Biologically Inspired Goal Directed Navigation For Mobile Robots

Presented by:
Paul Omondi Amayo

Prepared for:
Robyn Verrinder
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Masters of Science degree

May 24, 2016

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signed by candidate

Signature:.....

Paul Omondi Amayo

12/06/2016

Date:.....

Acknowledgments

To my supervisor, Robyn Verrinder, for her invaluable guidance and help throughout the project.

To Prof. Alireza Baghai-Wadji for his help and insight into algorithm development.

To my parents and brothers for their continued support and encouragement throughout my life.

To my friends from MRF, Rhodes, Linacre, Strathmore, Cape Town who have become family you have my undying gratitude.

To Muthoni, Coralie and Gilbert for not giving up even after I had.

To God for enabling me to come this far.

Abstract

This project involved an investigation into low-cost navigation of mobile robots with the aim of creating and adaptive navigation system inspired by behaviour seen in animals. The navigation module developed here would need to be able to successfully localise a robot and navigate it to a defined target. A critical literature review was carried out of current localisation and path-planning architectures and a bio-inspired approach using an Echo State Network and Liquid State Machine architecture was chosen as the base for the navigation modules. The navigation module implemented in this work is trained to navigate and localise itself in different environments drawing its inspiration from the behaviour of small rodents.

These architectures were adapted for use by a robot with a view on the physical implementation of these architectures on an embedded low-cost robot using a Raspberry Pi computer. This robot was then built using low-cost,noisy proximity sensors which formed the inputs to the navigation modules. Before the deployment on the embedded robot the system was tested and validated in a full physics simulator.

While the training of the Echo State Networks and Liquid State Machine has been carried out in the literature by the offline method of linear regression, in this work we introduce a novel way of training these networks that is online using concepts from adaptive filters. This online method increases the adaptability of this system while significantly decreasing its memory requirements making it very attractive for low-cost embedded robots.

The end result from the project was a functioning navigation module using an Echo State Network that was able to navigate the robot to a target position as well as learn new paths, either using offline or online methods. The results showed that the Echo State Network approach was valid both in simulation and practically as a base for creating navigation modules for low-cost robots and could also lead to more efficient and adaptable robots being developed if the training was carried out in an online manner. The increased computational complexity of implementing the liquid State machine on analytical machines however made it unsuitable for deployment on robots using embedded micro-controllers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to the study

Autonomous systems have long been thought to hold the key to solving a number of the earth's problems. One of the key motivations for development of these systems in recent years has been repeatability of performance. This has benefits ranging from increased road safety through self-driving cars, improved outputs in dangerous work-spaces such as mines and high-precision surgery work in the medical field. The repeatability of performance is distinctly different from biological systems where drops in performance due to fatigue are commonly observed.

This does not mean that these autonomous and biological systems are completely divorced from each other as the increased understanding of how humans and biological systems work has led to various breakthroughs especially in the field of robotic locomotion and actuation. There is however significantly less overlap when the field of goal-directed navigation in mobile robots is considered. Animals have been exhibiting very long-range goal-directed navigation for reasons of seasonal migration in birds, returning to a food source in the case of small insects in ants and most importantly the exploration of unknown environments. Such performance has only been recently reported in robots but this is mainly in very high-cost systems relying on very accurate sensor systems and operating in similar environments.

This places a large limitation on mobile robotic systems, especially those that operate in dynamic environment and the previously stated performance-repeatability guarantees no longer hold. The performance also depends largely on highly accurate systems that

are in turn very expensive, although current research has been able to lower these costs, limiting the applicability of such autonomous systems to many real-world situations.

This is antithetical to biology where animals with "low-accuracy sensors" are able to exhibit goal-directed navigation under a large range of conditions, with an added benefit of increased performance when the same task is performed multiple times.

