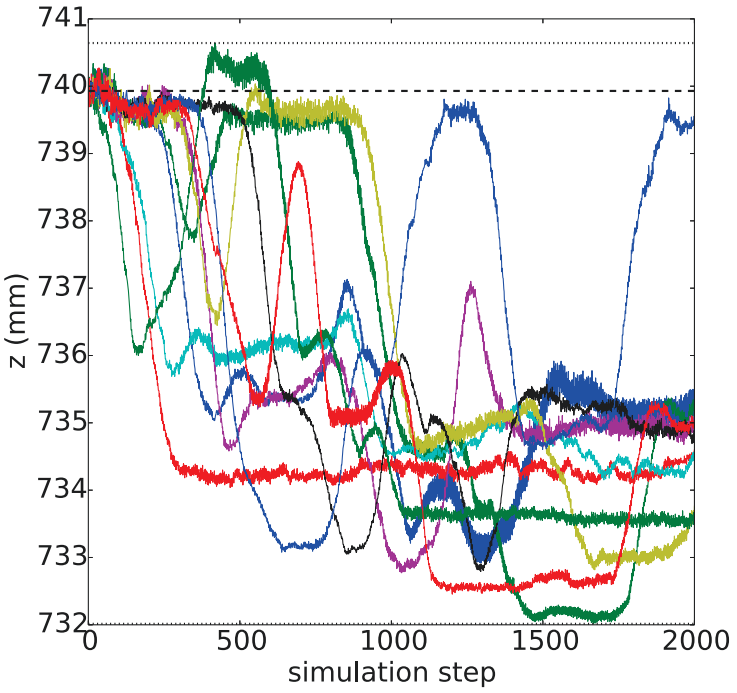


FIGURE 5.23: Visualization of the Z curves (square) in time (all ten trials). The dashed line represents the original curve ($Z_{original} = 739.93mm$) and the dotted ones the maximum/minimum values $Z_{max} = 740.64mm$ and $Z_{min} = 732.01mm$ - using the *serial* averaged method.

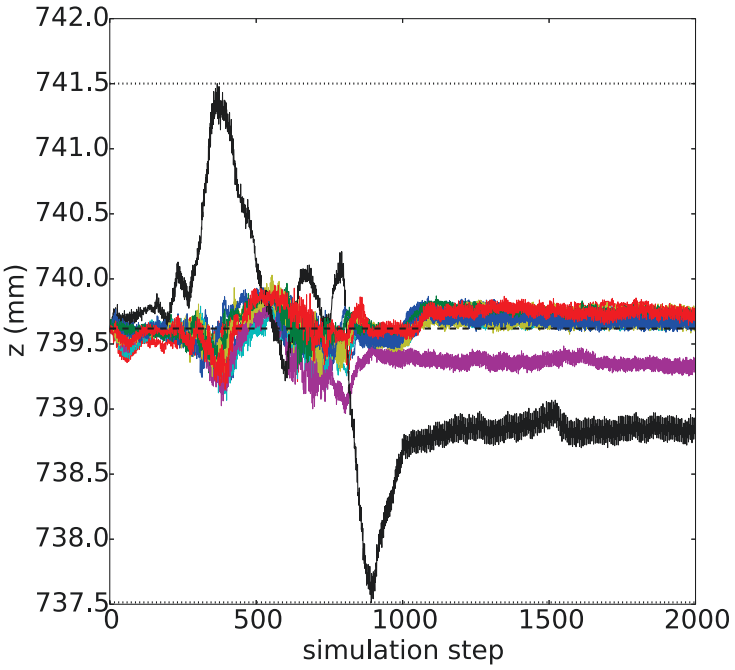


FIGURE 5.24: Visualization of the Z curves (circle) in time (all ten trials). The dashed line represents the original curve ($Z_{original} = 739.62mm$) and the pointed ones the maximum/minimum values $Z_{max} = 739.99mm$ and $Z_{min} = 739.07mm$ - using the framework presented here (*parallel* method).

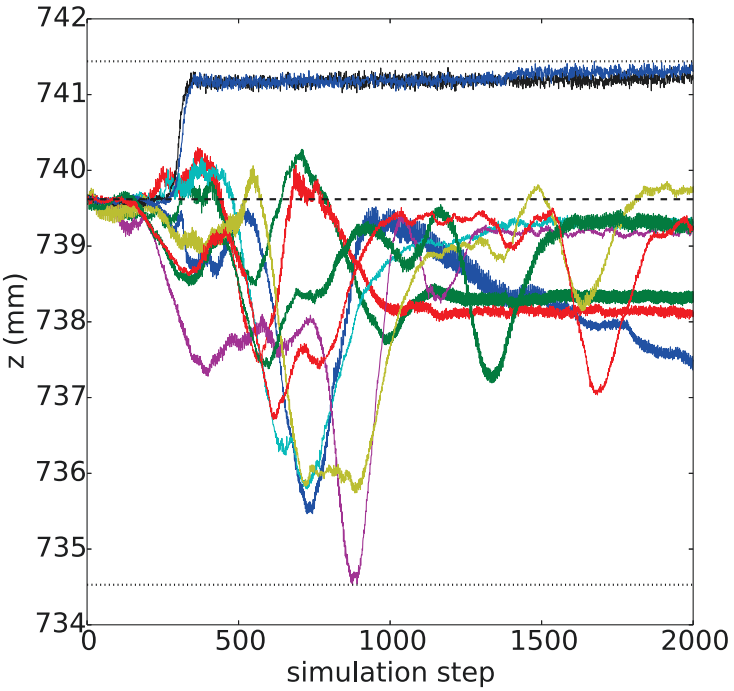


FIGURE 5.25: Visualization of the Z curves (circle) in time (all ten trials). The dashed line represents the original curve ($Z_{original} = 739.62mm$) and the pointed ones the maximum/minimum values $Z_{max} = 741.44mm$ and $Z_{min} = 734.53mm$ - using the *serial* averaged method.

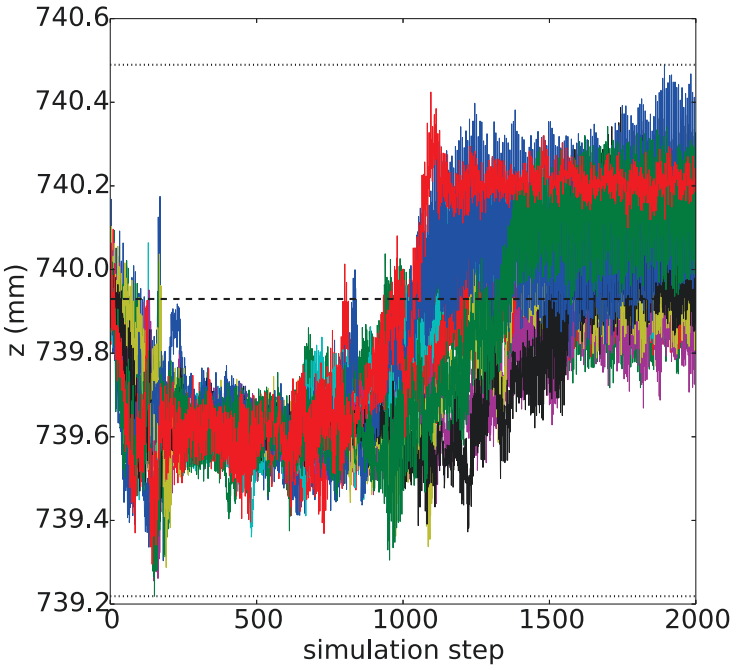


FIGURE 5.26: Visualization of the Z curves (triangle) in time (all ten trials). The dashed line represents the original curve ($Z_{original} = 739.93mm$) and the pointed ones the maximum/minimum values $Z_{max} = 740.49mm$ and $Z_{min} = 739.22mm$ - using the framework presented here (*parallel* method).

5.3.5 Real Baxter robot experiment

As stated at the beginning of this chapter, a proof-of-concept experiment was done using the real Baxter robot. It consisted of drawing a square (Figure 5.4) making use of its left arm and the parallel framework introduced here, but in an open loop configuration as the joint values were recorded from the robot simulations using V-REP. In Figure 5.28, it is possible to see the sequences of steps until the complete square is drawn. Not surprisingly, the final drawing did not have as much noise as the one from Figure 5.7 because the robot arm together with its actuators functioned as a low-pass filter. Although simple, this experiment was very useful to test the tools developed to communicate from the SNN to Baxter using UDP packets as well as the arm's calibration and table levelling. The Z axis spike (pen lifts up off the table) seen on Figure 5.22 could also be seen in the trial presented here, but not in all trials with the real robot as a result of the morphological filter formed by the whole physical set-up.

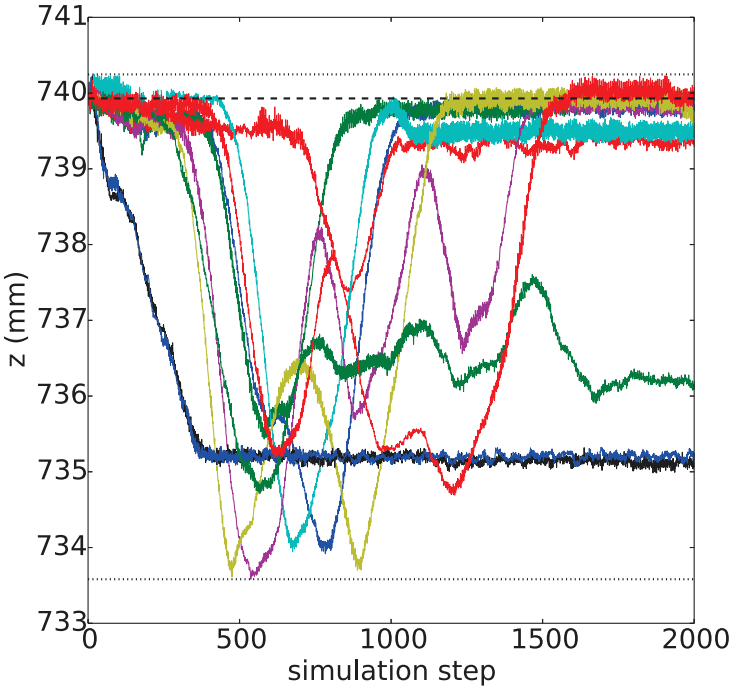


FIGURE 5.27: Visualization of the Z curves (triangle) in time (all ten trials). The dashed line represents the original curve ($Z_{original} = 739.93mm$) and the pointed ones the maximum/minimum values $Z_{max} = 740.25mm$ and $Z_{min} = 733.58mm$ - using the *serial* averaged method.

FIGURE 5.28: Proof-of-concept experiment where the resultant joints generated by the system presented in this chapter were directly injected into the real Baxter robot in order to generate the square drawing (Figure 5.7).

5.4 Conclusions

A new simple, yet powerful, idea of using parallel Liquid State Machines sharing the averaged outputs as their feedback was presented in this chapter. The task used as benchmark was based on the ideas of action learning [34] where an external teacher showed the humanoid Baxter robot how to draw simple shapes on top of a table.

Results presented here show a clear improvement in the task when the parallel method (Figure 5.2) was used. Besides the Liquid State Machine model introduced in [21], most of

the Spiking Neural Network implementations do not make use of high noise levels during the testing phase.

A similar idea of multiple liquids (or columns) to improve performance was already presented in [21]. However, only one bigger *readout* was used for all columns, there were no feedback connections from the output of the system to the input and the results came from serial averaging many trials instead of the parallel system proposed here.

Considering the results from this chapter, where it was shown the use of ensembles had a better performance than the traditional serial approach, the next chapters concentrate on the comparison between the system presented here and the multiple columns initially introduced by Maass et al. Chapter 6 explores experiments testing the robustness of both systems to the increase of noise levels while Chapter 7 focuses on the internal damage of the networks.

Chapter 6

Effects of Noise on Liquid State Machines Robot Controllers

6.1 Introduction

In an attempt to start developing solutions for the current problems robotic systems encounter when exposed to an environment with a high level of radiation, it is proposed in this chapter the use of biologically inspired robot controllers (see Chapter 5) for a more nature-like graceful degradation, instead of a catastrophic failure, when exposed to radiation. Modular (see Chapter 5 for more details) and Monolithic designs of a special type of feedback enhanced parallel Liquid State Machines (LSMs) [21, 24] are exposed to different noise levels, in a simulated environment, and the results analysed with a robotic task as the benchmark. White Gaussian noise is injected directly into the neuron model, which could be seen as an example of the result from the non-destructive effects of radiation. The idea of modelling faults in a more abstract level is not new and it has already been explored [131].

Additionally, LSMs are modelled based on Spiking Neural Networks (SNN); therefore, power efficiency could be easily acchieved implementing the SNN in a neuromorphic hardware such as SpiNNaker [30], BrainScaleS [64] or Silicon Neurons (SiN) [132] which could also improve the reliability even further.

6.2 Methods

The investigation presented in this chapter was based on the new humanoid robot control framework using parallel, diverse and noisy groups of biologically inspired LSM introduced in Chapter 5. The robot controller was able to reproduce trajectories (shapes) previously learned from a teacher, but the effects of varying noise levels were not studied.

More precisely, in this chapter, eleven different noise levels (100 trials each), starting from the standard one defined in [26] and going up to 100% above that (see Section 6.2.2), were employed to verify the noise effect on two different parallel LSM configurations: Modular and Monolithic (see Section 6.2.1). The final analysis was done through the robot's resultant movement performing the benchmark task of drawing a square shape on a table (see Section 6.2.3).

All the source code necessary to reproduce the results presented here are available at the author's Github repository[1].

6.2.1 Modular and Monolithic Parallel LSM

The idea of breaking an LSM into multiple liquids (or simplified models of cortical columns) in parallel to increase the computational power was initialy presented in [21], but only in Chapter 5 an external feedback loop was added, as suggested in [26, 24], explored for this particular situation. Moreover, the parallel system presented in [21] had an external output layer (*readout*) shared among all neurons contrasting with the one presented in Chapter 5 where each liquid was trained individually and had its own *readout* resulting in a system with improved learning capabilities. Those two approaches are called here the *Monolithic Parallel LSM* (Figure 6.2) and *Modular Parallel LSM* (Figure 6.1), respectively. To facilitate comparisons, the same random seeds from Chapter 5, therefore the same *liquids*, were employed here, but the readout layers were trained again as the Monolithic approach has not been tested before.

[1] github.com/ricardodeazambuja/ICONIP2016

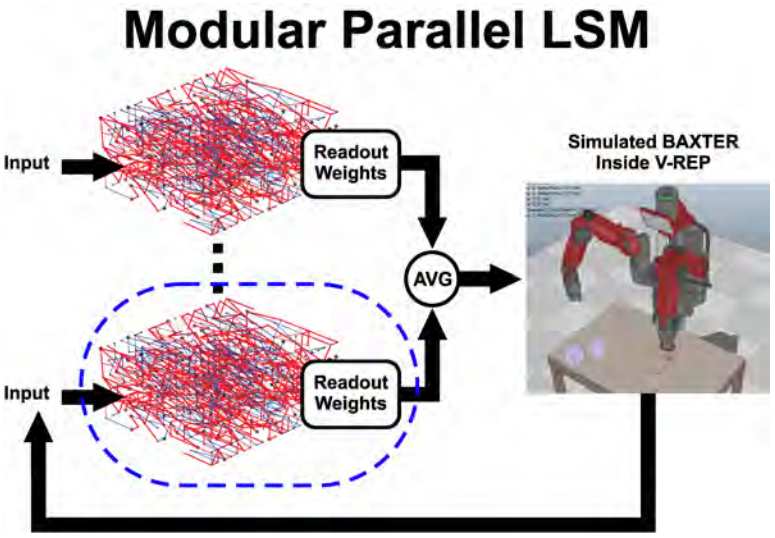


FIGURE 6.1: The *Modular* approach (above) uses individual readout layers for each liquid. It reuses the same five LSM (liquids) from Chapter 5, but with retrained readouts.

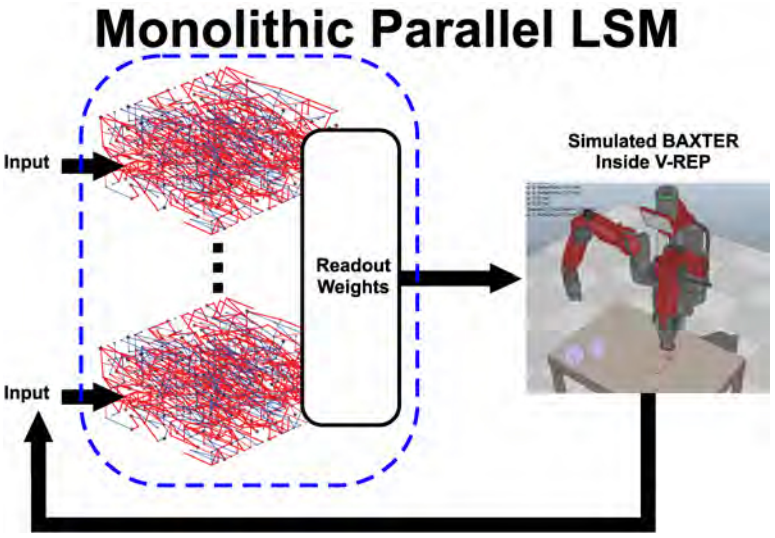


FIGURE 6.2: The *Monolithic* approach (above) has only one readout layer shared among all its neurons. It reuses the same five LSM (liquids) from Chapter 5, but with retrained readouts.

6.2.2 Neuron Model and Noise Levels

The neuron model applied here, the Leaky Integrate and Fire (LIF) partially represented by the Equation 2.1, has its membrane reset voltage (V_{reset}) drawn from a uniform distribution ($[13.8mV,\ 14.49mV]$) when the neural network is created and generates a spike when it reaches $15mV$ ($V_{threshold}$) - considering the last spike occurred long enough to avoid the refractory period. On the algorithmic level (BEE simulator, Section 2.3.6.3), the membrane

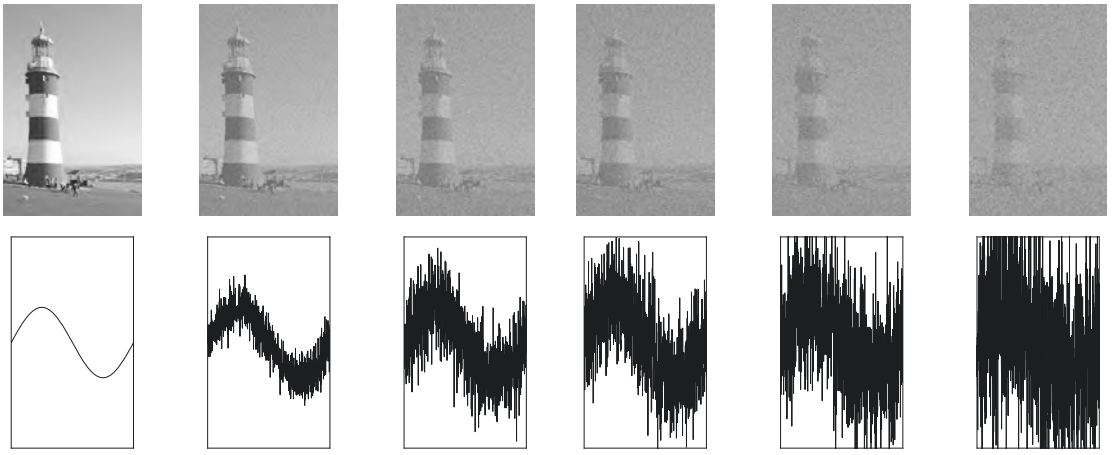


FIGURE 6.3: As an easy way to visualize the noise effects, the photographs (top row) had added to their greyscale values (0 to 255) noise proportional to how A_{noise} affects the membrane voltage, varying it from 0.0 to 1.0. On the bottom row, noise is applied to a sinusoid whilst keeping the same scale.

voltage is always clamped between $-15mV$ and $+15mV$, although its rest potential is $0mV$ and it is set back to the reset value (V_{reset}) after every spike (See Table 2.1 for more details). Consequently, most of the time, the neuron membrane will fluctuate between V_{reset} and $V_{threshold}$ or, in the worst scenario, with $\Delta V \approx 1.2mV$.

The simulation of a faulty system through the injection of noise (see Section 7.1) is accomplished using the i_{noise} variable from Equation 2.1. Its value is drawn from a Gaussian distribution ($\mu = 0$ and $\sigma = 1nA$) multiplied accordingly to what it's called here *noise level* (A_{noise}). Having a noise level of 100%, 110%, 120%, ..., 200% means the multiplier value goes from 1.0 up to 2.0. The parameters were defined according to what was presented in [21] and [26], hence $c_m = 30nF$ and $\tau_m = 30ms$. This yields, ignoring other noise sources, a Signal-to-noise ratio (SNR) of approximately $\left(\frac{\Delta V/mV}{A_{noise}}\right)^2$. Thus the system has its SNR varied from 1.44 to 0.36 (see Figure 6.3).

6.2.3 Benchmark Task

The benchmark test consisted of the simultaneous control of four joints (S1, E1, W0 and W1 from Figure 2.4) of a simulated Baxter robot in order to draw a square shape on top of a table (for more details see Chapter 5). All analyses were done on the robot's taskspace (Cartesian space) instead of joint space. Although being a two dimentional shape drawn

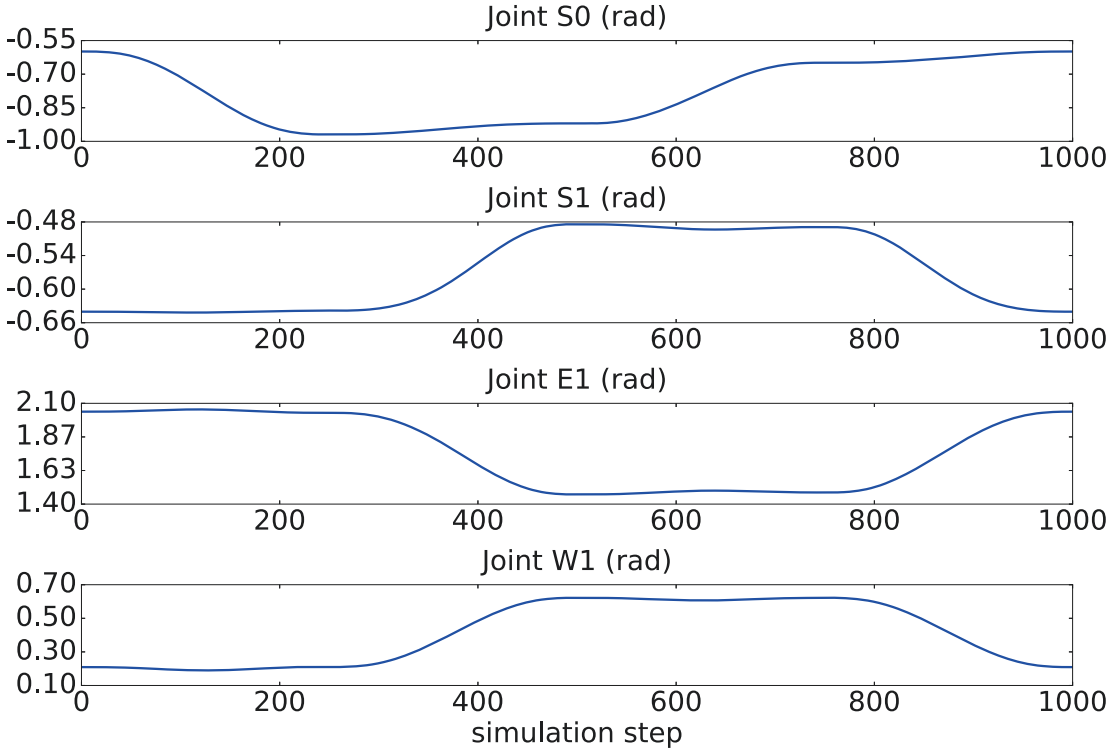


FIGURE 6.4: Joint curves necessary to command the robot to generate the square shape.

on a surface, the system follows a human-inspired movement [117] and, for that reason, must keep in control a total of four dimensions: X, Y, Z and time.

6.2.3.1 Cost calculation

The cost calculation was defined using the Dynamic Time Warping (DTW), already employed in Chapter 5 and better detailed in Section 2.3.7. Therefore, it considers the time series and warps it as necessary in order to find an optimal match and generates a final path cost. Contrasting with what was presented in Chapter 5, here cost values were not normalised, but the final value was used in order to enable comparisons between different noise levels too.

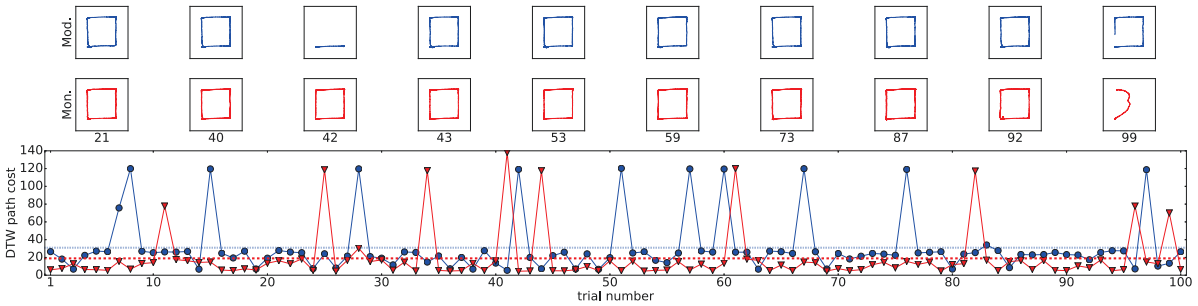
6.3 Results and Discussion

Eleven distinct levels of noise were tested here for both, Modular and Monolithic, approaches (Section 6.2.1) with A_{noise} varying from 1.0 to 2.0 (see LIF neuron model definition, Section 2.3.2.2). These experiments resulted in a total of $2,200$ simulations, where each one consisted of $3,000$ spiking neurons (five 600 neurons liquids in parallel). After every run, the joint values produced were loaded into the simulated Baxter robot inside V-REP to verify the final movement executed for the benchmark task and the results processed by the DTW algorithm (Chapter 2, Section 2.3.7).