In this study, we try and incorporate some of the techniques that can be attributed to nature for goal-directed navigation of low-cost robotics in an attempt to utilise the benefits provided by techniques in the biological and robotics realm. By abstracting these techniques we are able to deploy and test these systems at work in a simulated and physical experiment to validate their plausibility as a tool for low-cost robot navigation.

1.2 Objectives of this study

1.2.1 Problems to be investigated

The main areas to be investigated in this project are:

- The biological basis of neural networks and the steps taken to create artificial neural networks
- The development of a controller for a small robot based on biologically-based neural networks
- The performance of this controller using two different biologically-based network methodologies
- The different training methods available for these networks
- The performance of these networks on an embedded low-cost physical platform

1.2.2 Purpose of the study

These problems need to be investigated so that:

- Conclusions can be drawn about the validity of using biologically-based neural networks for navigation in small mobile robots.
- Recommendations can be made for the future development of biologically-based neural networks used in navigation of low-cost robots.

1.3 Scope and Limitations

The project was completed in the space of one year. The main focus of the project was to test the biologically based neural networks on a simulated as well as physical robots. The study does not make any claims on the biological.This project therefore does not aim to introduce new theory but uses state of the art techniques in the development, testing and deployment of these networks.

1.4 Plan of development

The report begins in Chapter 2 with a critical literature review of the main strategies that have been proposed for navigation in mobile robotics. It then goes on to give a brief review of the biologically-based neural networks that are used in this work.

This is followed in Chapter 3 by the theoretical background which showcases the steps taken in developing these neural networks from the neurons seen in animals. Two different learning paradigms for these networks are also described here.

Chapter 4 details the experiments taken to test these networks both from a simulation point of view and also from the physical testing.

Chapter 5 presents the results obtained both simulated and experimental. The results are then discussed in Chapter 6 before conclusions are drawn and recommendations made about them in Chapter 7 and 8 respectively.

Chapter 2

Literature Review

In this literature review, we investigate the problem of goal-directed navigation and specifically its sub-goals. Of particular interest is the difference between how these are implemented in biological systems and in the robotics literature which was investigated by Milford and Schulz [1]. Following this we then investigate different approaches taken into applying biological models in robotic systems before outlining the proposed approach.

2.1 Goal-directed navigation

Goal directed navigation as it is used in this work refers to the series of tasks by which a robot or animal is able to decide on a certain target, plan or recall a route from its current location and then execute the motion along this route [1].

This series of tasks can be further broken down into two questions for any agent, be it biological or robotic. The first is "Where am I?" and subsequently: "How do I get to where I want to go?" Some proposed solutions to these two questions as presented in the literature are discussed below.

2.1.1 Where am I? Localisation in a World Representation

Localisation as its name alludes to seeks to determine the position of an agent within a particular environment. Closely tied to this is the internal representation of the

environment used by the agent. Different internal representations lead to different localisation goals. Our review begins with those systems used in robotic systems before moving to those in biological systems.

Most advanced robotics systems use multiple representations of the environment, where the standard modus operandi is the combination of a global topoloigcal map with a local metric map to create a hybrid topometric map [2, 3, 4]. Topological maps represent the environment through recognisable places or features in the environment and connections between them without necessarily reflecting the precise geometric information between them [5]. Metric maps on the other hand describe the environment through its precise geometric information.

One of the important representations used in the local metric space of robotic maps is the occupancy grid [6]. These have been developed due to the presence of range-finding sensors such as lasers commonly found in robots, they represent the environment through a grid of locations that are either obstacles or free-space with an associated probability as shown in Figure 2.1. The generation of these representations has been a major research