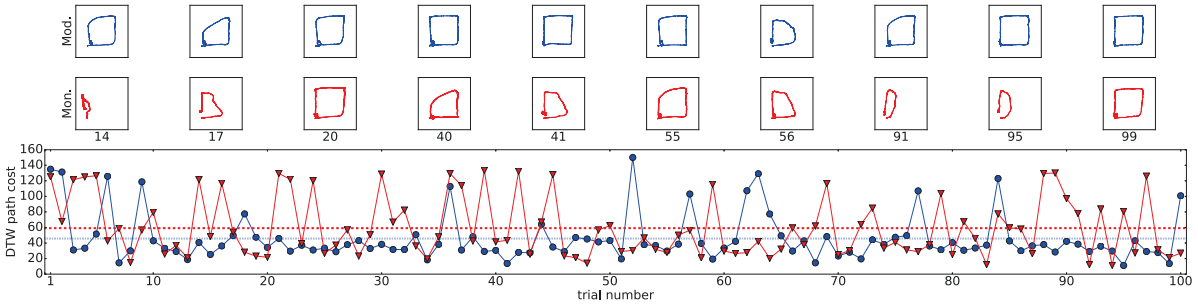
To better illustrate the final shape results, the DTW path cost values generated from three different noise intensities (A_{noise} equal to 1.0, 1.5 and 2.0) are presented in Figure 6.5. All the one hundred trials (bottom), but only ten examples of the final shapes generated (top - with trial number indicated) are depicted.

Clearly, as the noise is increased, the square shapes become strongly degraded, but the Modular approach still can produce some rectangular forms even with $A_{noise} = 2.0$ or a noise level twice that injected during the readout training phase (see Figure 6.3 for a visual hint about noise levels). However, when using the standard noise level ($A_{noise} = 1.0$), the Monolithic approach had a better performance with an average cost value about 39% smaller than the Modular one. This type of system, sometimes, get stuck into a value and needs noise to be able to proceed, but the DTW algorithm penalises it as the trajectory it sees, although with a nice quality, was not completed. Therefore the difference between Modular and Monolithic approaches, with $A_{noise} = 1.0$, could be explained by the limited number of simulated steps (2,000 steps).

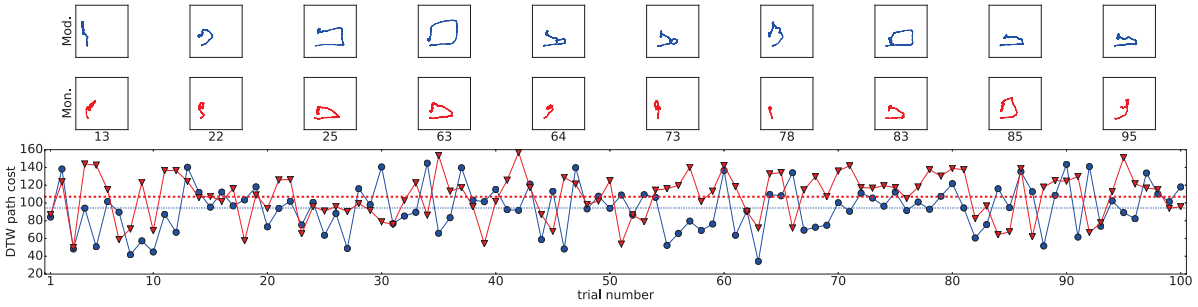
In Figures 6.6 and 6.7, all hundred trials with $A_{noise} = 2.0$ were plotted together on $3D$ Cartesian space (same scale for all views) to make it easier to present their $3D$ structure, as mean values do not work well if there are time delays among trials. Despite the fact that a strong effect on the $2D$ square shape is clear, the Z axis (or the height control) is barely affected (top right) when analysed in the $3D$ space.



(A) $A_{noise} = 1.0$



(B) $A_{noise} = 1.5$



(C) $A_{noise} = 2.0$

FIGURE 6.5: Each plot shows the DTW path cost (bottom) for all trials and some of the shape outcomes (top) comparing the Modular (blue, circles) and the Monolithic approaches (red, triangles). The shapes (top) were selected based on the sorted cost values of both configurations to show a more comprehensive set of examples. Average values plotted as horizontal dashed lines (bottom).

The main question raised at the introduction was about the behaviour of this kind of system when affected by different noise levels and if it would have a nature like graceful degradation. To analyse that, the DTW path cost average and standard error values were calculated and are presented in Figure 6.8. The same figure also presents what would be the evolution of the cost considering the initial values incremented in steps of 10%.

Both approaches presented here, Modular and Monolithic parallel LSM, had what is considered a graceful degradation, as with the increase of the noise the systems did not catastrophically fail, but the DTW path cost grew in a well behaved manner or, in other words,

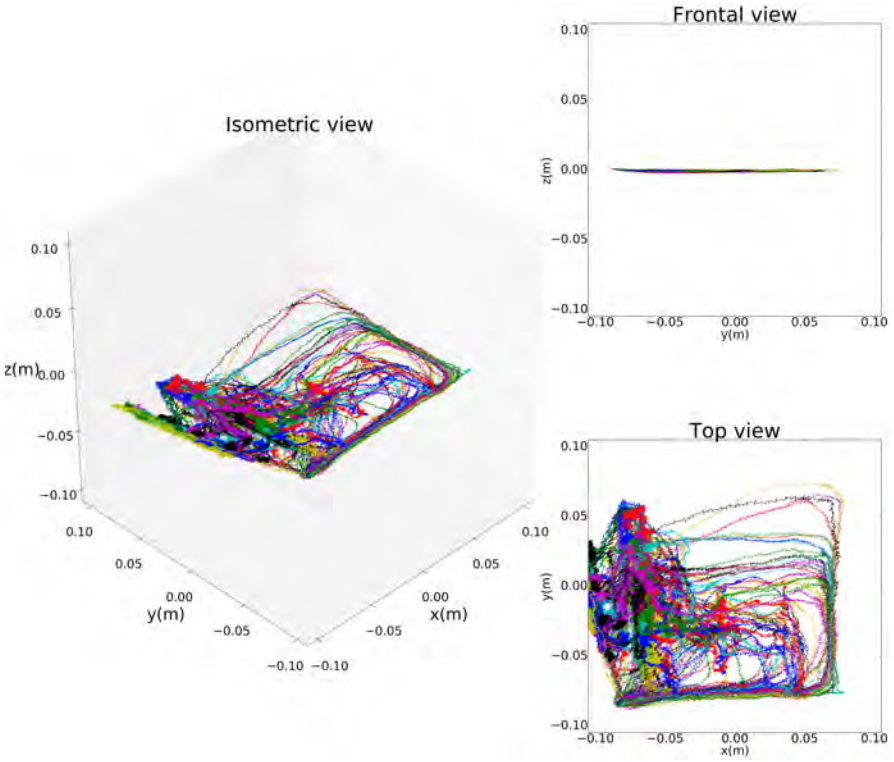


FIGURE 6.6: Modular approach with A_{noise}=2.0.

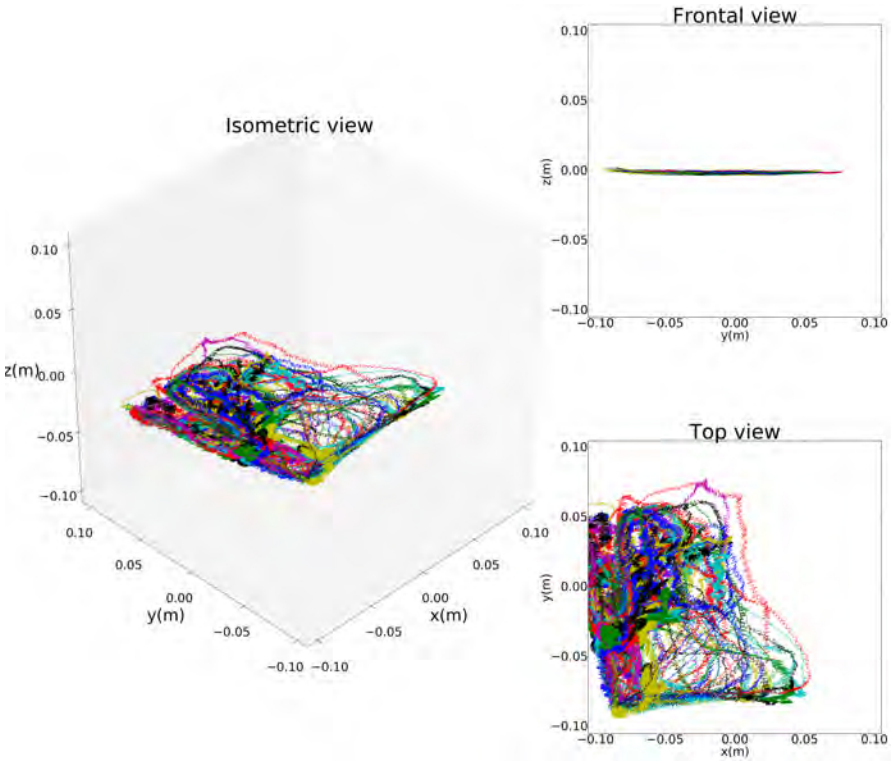


FIGURE 6.7: Monolithic approach with A_{noise}=2.0.

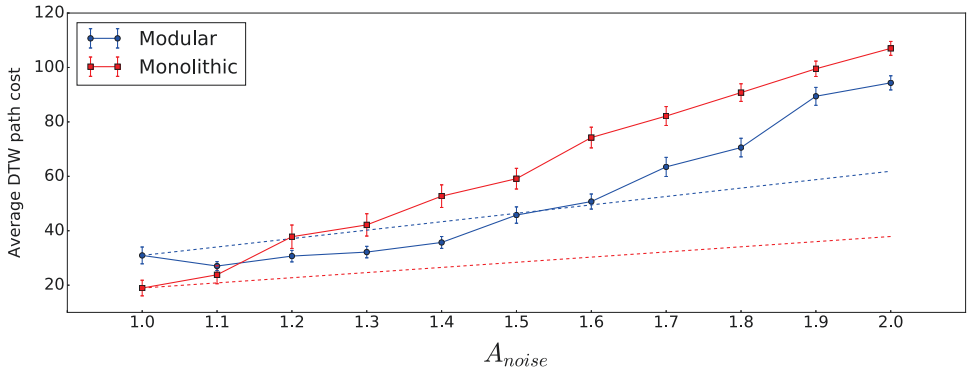


FIGURE 6.8: Average DTW path cost and its standard error for all trials (hundred in total for each A_{noise} level). The growth, considering the first value incremented by 10%, 20% ... 100%, is shown as a dashed line.

without abrupt changes. Comparing both LSM configurations, the Modular approach had an almost constant behaviour up to A_{noise}=1.4 when it started growing linearly with nearly the same slope as the Monolithic approach. Therefore, the Modular system (between the A_{noise} range of 1.0 to 1.4) was able to withstand the noise better than a simple linear growth as showed by the dashed blue line (Figure 6.8) whilst the Monolithic configuration always increased its DTW path cost with the increase of noise.

6.4 Conclusions

The robot controllers presented in this chapter were able to withstand, or at least gracefully degrade, when exposed to different noise levels - modelled here as white Gaussian noise based currents injected into the neuron model. These noisy currents could be seen, in a simplified way, as the consequences of exposition to non-destructive radiation.

It is important to develop systems that are able to be implemented using new technologies, such as neuromorphic hardware, as they seem to be one of the possible ways to bypass the declining applicability of Moore's law [133] without having to expend huge amounts of energy [4]. Also, one of the strategies to decrease energy consumption, in a quadratic way, is the reduction of the voltage supplied to the digital circuits (near-threshold voltage [134]). However, this naturally leads to a decrease in the noise immunity as the voltage margin until a transistor changes its state is reduced. Another consequence of voltage reduction is within the speed a transistor changes its state. Still, neural systems are well known to

be parallel, but relatively slow systems when compared to modern digital circuits. Even if MEMS-based logic gates [135] evolve up to the point of a final product, a digital system does not degrade gracefully in normal conditions and always needs extra gates to implement error correction.

The Modular design presented here opens up the possibility for a hot-swap hardware implementation, fitting SpiNNaker very well as it is able to turn on and off chips if necessary, and also decreasing the time and memory spent during learning. Moreover, having smaller readout layers, the time spent during learning is smaller than when using the Monolithic setup.

The Monolithic approach uses one big readout layer while the Modular one has smaller individual output layers and a node producing the average among them. In a future work, this simple average junction could be replaced by an extra on-line learning layer with weights connecting the analogue readout outputs directly to the neuron membrane, opening the possibility to vary the amount of trust the system has to each individual LSM without the need of changing the readout weights, thus saving energy and simplifying the design.

Additionally, to extend what was presented here, other parameters could be checked to verify their influence on the robustness. One good example, easily implemented, is the number of parallel liquids and the number of neurons used with each one.