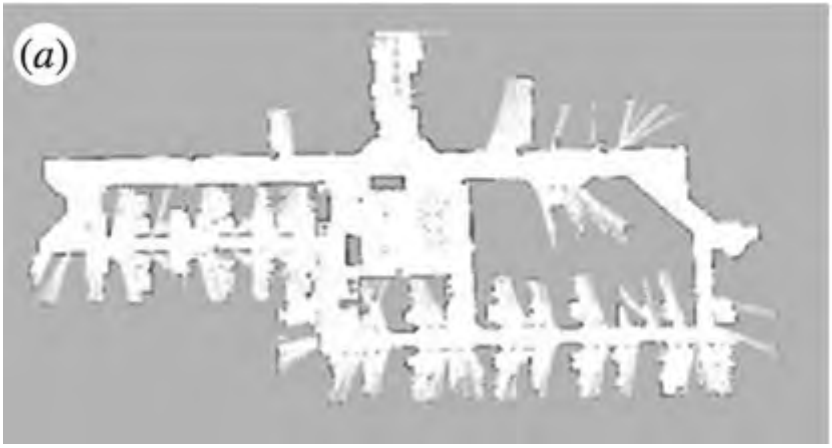


Figure 2.1: Figure showing an occupancy grid. [1]

area of robotics and has largely led to the idea of localisation being subsumed into the process of mapping, which was a key proponent of the Simultaneous Localisation And Mapping(SLAM) systems that dominated robotics research in the past 10 years or so. A short analysis of how these systems were developed is presented in the following section.

Simultaneous Localisation And Mapping

Within robotic research the issue of uncertainty from sensors has played a key role in the development of these technologies. This is because with perfect sensing of the

environment, simple techniques such as dead-reckoning would be sufficient for localisation. In the dead-reckoning approach all new positions are measured relative to the initial starting position. Once the robot is moving the velocity is measured and stored. To get the present position the velocity is integrated over time which results in the current position being calculated [7].

With zero uncertainty in the sensors, the precise location of the robot can be retrieved and by collating sensor data with previous positions an accurate representation of the environment can be reconstructed. This is however not the case in most robotic systems as sensors have uncertainty and by the nature of this approach errors tend to grow without bound over time as there is no inbuilt correction mechanism. This method is therefore suited for mostly slow moving objects whose velocities do not change rapidly and often. It also accumulates errors when going over uneven terrain [7].

The approach taken to finding this solution in the robotics community is largely probability based and it involves estimating robot's trajectory(localisation) and the position of landmarks (mapping). This approach, along with its subsequent equations, described by Durrant-Whyte et al. [8] in the tutorial to SLAM is presented below.

Durrant-Whyte et al. [8] described the SLAM problem as one that asks *"if it is possible for a mobile robot to be placed in an unknown environment and for the robot to incrementally build a consistent map of the environment while simultaneously determining the location within this map."* The solution to this would allow for robots to navigate autonomously.

The solution to the SLAM problem requires the computation of the following probability distribution.

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{2.1}$$

Where

- $\mathbf{x}_k$ is the current location of the robot.
- $\mathbf{m} = \{m_1, m_2, \cdots, m_n\}$ is the set containing all the landmark positions in the map.
- $\mathbf{Z}_{0:k} = \{z_1, z_2, \cdots, z_k\}$ is the set of all landmark observations.
- $\mathbf{U}_{0:k} = \{u_1, u_2, \cdots, u_k\}$ is the set containing the history of the control inputs to the robot.

A recursive solution to this probability distribution is usually sought and this gives two models. If the vehicle position and map are known, a distribution that describes the

probability of observing a landmark can be derived. This gives the observation model as below:

$$P(\mathbf{z}_k|, \mathbf{m}, \mathbf{x}_k) \tag{2.2}$$

The second model is the motion model, which gives the location of the robot, if the previous position of the car and the control input to the car are known. It follows that the new location of the robot is independent of the map and any observations giving the motion model as below.

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.3}$$

Durrant-Whyte et al. [8] then showed that the SLAM algorithm can be implemented in a two step recursive approach. Beginning with a time-update step.

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0)\mathrm{d}\mathbf{x}_{k-1} \tag{2.4}$$

Followed by a update on the measurements observed:

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{u}_k)P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{Z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})} \tag{2.5}$$

With these two equations a solution to calculating the joint posterior $P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)$ is provided. This solution gives the robot position and the map at time k. If the map is known with certainty then a localisation solution is one that calculates the vehicle's position in relation to the map. This can be done by computing the distribution $P(\mathbf{x}_k|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0, \mathbf{m})$.

To efficiently implement the SLAM algorithm as shown above a computationally-efficient representation of both the motion and observation models must be found. This has been implemented by Dissanayake et al. [9] using a state-space model in the case of Extended Kalman Filter-SLAM (EKF-SLAM), or using a particle filter as in the FastSLAM implementation seen in the work of Montomerlo et al. [10].

While the methods above present a thorough mathematical representation for localisation, what is missing from the studies above is that they assume high-accuracy sensors with a low covariance and uncertainty from the beginning which is often not the case. The higher the uncertainty and covariance, the more computational effort would be needed to properly estimate the location of the robot.

This translates to a higher cost of the overall system because you either pay for more accurate sensors or for more computational power and as the optimisation adage goes: *"There is no free lunch"* [11].

Biological systems

While considering biological systems it is of particular significance to this work those particular systems from biology that can be mimicked in robotic technology or so called biomimetic models [1].

Within this subset of biological models one of the most significant correlations between biology and robotics is found in rodents where evidence suggests that they encode space in a manner similar to occupancy grids. This is through *boundary vector cells* (BVCs) which have been shown to fire at a particular range and orientation from an environmental boundary [12].

An attempt to recreate this behaviour in robots was reported in the paper by Antonelo et al. [13] in which they mimicked the behaviour of a small rodent to localise a small robot. They mimicked the behaviour of place cells which are neurons in a rat's hippocampus that respond maximally to certain locations in the rat's environment [14]. The mimicry was done by using the biologically realistic implementation of recurrent neural networks called the Echo State Networks (ESN) [15].

In their experiment Antonelo et al. designed an environment that can be seen in Figure 2.2. This room environment contained 30 different locations designated as points together with potential paths joining some of these points together.

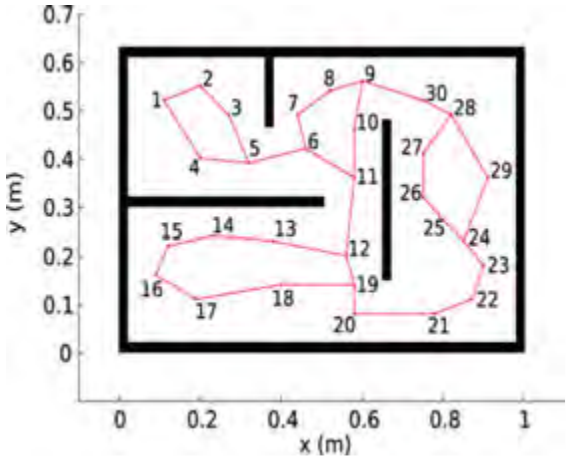


Figure 2.2: Figure showing the room environment used in the localisation experiment. Different locations in the environment were denoted by numbers (1-30), and red lines represent possible paths through the locations. (Taken from [13])

In the experiment a simulated robot was driven through the room environment from one point to another using a controller and the lines between the locations were used as paths for the robots. Whenever a point was reached that led to two other points, one of the

points was chosen by the controller with equal probability. The task of the Echo State Network was to predict which of the 30 locations the robot was closest to.

Once the Echo State Network was trained using a supervised learning algorithm it was able to perform the localisation task. The output of the networks for one of the experiments can be seen in the occupancy grid in Figure 2.3 below.