In some trials, the systems got stuck in the middle of a well defined trajectory producing high DTW path cost values (see Figure 6.5a, trials 42 and 99). Our experience, after several experiments have been done using this type of system, together with the results presented in the Figure 6.8, suggests a certain minimum background noise is actually necessary for this kind of system. This idea of a “good" noise is not new [78] and will be left as another avenue for future works.

Chapter 7

Robustness to Neural Decimation on Multiple Column Liquid State Machines

7.1 Introduction

In Chapter 6, two different types of multiple columns LSMs, Modular and Monolithic (see Sections 6.2 and 7.2 for a better definition), enhanced with external feedback connections (see [24] and Chapter 5.2 for more details) were tested against noise injected directly into its artificial neurons. The results confirmed both systems had a gracious degradation, i.e. the performance decreased proportionally to the noise amplitude instead of an abrupt or catastrophic failure, but nothing was said about their behaviour in relation to damage to the internal nodes.

Here, those same LSM systems from Chapter 6 were exposed to different types of decimation. They had their internal connections (Section 7.2.2), neurons (Section 7.2.3 and 7.2.4) and whole columns (Section 7.2.5) decimated while they were benchmarked in a robotic task based on the collaborative robot Baxter (Section 7.2.6.1) using Dynamic Time Warping (Section 7.2.6.2) to generate a comparative cost value.

7.2 Methods

When first introduced [21], the LSM was already known to be computationally more powerful when multiple columns, also called *liquids*, were used instead of a bigger one, but it did not have the external feedback connection between the output and the input [24]. When such a feedback connection is introduced, an LSM becomes capable of emulating arbitrary Turing machines in an ideal situation.

Traditionally, an LSM experiment generates its final results averaging several trials. However, multiple trials usually have different time delays and simply averaging them do not generate the best results. A solution for this problem was presented in Chapter 5 where a parallel implementation yields better results than averaging multiple trials. Furthermore, such a system was already tested against the effects of varying the noise injected directly into the neuron model (Chapter 6), showing a graceful degradation instead of a catastrophic failure.

The Modular and Monolithic LSM approaches (Section 7.2.1) have been already tested in relation to their robustness when a noisy current is injected directly into the neuron model (see Chapter 6), still no experiments were done to verify their behaviour to internal damage. Also, it was not possible to find in the literature any work where an LSM using multiple columns and an external feedback connection [24] was tested against internal damage. Schürmann et al. [94] only mention their liquids shown robustness against faults introduced after the readout was trained, but no figures or data was presented. Hazan and Manevitz [136] presented experiments with an LSM tested against damage and noise, but their system was mono-column, it did not have the important external feedback connection (between the readout output and the input) and the task was not a robotic one.

Since in Chapter 6 it was only addressed the possibility of Single-Event Upsets (SEU) or *soft-errors*, it was very important to verify how the implemented LSM would react to more destructive events such as a Single-Event Latchup (SEL)[1], Single-Event Gate

[1]"An abnormal high-current state in a device caused by the passage of a single energetic particle through sensitive regions of the device structure and resulting in the loss of device functionality." from https://www.jedec.org/standards-documents/dictionary/terms/single-event-latch-sel

Rupture (SEGR)[2], or Single-Event Burnout (SEB)[3].

Here, SEL, SEGR and SEB are simulated in a very simplified way by deactivating neurons or connections. Four different possible scenarios were individually considered: decimation of internal connections (Section 7.2.2), decimation of individual neurons (Section 7.2.3), decimation of individual neurons in a single column (Section 7.2.4) and decimation of entire columns (Section 7.2.5). The verification was done using a robotic task, drawing a square on top of a flat surface (Section 7.2.6.1), together with Dynamic Time Warping (Section 7.2.6.2) as a benchmark. The results were compared with the help of Welch's t-test (Section 7.2.6.3).

All implementation details and source code necessary to reproduce what was presented here can be found at the author's Github repository[4].

7.2.1 Modular and Monolithic Multiple Columns LSM

The Modular and Monolithic systems can be seen in the Figures 7.1 and 7.2. They have the same specifications presented in Chapter 6. They are composed of five columns (or liquids or reservoirs), each one including six hundred artificial neurons (three layers, or a structure with 20x5x6 neurons), totalling three thousand neurons. The neuron model is the LIF with exponential synapses (see 2.3.2.2). In addition, the artificial neurons have white Gaussian noise ($\mu = 0$ and $\sigma = 1nA$) directly injected as a current (this happens every simulation time step) and the membrane voltage values are drawn from an uniform distribution ($13.5mV$ to $14.9mV$) every time a new simulation starts. Consequently, all trials are, by design, unique.

A Modular system can be implemented and put together using different hardware, because it is a complete system by itself and the training occurs individually. Even if one of the parallel pieces needed replacement, it would only require the connection of a spare one. In

[2]"An event in which a single energetic-particle strike results in a breakdown and subsequent conducting path through the gate oxide of a MOSFET." from https://www.jedec.org/standards-documents/dictionary/terms/single-event-gate-rupture-segr

[3]"An event in which a single energetic-particle strike induces a localized high-current state in a device that results in catastrophic failure." from https://www.jedec.org/standards-documents/dictionary/terms/single-event-burnout-seb

[4]github.com/ricardodeazambuja/IJCNN2017-2

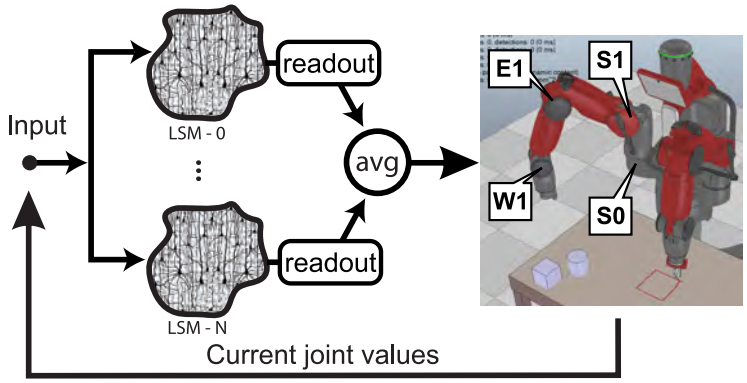


FIGURE 7.1: A Modular system is formed by multiple distinct complete LSM in parallel. They receive the same inputs (proprioceptive feedback) and their readout outputs are averaged and only then sent to the robot.

this work, it was implemented using the simulator developed in this thesis (Section 2.3.6.3) running on a PC.

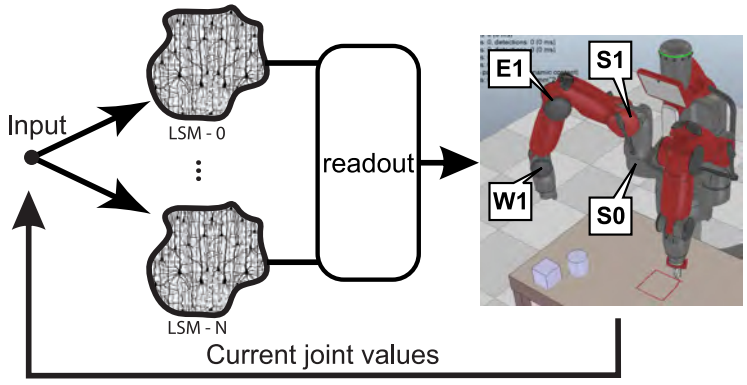


FIGURE 7.2: A Monolithic system is similar to the Modular one, since it has multiple columns (liquids). However, each column receives only part of the inputs and there is only one readout output.

The Monolithic approach has only its columns implemented separately since the inputs each column receives are not the same. For the experiments presented here, the same columns from the Modular system (this was guaranteed by the use of the same random seeds) were employed and everything was simulated in a PC using the same software mentioned earlier.

7.2.2 Decimation of Internal Connections

The experiments testing the effects of damages to the column's internal connections were simulated using nine different levels of decimation: 1%, 2.5%, 5%, 7.5%, 10%, 25%, 50%, 75% and 100% of connections. The same number of connections was randomly decimated

from all columns, therefore they are evenly distributed. Modular and Monolithic systems were tested with one hundred trials each level (1800 trials in total). Both excitatory and inhibitory connections were randomly pruned during each trial. The pruning was achieved by changing the connection weight value to zero. Input and output connections were not changed.

7.2.3 Decimation of Neurons

Following pilot experiments, not presented here, the number of decimated neurons per column was defined as: 6, 12, 18, 24, 30, 36, 42, 48, 54 and 60 neurons. Again, one hundred trials were executed for each configuration (2000 trials in total). During each trial, the same number of random selected neurons was deactivated in each column and, therefore, those neurons did not generate spikes during that trial.

7.2.4 Decimation of Neurons in a Single Column

In these experiments, instead of deactivating the same number of neurons in all columns (Section 7.2.3) the decimation was concentrated in only one column. The number of neurons decimated were: 30, 60, 90, 120, 150, 180, 210, 240, 270 and 300 neurons. For each experiment, one hundred trials were executed (2000 trials in total).

7.2.5 Decimation of Columns

After testing individual columns to partial damage (Section 7.2.4), since the Modular approach has the possibility of implementing hot swap for each parallel LSM, individual columns were totally disconnected. Because both, Modular and Monolithic configurations, had a total of five parallel columns, they were tested for the disconnection of 1, 2, 3 and 4 columns. One more time, one hundred trials were executed for each situation (totalling of 800 trials).

For the Monolithic system, all the neurons from the decimated columns stopped producing spikes and, consequently, the readout units associated to those neurons output a zero value.

The Modular configuration had entire modules shut down and, therefore, those modules were not used for the generation of the final averaged output value.

7.2.6 Benchmark task

7.2.6.1 Simulated Baxter Robot

In the Chapters 5 and 6, all the simulations were executed using V-REP (Chapter 2, Section 2.3.5). However, here, the testing phase was carried out with only a simplified model of the Inverse Kinematics (IK). Using this simplified IK instead of the full V-REP simulator, more trials were generated in the same amount of time. The details and source code for the IK can be found on the Github repository from the link provided in Section 7.4.

7.2.6.2 Dynamic Time Warping

The output generated by the robotic benchmark task (Section 7.2.6.1) can be seen in two different ways: a time series formed by all points (X, Y, Z) or the final 2D drawing. Still, if one just analyses the time series, e.g. a Mean Square Error comparing to the original one, time delays or changes on the velocity profile could have a huge influence whilst the final drawing would look perfect. Using the cost generated by the Dynamic Time Warping (DTW) method (Chapter 2, Section 2.3.7) it's easier to compare the final generated 2D shape, still taking into account the time dimension. This same idea, using DTW as a benchmark for a robotic task, has already been applied in previous chapters.

7.2.6.3 Welch's t-test

The comparison between the DTW results from the Modular and Monolithic approaches were made using the Welch's t-test. This test is a variant of the famous Student's t-test. The Welch's t-test was created to be applied when the variances of the two samples are unequal (heteroscedastic) [137]. This test was applied to verify if two experimental results have equal means (null hypothesis). To calculate the Welch's t-test, the Scipy package

method *ttest_ind*, with the option *equal_ var=False* was adopted and both the *t-value* and the *p-value* were reported.

7.3 Results and Discussion

The results presented in this section are the outcome of four experiments that generated in total six thousand and six hundred unique trials. The data were analysed separately according to the type of decimation tested.

DTW results are presented in Figures 7.3 to 7.7. Each point represents the average of one hundred trials and the bars the standard error.

Statistics comparing both approaches, Modular and Monolithic, are summarised in Tables 7.1 to 7.4. Results that are statistically significant are shown in bold.

Finally, the drawings generated during the experiments, where the two systems were tested for the decimation of neurons in a single column (Section 7.2.4 and 7.3.1.3), are presented in Figure 7.6. These results are interesting because they give an idea about what range of costs the DTW method generates since the values can be verified in Figure 7.5.

7.3.1 Simulation Results

7.3.1.1 Decimation of Internal Connections

After an internal connection is decimated, even though the presynaptic neuron is still active, the postsynaptic neuron does not receive new spikes from that connection reducing its computational power. The readout is trained to generate values based on the column's dynamics and if it changes too much the readout loses its ability to generate the correct next joint values.

In the Figure 7.3, it's possible to visualise, when the percentage of randomly decimated connections was below 5%, the Modular configuration was less affected by decimation. The same situation can be also verified through Table 7.1. Clearly, from 5% both systems

start converging to the same curve signalling there's a threshold value where the Modular approach can't withstand this type of aggression.

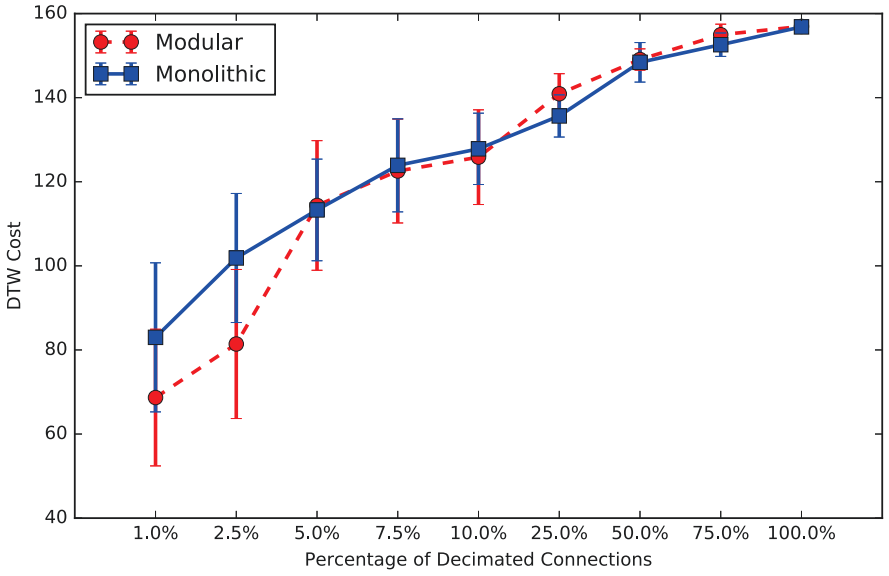


FIGURE 7.3: Decimated internal connections - Results from the experiments where a certain percentage of the internal connections between the neurons inside each column, randomly selected, were decimated. The bars represent the standard error (total of 100 trials). Statistics are presented in the Table 7.1.

7.3.1.2 Decimation of Neurons

The results from the experiments where neurons were randomly deactivated to simulate a destructive event are presented in Figure 7.4 and Table 7.2. The two configurations, Modular and Monolithic, had approximately the same behaviour after 18 neurons (or 3% of the neurons) were randomly decimated inside all available columns. Although, up to that point the Modular one suffered fewer effects from the damages.

When a neuron in the simulations is deactivated and stops producing spikes, all its post-synaptic connections become inactive too. The main difference in relation to the results

TABLE 7.1: Welch's t-test results between Modular and Monolithic approaches - decimated internal connections (100 trials)

Percentage of decimated internal connections	*t-value*	*p-value*
1.0%	**1.9750**	**0.0497**
2.5%	**2.8941**	**0.0042**
5.0%	-0.1803	0.8571
7.5%	0.2695	0.7878
10.0%	0.4656	0.6421
25.0%	**-2.5206**	**0.0125**
50.0%	-0.4167	0.6775
75.0%	**-2.0999**	**0.0370**
100.0%	**-7.5188**	**0.0000**

TABLE 7.2: Welch's t-test results between Modular and Monolithic approaches - decimated neurons (100 trials)

Number of decimated neurons per column	*t-value*	*p-value*
6	**2.9848**	**0.0032**
12	**3.5899**	**0.0004**
18	0.5569	0.5782
24	0.0574	0.9543
30	1.5051	0.1339
36	0.3256	0.7451
42	1.9258	0.0556
48	-1.9451	0.0532
54	0.3843	0.7012
60	0.2321	0.8167

presented in Section 7.3.1.1 (Figure 7.3 and Table 7.1) is that even when some of the connections are broken the postsynaptic neuron still can produce spikes, whilst here it stops supplying information to the low-pass membrane filter and, consequently, the readout. This could explain why the decimation of connections generated a stronger reaction only when it reached 5% and the decimation of neurons had a stronger destructive effect already at 3%.

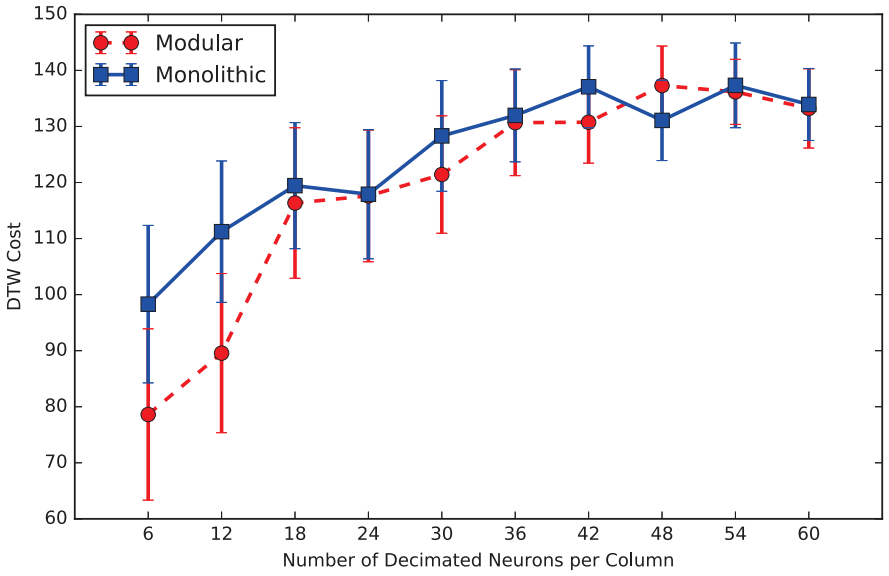


FIGURE 7.4: Decimated neurons - Results from the experiments where a certain number of neurons were randomly decimated from each of the five columns. The bars represent the standard error (total of 100 trials). Statistics are presented in the Table 7.2. The lower the DTW Cost, the better are the results.

7.3.1.3 Decimation of Neurons in a Single Column

Sometimes, when a electronic system is affected by a Single-Event Latchup, Single-Event Gate Rupture, or Single-Event Burnout, those events could generate a chain reaction affecting components closely connected. Since the systems tested here are capable to be implemented in parallel, a possible outcome would be a situation where only one column is affected. The results from Figure 7.5 and Table 7.3 show exactly this situation and some of the trials are presented in the Figure 7.6.

To clarify, in Section 7.3.1.2, the same number of decimated neurons, randomly chosen, would be distributed among all five column, therefore affecting all them at the same time. Here, the neurons were also randomly chosen, but only one random column had its neurons affected by the decimation during each trial.

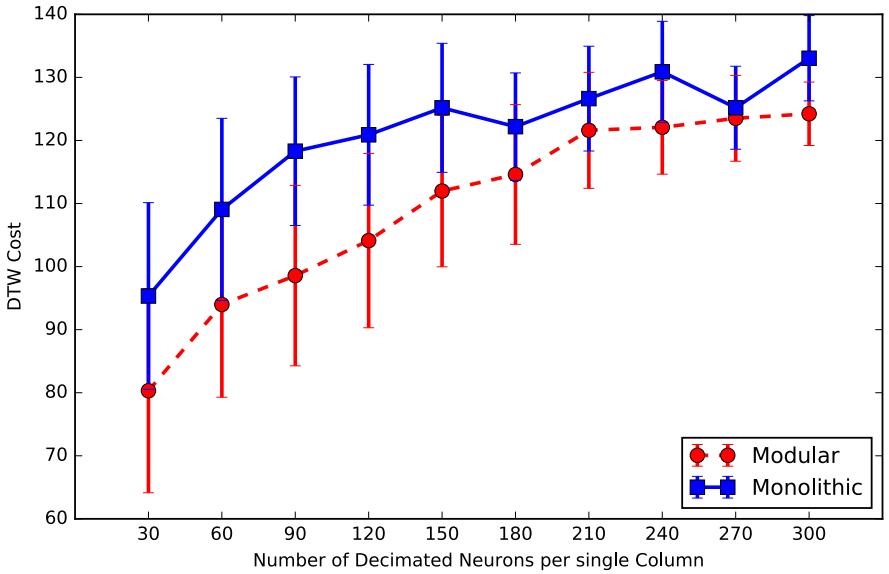


FIGURE 7.5: Decimated neurons, single column - Results from the experiments where a certain number of neurons were randomly decimated from only one (randomly chosen) of the five columns. The bars represent the standard error (total of 100 trials). Statistics are presented in the Table 7.3. The lower the DTW Cost, the better are the results.

7.3.1.4 Decimation of Columns

The idea, when the Modular system was initially presented in Chapter 5, was to be able to distribute it among several different pieces of hardware in order to have more inspiration from nature where diversity could help increasing the reliability. This would work similarly to a robotic system implemented using ROS [106] or YARP [138] - since they enable modularity, and one node would be responsible to aggregate the outputs from all columns (the *avg* node from Figure 7.1). In this situation, this aggregator node would be able to ignore a node that started behaving erratically. The results in Figure 7.7 and Table 7.4 show that case scenario.

A system following the Monolithic approach has its readout trained to receive inputs from all columns; therefore it fails even when only one column is disconnected. The Modular configuration is composed of stand-alone modules, consequently it withstands much better

TABLE 7.3: Welch's t-test results between Modular and Monolithic approaches - decimated neurons in a single column (100 trials)

Number of decimated neurons in a single column	*t-value*	*p-value*
30	**2.1544**	**0.0324**
60	**2.3023**	**0.0224**
90	**3.3502**	**0.0010**
120	**2.9712**	**0.0034**
150	**2.6311**	**0.0092**
180	1.7074	0.0894
210	1.2765	0.2033
240	**2.5349**	**0.0120**
270	0.5581	0.5774
300	**3.2891**	**0.0012**

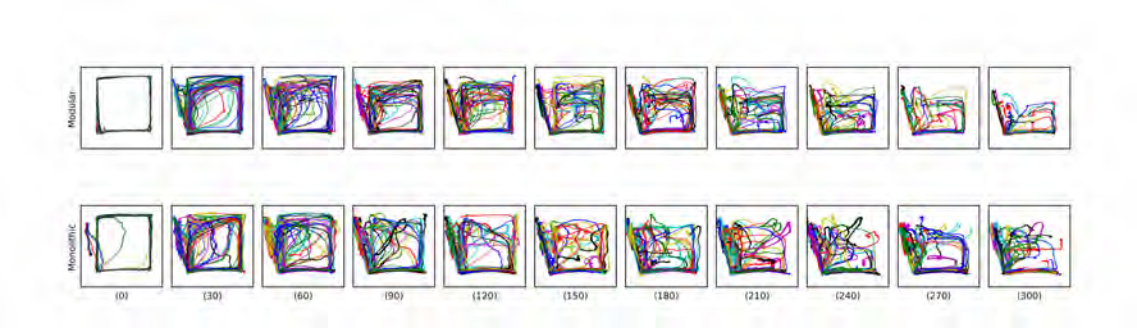


FIGURE 7.6: The final shapes that generated the DTW values in Figure 7.5 and Table 7.3 are presented here. It was also included one extra case were the columns were intact, therefore the number of decimated neurons was zero. The abscissa shows the Number of Decimated Neurons per single Column. All one hundred trials were plotted together.

the disconnection of nodes and beats the Monolithic case in all tested scenarios (Figure 7.7 and Table 7.4).

TABLE 7.4: Welch's t-test results between Modular and Monolithic approaches - decimated columns (100 trials)

Number of decimated columns	*t-value*	*p-value*
1	**38.7050**	**0.0000**
2	**28.3430**	**0.0000**
3	**29.9274**	**0.0000**
4	**21.9195**	**0.0000**

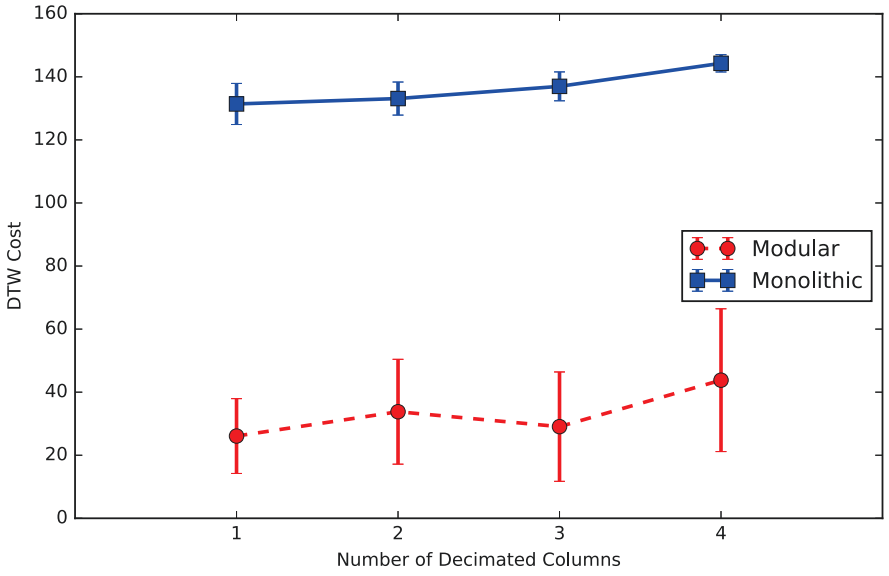


FIGURE 7.7: Decimated column - Results from the experiments where whole columns, randomly selected, were decimated. The bars represent the standard error (total of 100 trials). Statistics are presented in the Table 7.4. The lower the DTW Cost, the better are the results. The lower the DTW Cost, the better are the results.

7.4 Conclusions

Neurorobotic systems get inspiration from nature and also interact with the external world.

To accomplish this, it is at least necessary to withstand the inherent real-world uncertainty.

In this Chapter, it was presented that framework from Chapter 5 is not only biologically inspired, but also does not catastrophic fails when exposed to damage into its inner parts. Besides, it was tested in two different configurations in order to compare and contrast them in relation to the decimation of neurons and connections as an extension of what was already presented in Chapter 6.

Both approaches had quite similar behaviours when exposed to decimation in randomly chosen connections or neurons. However, the Modular one was able to better withstand decimation concentrated on a single column, or when an entire column was switched off, whilst the Monolithic catastrophic failed after only one column was completely removed.

Chapter 8

Conclusions

8.1 Overview

The main motivation for the work developed and presented in this thesis was the need to change the way humanoid robots are controlled by the use of ideas from neurorobotics. Therefore, new ways to process the computations necessary for commanding a robot were needed. However, it is important to highlight, this is not a single-man task and it was never planned to be solved in one thesis, too; therefore, the results obtained are extra steps on the direction of a bigger goal yet to come.

The reasons for trying a new way to process computations are many. Current technology is struggling to keep on track with market's expectation in relation to Moore's law, engineered approaches applied to solve high cognitive tasks have been proved incapable to scale up in part because of the curse of dimensionality and, lastly, current traditional digital solutions are fragile (can't degrade gracefully) and unable to recover from most defects or even deal with the unpredictability of real-world scenarios.

Mainstream silicon based technology is not capable to deliver for much longer the same increase in processing speed as it has to overcome some basic physical constraints; a transistor can not shrink to be smaller than an atom and the dissipated energy needs to be extracted somehow from the final circuit. Since a long time now, multi-purpose CPUs

are increasing the number of operations they can process per second mainly by expanding the number of cores in parallel and increasing the cache memory available. NVidia, the biggest player on specialized Deep Learning hardware, still employs the traditional model where memory and processing units are far from each other and they also try to bypass the problem with the use of cache memory. In academia, some projects as SpiNNaker, Neurogrid, Spikey and ROLLS were built to mimic more closely the brain, but they, as well, face the problem of the explosion of number of connections. In the last year or so, new players arrived. Two big companies, Google and Intel, introduced a tensor processing unit (TPU) and a new type of video processing unit (VPU) supporting deep learning frameworks, respectively. Cerebras (still a stealth mode start-up while this thesis was written), Graphcore and Wave Computing are all trying to change the paradigm to something more brain-like where multiple small units perform the computations achieving incredible power savings. However, the work presented in this thesis was based on Liquid State Machines and this makes it so flexible and innovative that it could be capable, in an extreme case, to use a bucket full of water as one of its processing unit.

The most popular examples of systems that not many years ago were dominated by engineered approaches are the Deep Convolutional Neural Networks applied to computer vision. All the deep network *flavours*, together with the Recurrent Neural ones, have reached state-of-the-art in many tasks. In spite of that, they are still implemented using deterministic computational units and are computationally very costly to train when compared to Reservoir Computing systems as Liquid State Machines.

Contrasting to man-made systems, nature designs have an innate ability to adapt and overcome failures while working in a noisy and unpredictable world. They also present another very important characteristic: they degrade gracefully as internal parts become faulty. The Neurorobotics approach employed in this thesis focused on mimicking nature implementations, up to a certain level, trying to reproduce some of those robustness aspects. All these was accomplished by the use of bio-inspired Liquid State Machines as the core of the robot arm controllers developed and presented through this thesis.

A Liquid State Machine, using LIF neurons, should have the capability to reproduce many

possible systems of differential equations, or filters. Cascading such systems of first order (integrators) would generate equivalent responses of higher orders. Considering its multiple internal feedback loops and with enough time, such a system could reproduce systems of any order. In addition, the LIF neuron has also a non-linearity that empowers it even further. Therefore, one could think of the readout as a pondered sum of those many possible systems and it would be seen as something behaving like the coefficients in a Fourier decomposition. Because of the fading memory characteristic of the Liquid State Machines, the ability to generate any pondered output would be limited, but feeding back the output of the Liquid State Machine to itself solves this problem and creates such a possibility of a programmable analog "like" computer. This idea was pursued and extended in this thesis generating some important contributions to knowledge.

Before the work presented in this thesis, Liquid State Machines were employed controlling robot arms only by Joshi and Maass back in 2005. However, in their work the controlled arm was a very simple two degrees of freedom, planar simulated one.

8.2 Summary of the Contributions to Knowledge

The key goal of the work presented in this thesis was the advancement of neurorobotics or brain inspired humanoid robot controllers. Moreover, this high level goal was initially divided in three main areas:

1. Generation of innovative new ways to control robots using a truly parallel approach instead of the classical serial paradigm.
2. Development of control systems that are able to better deal with noise or even take advantage of it.
3. Extension of the Liquid State Machine framework, using the learn by examples (action learning) methodology, to real-world robotic arm models with more than 2 degrees of freedom.

At first, this thesis explored a new approach to what was the current state-of-the-art in robot arm control using Liquid State Machines. That effort resulted in a new implementation of the work from Joshi and Maass based on Python and using an extention of the spiking neural network simulator Brian as presented in Chapter 3. Considering that previously the literature lacked a deep analysis of that controller for noise, decimation and importance of STP, this was a contribution that was partially published as paper presented at IJCNN 2017 [139].

Contrasting to what was presented by Joshi and Maass, from Chapter 4 onwards, instead of a simple two degree of freedom robot, a humanoid robot was always employed as the benchmark task. In Chapter 4, a simulated version of the collaborative humanoid robot Baxter was commanded by an adaptation of the system presented in Chapter 3 where the output of the reservoir was normalised by a gain and a bias. To the best of this author's knowledge, this was the first time a Liquid State Machine was employed controlling a humanoid robot or a collaborative robot like Baxter.

Following the work presented in Chapter 4, a novel idea based on the use of Liquid State Machine ensembles was introduced in Chapter 5. The Baxter robot was thought to draw, based on V-REP to simulate as if it was a human teacher, using the developed framework. It was capable to show an improvement on all the previous works on robot arm control using the same biologically inspired spiking neural network technology. Moreover, instead of controlling the robot to draw simple lines, this time, three different closed shapes (square, triangle and a circle) were employed as the benchmark task. The new task forced the system to be able to stop at the end of the task while the simulation would still run for some more time steps. This proved the implemented system was capable to follow the teacher's intructions and autonomously halt at the end. In addition, the literature traditionally made use of benchmarks that would not reflect the reality because they suffered with time delays and distortions introduced by averaging many results. In the same chapter was introduced the use of Dynamic Time Warping based on a new implementation mixing a Python wrapper and a C kernel. The use of the Dynamic Time Warping was important to generate a measurement that was closer to what us, humans, would expect. Also, since in Chapters 3 and 4 the simulations where based on Brian, a new specialized Liquid State

Machine Simulator, called BEE, was developed to speed up and prepare the system for real-world experiments. On top of that, to the best of this author's knowledge that was the first time a Liquid State Machine was employed using ensembles or commanding a robot to draw closed shapes or using the special normalisation presented in Chapter 5. The work presented in this chapter was previously presented at IJCNN 2016 [140].

Since the framework developed in Chapter 5 was intended for real-world action-learning applications, the work was focused on the problems encountered by traditional digital systems when operating in hostile environments. Therefore, chapters 6 and 7 were focused on experiments to verify the robustness of the robot controller to noise and destruction of its internal parts.

Starting in Chapter 6, the implemented Liquid State Machine robot controller from Chapter 5 was tested against noise in an attempt to simulate the effects of radiation and the generated soft-errors. Yet, to prove the use of ensembles had advantages when compared to the classical multi-column approach devised by Maass, both systems were compared. The solution introduced in this thesis was shown to better withstand noise in a certain range, being presented at ICONIP 2016 [141]. Although the literature had already works testing the robustness of Liquid State Machines, to the best of this author's knowledge that was the first time its robustness was tested in a more complex task as the robot arm control one used in Chapter 5.

In addition to soft-errors, the exposition of a digital system to an adverse environment can also have more destructive effects. In Chapter 7, both systems from Chapter 6, Modular and Monolithic, were tested in a way to simulated destructive events common to digital circuits like Single-Event Latchup, Single-Event Gate Rupture or Single-Event Burnout. The simulation was accomplished by the random deactivation of internal neurons, connections and whole columns (or liquids). The approach presented in Chapter 5 was capable to withstand all the aggressions degrading gracefully and better than simply using the multiple column system introduced by Maass. One more time, the results presented were not seen in the literature before considering a complex robot task. Chapter 7 was based on the work presented at IJCNN 2017 [142].

Finally, the work developed in this thesis was already extended by the author's co-supervised Brazilian, MSc. student at that time, Mr. Sala. This extension focused in the use of sensor fusion to avoid the problems that arise when the real robot tries to draw on a hard surface that is not perfectly aligned to the robot's plane. The novel results from this work were presented at I2MTC 2017 [143].

8.3 Suggestions of Future Works

Time was a big constraint to the work developed in this thesis, as it probably always happen with any PhD project. Therefore, many extra ideas were developed, yet not to the point to generate solid results or become part of the thesis. However, it becomes a treasure for possible future developments based on this thesis.

First, because the system presented in this thesis was only tested with tasks that kept the movement confined into the XY plan (the Z-axis was always kept constant), the next step is to apply the same ideas to more complex tasks. Considering that all the software necessary for real-world experiments was already developed, this could be done directly using the real robot and the reactions to the system according to changes in the scene could be studied.

From Chapter 5 onwards, the implemented controller was inherently capable to run in different machines in parallel. It would be interesting to verify the effects of using many different pieces of hardware instead of running all Liquid State Machines inside the same computer. In such setup, it could be verified if the system was capable to learn how to deal with different transmission delays between all the included parallel systems.

Also, since Baxter is a collaborative robot, it would be important to explore that with an experiment where the robot could interact with a person, maybe copying a simple industrial set-up that could easily match the closed shapes tested before, e.g. gluing. In order to be a collaborative task, the robots should need to autonomously wait until the user prompts it for help.

An interesting venue is the test of this system with soft-robots, robots that have more than six degrees of freedom or ones with variable stiffness. As long as an external teacher is capable to move the robot generating the necessary joint values, the system presented in this thesis could be employed to reproduce those as requested.

A Liquid State Machine has many internal parameters that are usually set beforehand (e.g. connection density, number of neurons, shape of the reservoir, spiking neuron parameters, etc). The work presented in this thesis always kept the parameters the same. Therefore, a possible extension of the work presented here would be to reproduce the same experiments testing the effects of different parameter sets.

Some experiments started, but not concluded, during this PhD were related to the generation of one thousand different reservoirs and employing them to the same task. Since they have a random component during their creation, the suggested idea is to use an approach like Monte Carlo to verify what characteristics would be best for each task. Additionally, it could be better tested the effects of having multiple liquids in parallel and if there is an optimal number of parallel systems for a certain task.

Finally, the implementation of spiking neural networks directly using SpiNNaker instead of a normal PC is the clear next step to verify if it is possible to increase the speed of the system enabling the use of more sophisticated scenarios. Some work was already done by the author in this direction in order to translate the implemented systems using the BEE simulator to SpiNNaker and it will only be necessary some extra optimisations to avoid the communication bottle necks related to receiving and sending spikes to external devices using the Ethernet port.

Appendix

Appendices 1 to 5 have been removed due to copyright restrictions.

Appendix 1:
R. de Azambuja, A. Cangelosi, and S.V. Adams. “Diverse, Noisy and Parallel: A New Spiking Neural Network Approach for Humanoid Robot Control.” In 2016 International Joint Conference on Neural Networks (IJCNN), 1134–42. Vancouver, 2016. doi:10.1109/IJCNN.2016.7727325.

Appendix 2:
R. de Azambuja, F. B. Klein, M. F. Stoelen, S. V. Adams, and A. Cangelosi. “Graceful Degradation Under Noise on Brain Inspired Robot Controllers.” In Neural Information Processing, edited by Akira Hirose, Seiichi Ozawa, Kenji Doya, Kazushi Ikeda, Minho Lee, and Derong Liu, 195–204. Lecture Notes in Computer Science 9947. Springer International Publishing, 2016. doi:10.1007/978-3-319-46687-3_21.

Appendix 3:
R. de Azambuja, D.H. García, M.F. Stoelen and A. Cangelosi. “Neurorobotic Simulations on the Degradation of Multiple Column Liquid State Machines.” In 2017 International Joint Conference on Neural Networks (IJCNN). Anchorage, 2017. doi:10.1109/IJCNN.2017.7965834.

Appendix 4:
R. de Azambuja, F.B. Klein, S.V. Adams, M.F. Stoelen and A. Cangelosi. “Short-Term Plasticity in a Liquid State Machine Biomimetic Robot Arm Controller.” In 2017 International Joint Conference on Neural Networks (IJCNN). Anchorage, 2017 doi:10.1109/IJCNN.2017.7966283.

Appendix 5:
D. A. Sala, V. J. Brusamarello, R. de Azambuja and A. Cangelosi “Positioning Control on a Collaborative Robot by Sensor Fusion with Liquid State Machines.” In Instrumentation and Measurement Technology Conference (I2MTC), 2017 IEEE International. Milano, 2017. doi:10.1109/I2MTC.2017.7969728.

Bibliography

[1] S. E. Kerns, B. D. Shafer, N. van Vonno, and F. E. Barber, "The design of radiation-hardened ICs for space: A compendium of approaches," *Proceedings of the IEEE*, vol. 76, no. 11, pp. 1470–1509, 1988.

[2] T. T. Rogers and J. L. McClelland, "Parallel Distributed Processing at 25: Further Explorations in the Microstructure of Cognition," *Cognitive Science*, vol. 38, pp. 1024–1077, Aug. 2014.

[3] S. Herculano-Houzel, "Scaling of Brain Metabolism with a Fixed Energy Budget per Neuron: Implications for Neuronal Activity, Plasticity and Evolution," *PLoS ONE*, vol. 6, p. e17514, Mar. 2011.

[4] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," tech. rep., University of Notre Dame, Sept. 2008.

[5] H. Poole, *Fundamentals of Robotics Engineering*. Springer Netherlands, 2012.

[6] A. Cangelosi, M. Schlesinger, and L. B. Smith, *Developmental Robotics: From Babies to Robots*. Cambridge, Massachusetts: The MIT Press, Jan. 2015.

[7] G. Sandini, G. Metta, and D. Vernon, "The icub cognitive humanoid robot: An open-system research platform for enactive cognition," in *50 Years of Artificial Intelligence*, pp. 358–369, Springer, 2007.

[8] E. Guigon, P. Baraduc, and M. Desmurget, "Computational Motor Control: Redundancy and Invariance," *Journal of Neurophysiology*, vol. 97, pp. 331–347, Jan. 2007.

[9] A. P. Georgopoulos, R. E. Kettner, and A. B. Schwartz, "Neuronal Population Coding of Movement Direction," *Science*, vol. 233, pp. 1416 –1419, 1986.

[10] T. Mergner and K. Tahboub, "Neurorobotics approaches to human and humanoid sensorimotor control," *Journal of Physiology-Paris*, vol. 103, pp. 115–118, May 2009.

[11] F. Kaplan, "Neurorobotics: An experimental science of embodiment," *Frontiers in Neuroscience*, p. 23, 2008.

[12] D. V. Buonomano and W. Maass, "State-Dependent Computations: Spatiotemporal Processing in Cortical Networks," *Nature Reviews Neuroscience*, vol. 10, pp. 113–125, Feb. 2009.

[13] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge university press, 2002.

[14] M. A. Nugent and T. W. Molter, "AHaH Computing–From Metastable Switches to Attractors to Machine Learning," *PLoS ONE*, vol. 9, p. e85175, Feb. 2014.

[15] S. Dura-Bernal, G. L. Chadderdon, S. A. Neymotin, X. Zhou, A. Przekwas, J. T. Francis, and W. W. Lytton, "Virtual musculoskeletal arm and robotic arm driven by a biomimetic model of sensorimotor cortex with reinforcement learning," Dec. 2013.

[16] A. Bouganis and M. Shanahan, "Training a spiking neural network to control a 4-DoF robotic arm based on spike timing-dependent plasticity," in *Neural Networks (IJCNN), The 2010 International Joint Conference On*, pp. 1–8, 2010.

[17] D. Gamez, R. Newcombe, O. Holland, and R. Knight, "Two simulation tools for biologically inspired virtual robotics," in *Proceedings of the IEEE 5th Chapter Conference on Advances in Cybernetic Systems, Sheffield*, pp. 85–90, 2006.

[18] T. E. Milner, "A model for the generation of movements requiring endpoint precision," *Neuroscience*, vol. 49, no. 2, pp. 487–496, 1992.

[19] F. A. Mussa-Ivaldi, S. F. Giszter, and E. Bizzi, "Linear combinations of primitives in vertebrate motor control," *Proceedings of the National Academy of Sciences*, vol. 91, no. 16, pp. 7534–7538, 1994.

[20] J. Wessberg, C. R. Stambaugh, J. D. Kralik, P. D. Beck, M. Laubach, J. K. Chapin, J. Kim, S. J. Biggs, M. A. Srinivasan, and M. A. L. Nicolelis, "Real-time prediction of hand trajectory by ensembles of cortical neurons in primates," *Nature*, vol. 408, pp. 361–365, Nov. 2000.

[21] W. Maass, T. Natschläger, and H. Markram, "Real-Time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural computation*, vol. 14, pp. 2531–2560, Nov. 2002.

[22] K. P. Dockendorf, I. Park, P. He, J. C. Príncipe, and T. B. DeMarse, "Liquid State Machines and Cultured Cortical Networks: The Separation Property," *Biosystems*, vol. 95, pp. 90–97, Feb. 2009.

[23] W. Maass, T. Natschläger, and H. Markram, "Fading Memory and Kernel Properties of Generic Cortical Microcircuit Models," *Journal of Physiology-Paris*, vol. 98, pp. 315–330, July 2004.

[24] W. Maass, P. Joshi, and E. D. Sontag, "Computational Aspects of Feedback in Neural Circuits," *PLoS Comput Biol*, vol. 3, p. e165, Jan. 2007.

[25] H. Hauser, R. M. Füchslin, and K. Nakajima, "The physical body as a computational resource," in *Opinions and Outlooks on Morphological Computation* (H. Hauser, R. M. Füchslin, and R. Pfeifer, eds.), ch. 20, pp. 226–244, 2014.

[26] P. Joshi and W. Maass, "Movement Generation with Circuits of Spiking Neurons," *Neural Computation*, vol. 17, no. 8, pp. 1715–1738, 2005.

[27] H. Hazan, "The Liquid State Machine is not robust to problems in its components but topological constraints can restore robustness," pp. 258–264, SciTePress - Science and and Technology Publications, 2010.

[28] E. T. Rolls and G. Deco, *The Noisy Brain: Stochastic Dynamics as a Principle of Brain Function*, vol. 28. Oxford university press New York, 2010.

[29] M. A. Lewis and T. J. Klein, "Neurorobotics Primer," in *The Path to Autonomous Robots*, pp. 1–25, Springer, 2009.

[30] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, pp. 2454–2467, Dec. 2013.

[31] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, no. 3, pp. 251–265, 1988.

[32] A. K. Seth, O. Sporns, and J. L. Krichmar, "Neurorobotic models in neuroscience and Neuroinformatics," *Neuroinformatics*, vol. 3, pp. 167–170, Sept. 2005.

[33] J. Krichmar, "Neurorobotics," *Scholarpedia*, vol. 3, no. 3, p. 1365, 2008.

[34] V. J. Marsick and J. O'Neil, "The Many Faces of Action Learning," *Management Learning*, vol. 30, pp. 159–176, June 1999.

[35] T. Ziemke, "Are robots embodied," in *First International Workshop on Epigenetic Robotics Modeling Cognitive Development in Robotic Systems*, vol. 85, Citeseer, 2001.

[36] S. Schaal, "Is imitation learning the route to humanoid robots?," *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[37] H. Burgsteiner, "Imitation learning with spiking neural networks and real-world devices," *Engineering Applications of Artificial Intelligence*, vol. 19, pp. 741–752, Oct. 2006.

[38] Y. Demiris and A. Meltzoff, "The robot in the crib: A developmental analysis of imitation skills in infants and robots," *Infant and Child Development*, vol. 17, pp. 43–53, Jan. 2008.

[39] S. Haykin, *Neural Networks: A Comprehensive Foundation*. International edition, Prentice Hall, 1999.

[40] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, pp. 1659–1671, Dec. 1997.

[41] S. Ablameyko, *Neural Networks for Instrumentation, Measurement and Related Industrial Applications*. NATO Science Series, IOS, 2003.

[42] F. Alnajjar and K. Murase, "Sensor-fusion in spiking neural network that generates autonomous behavior in real mobile robot," in *IEEE International Joint Conference on Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence)*, pp. 2200–2206, June 2008.

[43] E. Nichols, L. J. McDAID, and N. H. Siddique, "Case Study on a Self-organizing Spiking Neural Network for Robot Navigation," *International Journal of Neural Systems*, vol. 20, pp. 501–508, Dec. 2010.

[44] A. Cyr and M. Boukadoum, "Classical conditioning in different temporal constraints: An STDP learning rule for robots controlled by spiking neural networks," *Adaptive Behavior*, vol. 20, pp. 257–272, June 2012.

[45] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke, "Evolving spiking neural network controllers for autonomous robots," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference On*, vol. 5, pp. 4620–4626, IEEE, 2004.

[46] X. Wang, Z.-G. Hou, A. Zou, M. Tan, and L. Cheng, "A behavior controller based on spiking neural networks for mobile robots," *Neurocomputing*, vol. 71, pp. 655–666, Jan. 2008.

[47] S. Adams, T. Wennekers, G. Bugmann, S. Denham, and P. Culverhouse, "Application of arachnid prey localisation theory for a robot sensorimotor controller," *Neurocomputing*, vol. 74, pp. 3335–3342, Oct. 2011.

[48] R. Batllori, C. Laramee, W. Land, and J. Schaffer, "Evolving spiking neural networks for robot control," *Procedia Computer Science*, vol. 6, pp. 329–334, Jan. 2011.

[49] A. Jimenez-Fernandez, G. Jimenez-Moreno, A. Linares-Barranco, M. J. Dominguez-Morales, R. Paz-Vicente, and A. Civit-Balcells, "A Neuro-Inspired Spike-Based PID Motor Controller for Multi-Motor Robots with Low Cost FPGAs," *Sensors (Basel, Switzerland)*, vol. 12, pp. 3831–3856, Mar. 2012.

[50] P. Trhan, "The Application of Spiking Neural Networks in Autonomous Robot Control," *Computing and Informatics*, vol. 29, no. 5, pp. 823–847, 2012.

[51] L. I. Helgadottir, J. Haenicke, T. Landgraf, R. Rojas, and M. P. Nawrot, "Conditioned behavior in a robot controlled by a spiking neural network," in *Neural Engineering (NER), 2013 6th International IEEE/EMBS Conference On*, pp. 891–894, IEEE, 2013.

[52] U. Markowska-Kaczmar and M. Koldowski, "Spiking neural network vs multilayer perceptron: Who is the winner in the racing car computer game," *Soft Computing*, pp. 1–14, Dec. 2014.

[53] S. Y. Nof, *Handbook of Industrial Robotics*. John Wiley & Sons, 1999.

[54] J. M. Winters and S. L.-Y. Woo, *Multiple Muscle Systems: Biomechanics and Movement Organization*. Springer Science & Business Media, Dec. 2012.

[55] "Towards Autonomous, Adaptive, and Context-Aware Multimodal Interfaces: Theoretical and Practical Issues," in *Third COST 2102 International Training School, Caserta, Italy, March 15-19, 2010, Revised Selected Papers* (A. Esposito, A. M. Esposito, R. Martone, V. C. Müller, G. Scarpetta, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, and G. Weikum, eds.), vol. 6456 of *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[56] T. N. Aflalo and M. S. A. Graziano, "Relationship between Unconstrained Arm Movements and Single-Neuron Firing in the Macaque Motor Cortex," *Journal of Neuroscience*, vol. 27, pp. 2760–2780, Mar. 2007.

[57] A. J. Bastian, "Learning to predict the future: The cerebellum adapts feedforward movement control," *Current Opinion in Neurobiology*, vol. 16, pp. 645–649, Dec. 2006.

[58] M. M. Shanechi, R. C. Hu, and Z. M. Williams, "A cortical–spinal prosthesis for targeted limb movement in paralysed primate avatars," *Nature Communications*, vol. 5, Feb. 2014.

[59] A. B. Schwartz, "Cortical Neural Prosthetics," *Annual Review of Neuroscience*, vol. 27, no. 1, pp. 487–507, 2004.

[60] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, pp. 164–171, July 2006.

[61] J. K. Chapin, "Using multi-neuron population recordings for neural prosthetics," *Nature Neuroscience*, vol. 7, pp. 452–455, May 2004.

[62] J. K. Chapin, K. A. Moxon, R. S. Markowitz, and M. A. Nicolelis, "Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex," *Nature neuroscience*, vol. 2, no. 7, pp. 664–670, 1999.

[63] B. V. Benjamin, Peiran Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102, pp. 699–716, May 2014.

[64] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling," *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS"10)*, pp. 1947–1950, 2010.

[65] M. Noack, J. Partzsch, C. G. Mayr, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schüffny, "Switched-Capacitor Realization of Presynaptic Short-Term-Plasticity and Stop-Learning Synapses in 28 Nm CMOS," *Frontiers in Neuroscience*, vol. 9, Feb. 2015.

[66] P. Rowcliffe and J. Feng, "Training Spiking Neuronal Networks With Applications in Engineering Tasks," *IEEE Transactions on Neural Networks*, vol. 19, pp. 1626–1640, Sept. 2008.

[67] R. R. Carrillo, E. Ros, C. Boucheny, and O. J.-M. Coenen, "A real-time spiking cerebellum model for learning robot control," *Biosystems*, vol. 94, pp. 18–27, Oct. 2008.

[68] Q. Wu, T. M. McGinnity, L. Maguire, A. Belatreche, and B. Glackin, "2D co-ordinate transformation based on a spike timing-dependent plasticity learning mechanism," *Neural Networks*, vol. 21, pp. 1318–1327, Nov. 2008.

[69] E. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?," *IEEE Transactions on Neural Networks*, vol. 15, pp. 1063–1070, Sept. 2004.

[70] G. L. Chadderdon, S. A. Neymotin, C. C. Kerr, and W. W. Lytton, "Reinforcement Learning of Targeted Movement in a Spiking Neuronal Model of Motor Cortex," *PLoS ONE*, vol. 7, p. e47251, Oct. 2012.

[71] N. Srinivasa and Y. Cho, "Self-Organizing Spiking Neural Model for Learning Fault-Tolerant Spatio-Motor Transformations," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, pp. 1526–1538, Oct. 2012.

[72] S. Dura-Bernal, G. L. Chadderdon, S. A. Neymotin, J. T. Francis, and W. W. Lytton, "Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm," *Pattern Recognition Letters*, vol. 36, pp. 204–212, May 2013.

[73] S. A. Neymotin, G. L. Chadderdon, C. C. Kerr, J. T. Francis, and W. W. Lytton, "Reinforcement Learning of Two-Joint Virtual Arm Reaching in a Computer Model of Sensorimotor Cortex," *Neural Computation*, vol. 25, pp. 3263–3293, Sept. 2013.

[74] C. Casellato, A. Antonietti, J. A. Garrido, R. R. Carrillo, N. R. Luque, E. Ros, A. Pedrocchi, and E. D'Angelo, "Adaptive Robotic Control Driven by a Versatile Spiking Cerebellar Network," *PLoS ONE*, vol. 9, p. e112265, Nov. 2014.

[75] M. Hulea and C. Caruntu, "Spiking neural network for controlling the artificial muscles of a humanoid robotic arm," in *System Theory, Control and Computing (ICSTCC), 2014 18th International Conference*, pp. 163–168, Oct. 2014.

[76] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems.* Computational neuroscience, Cambridge, Mass: MIT Press, 2003.

[77] S. Menon, S. Fok, A. Neckar, O. Khatib, and K. Boahen, "Controlling articulated robots in task-space with spiking silicon neurons," in *Biomedical Robotics and Biomechatronics (2014 5th IEEE RAS EMBS International Conference On*, pp. 181–186, Aug. 2014.

[78] W. Maass, "Noise as a Resource for Computation and Learning in Networks of Spiking Neurons," *Proceedings of the IEEE*, vol. 102, pp. 860–880, May 2014.

[79] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-Based Strategies for Rapid Processing," *Neural networks*, vol. 14, no. 6, pp. 715–725, 2001.

[80] S. V. Adams, A. D. Rast, C. Patterson, F. Galluppi, K. Brohan, J.-A. Pérez-Carrasco, T. Wennekers, S. Furber, and A. Cangelosi, "Towards Real-World Neurorobotics: Integrated Neuromorphic Visual Attention," in *Neural Information Processing*, pp. 563–570, Springer, 2014.

[81] H. Markram, "A history of spike-timing-dependent plasticity," *Frontiers in Synaptic Neuroscience*, vol. 3, 2011.

[82] M. Tsodyks and S. Wu, "Short-Term Synaptic Plasticity," *Scholarpedia*, vol. 8, no. 10, p. 3153, 2013. revision #136920.

[83] W. Maass and H. Markram, "Synapses as Dynamic Memory Buffers," *Neural Networks*, vol. 15, no. 2, pp. 155 – 161, 2002.

[84] Z. Rotman, P.-Y. Deng, and V. A. Klyachko, "Short-Term Plasticity Optimizes Synaptic Information Transmission," *Journal of Neuroscience*, vol. 31, pp. 14800–14809, Oct. 2011.

[85] H. Markram, Y. Wang, and M. Tsodyks, "Differential Signaling via the Same Axon of Neocortical Pyramidal Neurons," *Proceedings of the National Academy of Sciences*, vol. 95, no. 9, pp. 5323–5328, 1998.

[86] G. Mongillo, O. Barak, and M. Tsodyks, "Synaptic Theory of Working Memory," *Science*, vol. 319, pp. 1543–1546, Mar. 2008.

[87] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, Sept. 1995.

[88] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[89] L. Pape, J. de Gruijl, and M. Wiering, "Democratic Liquid State Machines for Music Recognition," in *Speech, Audio, Image and Biomedical Signal Processing Using Neural Networks* (B. Prasad and S. Prasanna, eds.), vol. 83 of *Studies in Computational Intelligence*, pp. 191–215, Springer Berlin Heidelberg, 2008.

[90] H. Ju, J.-X. Xu, and A. M. VanDongen, "Classification of musical styles using liquid state machines," in *Neural Networks (IJCNN), The 2010 International Joint Conference On*, pp. 1–7, IEEE, 2010.

[91] E. Goodman and D. Ventura, "Spatiotemporal Pattern Recognition via Liquid State Machines," in *Neural Networks, 2006. IJCNN '06. International Joint Conference On*, pp. 3848–3853, 2006.

[92] A. Oliveri, R. Rizzo, and A. Chella, "An application of spike-timing-dependent plasticity to readout circuit for liquid state machine," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference On*, pp. 1441–1445, IEEE, 2007.

[93] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated Word Recognition with the Liquid State Machine: A Case Study," *Information Processing Letters*, vol. 95, pp. 521–528, Sept. 2005.

[94] F. Schürmann, K. Meier, and J. Schemmel, "Edge of Chaos Computation in Mixed-Mode Vlsi-a Hard Liquid," in *Advances in Neural Information Processing Systems*, pp. 1201–1208, 2004.

[95] W. Maass and H. Markram, "On the Computational Power of Circuits of Spiking Neurons," *Journal of Computer and System Sciences*, vol. 69, pp. 593–616, Dec. 2004.

[96] D. J. Watts and S. H. Strogatz, "Collective Dynamics of 'small-World' Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[97] D. S. Bassett and E. Bullmore, "Small-World Brain Networks," *The Neuroscientist*, vol. 12, pp. 512–523, Dec. 2006.

[98] M. D. McDonnell and L. M. Ward, "The Benefits of Noise in Neural Systems: Bridging Theory and Experiment," *Nature Reviews Neuroscience*, vol. 12, no. 7, pp. 415–426, 2011.

[99] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in Artificial Life*, pp. 588–597, Springer, 2003.

[100] B. Jones, D. Stekel, J. Rowe, and C. Fernando, "Is there a liquid state machine in the bacterium escherichia coli?," in *Artificial Life, 2007. ALIFE'07. IEEE Symposium On*, pp. 187–191, IEEE, 2007.

[101] B. J. Grzyb, E. Chinellato, G. M. Wojcik, W. Kaminski, and others, "Facial expression recognition based on liquid state machines built of alternative neuron models," in *Neural Networks, 2009. IJCNN 2009. International Joint Conference On*, pp. 1011–1017, IEEE, 2009.

[102] R. Veale and M. Scheutz, "Neural circuits for any-time phrase recognition with applications in cognitive models and human-robot interaction," in *Proceedings of the 34th Annual Conference of the Cognitive Science Society, N. Miyake, D. Peebles, and RP Cooper, Eds. Austin, TX: Cognitive Science Society*, pp. 1072–1077, 2012.

[103] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[104] J. Steil, "Backpropagation-decorrelation: Online recurrent learning with O(N) complexity," in *2004 IEEE International Joint Conference on Neural Networks, 2004. Proceedings*, vol. 2, pp. 843–848 vol.2, July 2004.

[105] P. F. Dominey, "Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning," *Biological cybernetics*, vol. 73, no. 3, pp. 265–274, 1995.

[106] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, p. 5, 2009.

[107] G. A. Pratt and M. M. Williamson, "Series elastic actuators," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 1, pp. 399–406 vol.1, Aug. 1995.

[108] E. Rohmer, S. P. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference On*, pp. 1321–1326, IEEE, 2013.

[109] G. Rossum, "Python Reference Manual," tech. rep., CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, 1995.

[110] D. F. M. Goodman, "The Brian Simulator," *Frontiers in Neuroscience*, vol. 3, pp. 192–197, Sept. 2009.

[111] F. Pérez and B. E. Granger, "IPython: A System for Interactive Scientific Computing," *Computing in Science and Engineering*, vol. 9, pp. 21–29, May 2007.

[112] R. de Azambuja, "BEE - The Spiking Reservoir Simulator." `https://github.com/ricardodeazambuja/BEE`, 2017.

[113] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, pp. 43–49, Feb. 1978.

[114] Meinard Müller, "Dynamic time warping," in *Information Retrieval for Music and Motion*, pp. ch. 4, 69 – 82, New York: Springer-Verlag, 2007.

[115] R. Ratcliff, *Continuous versus Discrete Information Processing: Modeling Accumulation of Partial Information.* Master thesis, Radboud University Nijmegen, Nijmegen, The Netherlands, 2004.

[116] R. de Azambuja, "DTW - Dynamic Time Warping in Python / C." `https://github.com/ricardodeazambuja/DTW`, 2017.

[117] T. Flash and N. Hogan, "The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model," *The Journal of Neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

[118] E. Jones, T. Oliphant, P. Peterson, and others, *SciPy: Open Source Scientific Tools for Python.* 2001–. [Online; accessed 2015-06-02].

[119] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-Learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[120] C. M. Bishop, "Training with Noise Is Equivalent to Tikhonov Regularization," *Neural Computation*, vol. 7, pp. 108–116, Jan. 1995.

[121] D. Pecevski, T. Natschläger, K. Schuch, D. Pecevski, T. Natschläger, and K. Schuch, "PCSIM: A parallel simulation environment for neural circuits fully integrated with Python," *Frontiers in Neuroinformatics*, vol. 3, p. 11, 2009.

[122] J. M. Stern, *Atlas of EEG Patterns.* Lippincott Williams & Wilkins, Mar. 2013.

[123] A. Pogosyan, L. D. Gaynor, A. Eusebio, and P. Brown, "Boosting Cortical Activity at Beta-Band Frequencies Slows Movement in Humans," *Current Biology*, vol. 19, pp. 1637–1641, Oct. 2009.

[124] Y. I. Arshavsky, T. G. Deliagina, and G. N. Orlovsky, "Pattern generation," *Current Opinion in Neurobiology*, vol. 7, pp. 781–789, Dec. 1997.

[125] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, pp. 642–653, May 2008.

[126] E. P. Zehr and J. Duysens, "Regulation of Arm and Leg Movement during Human Locomotion," *The Neuroscientist*, vol. 10, pp. 347–361, Aug. 2004.

[127] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, "Towards a theoretical foundation for morphological computation with compliant bodies," *Biological Cybernetics*, vol. 105, pp. 355–370, Dec. 2011.

[128] S. V. Adams and C. M. Harris, "A Computational Model of Innate Directional Selectivity Refined by Visual Experience," *Scientific Reports*, vol. 5, p. 12553, July 2015.

[129] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A Digital Liquid State Machine With Biologically Inspired Learning and Its Application to Speech Recognition," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–1, 2015.

[130] R. M. Rifkin and R. A. Lippert, "Notes on regularized least squares," 2007.

[131] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *Neural Networks, IEEE Transactions on*, vol. 6, no. 2, pp. 446–456, 1995.

[132] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. SAÏGHI, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Neuromorphic Engineering*, vol. 5, p. 73, 2011.

[133] M. M. Waldrop, "More than Moore," *Nature*, vol. 530, Feb. 2016.

[134] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "Near-threshold voltage (NTV) design: Opportunities and challenges," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 1153–1158, ACM, 2012.

[135] F. K. Chowdhury, D. Choe, T. Jevremovic, and M. Tabib-Azar, "Design of MEMS based XOR and AND gates for rad-hard and very low power LSI mechanical processors," in *2011 IEEE Sensors*, pp. 762–765, Oct. 2011.

[136] H. Hazan and L. M. Manevitz, "Topological constraints and robustness in liquid state machines," *Expert Systems with Applications*, vol. 39, pp. 1597–1606, Feb. 2012.

[137] G. D. Ruxton, "The Unequal Variance T-Test Is an Underused Alternative to Student's T-Test and the Mann–Whitney U Test," *Behavioral Ecology*, vol. 17, pp. 688–690, Jan. 2006.

[138] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[139] R. de Azambuja, F. B. Klein, S. V. Adams, M. F. Stoelen, and A. Cangelosi, "Short-term plasticity in a liquid state machine biomimetic robot arm controller," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3399–3408, May 2017.

[140] R. de Azambuja, A. Cangelosi, and S. Adams, "Diverse, Noisy and Parallel: A New Spiking Neural Network Approach for Humanoid Robot Control," in *2016 International Joint Conference on Neural Networks (IJCNN)*, (Vancouver), pp. 1134–1142, July 24-29 2016.

[141] R. de Azambuja, F. B. Klein, M. F. Stoelen, S. V. Adams, and A. Cangelosi, "Graceful Degradation Under Noise on Brain Inspired Robot Controllers," in *Neural Information Processing* (A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, eds.), no. 9947 in Lecture Notes in Computer Science, pp. 195–204, Springer International Publishing, Oct. 2016.

[142] R. de Azambuja, D. H. García, M. F. Stoelen, and A. Cangelosi, "Neurorobotic simulations on the degradation of multiple column liquid state machines," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 46–51, May 2017.

[143] D. A. Sala, V. J. Brusamarello, R. de Azambuja, and A. Cangelosi, "Positioning control on a collaborative robot by sensor fusion with liquid state machines," in *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–6, May 2017.