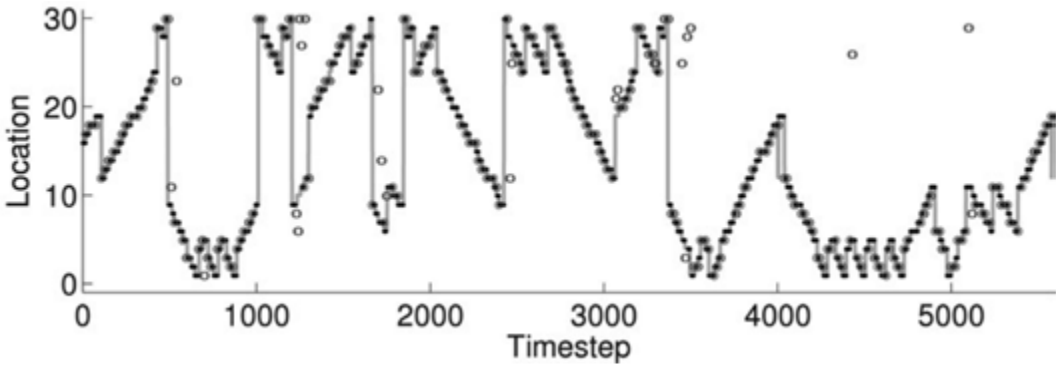


Figure 2.3: Figure showing the location detection performed by the Echo State Network. The gray solid line represents the ground truth for the location. Shaded circles are correctly predicted locations while mispredictions are marked with clear circles. [13]

The network had an accuracy of 84% on the test data given, as can be seen in the occupancy grid, which for the room environment can be considered as successfully localising itself.

Several models based on place cells in the rodents hippocampus have also been seen in the literature. Long-term potentiation (LTP) was used to learn temporal and spatial information about a rats motion by modifying inter-place cell synaptic strengths. When a simulated rat traversed a familiar route, a delay in LTP induction strengthened links between pre-synaptic cells and postsynaptic cells further along the path. Over time, place cell activity started to anticipate the rats future location along this route [16].

Attractor-based models have also been used for real and simulated robots. The RatSLAM rodent-inspired model encodes the spatial layout of space with a single-scale attractor model of grid cells [17]. Similarly landmark based models in combination with dead reckoning have been seen in the literature in attempts at localisation[18].

2.1.2 How do I get to where I want to go? Route Planning and Execution

Once successful localisation has taken place in an environment, the next task for a robot would be path-planning. Path-planning can be broadly defined as the task of getting the optimal sequence of robotic configurations that result in movement from the robot's present position to a predefined goal [19]. This sequence thus forms a path through a workspace. How optimal this path is depends on some application-specific criteria including distance covered, time taken, manoeuvres used, obstacle avoidance as well as computation effort.

In the following section three different approaches to path-planning that are found in the literature are discussed.

Search algorithms

Graph search algorithms are one of the oldest methods of finding a minimum cost path between two positions. They allocate costs to each point in the search space referred to as nodes. The costs could be in terms of Cartesian distance, time travelled or any other criteria chosen. By expanding from the start node through the minimum cost nodes, a path can be generated from the start to the goal node.

The cost to a node is given by the equation below where $g(n)$ is the cost from the start to the current node and $h(n)$ is an underestimate of the cost from the present node to the goal.

$$\text{Cost to present node } f(n) = g(n) + h(n).$$

These methods, such as the popular Dijkstra algorithm [20] have their roots firmly in mathematics although they have been modified over the years by the robotics community. Hart et al. [21] created the A* graph search algorithm at Stanford University for path-planning for terrestrial robots. The A* algorithm has been successfully used in path-planning applications for mobile robots as it easily encodes static obstacles. A* however has poor performance when dynamic obstacles are considered as it performs re-planning in an inefficient manner. The D* algorithm addresses this problem and is now widely used in path-planning methods for robots [22].

Potential Field Methods

Potential field methods are modelled on electrostatic forces. In this approach paths are generated by running a gradient descent on a global potential field that is generated in the workspace [23]. The goal is considered to have an attractive potential while any obstacles are considered repulsive.

The potential function is the sum of these positive and negative forces as can be seen in the Figure 2.4 below.

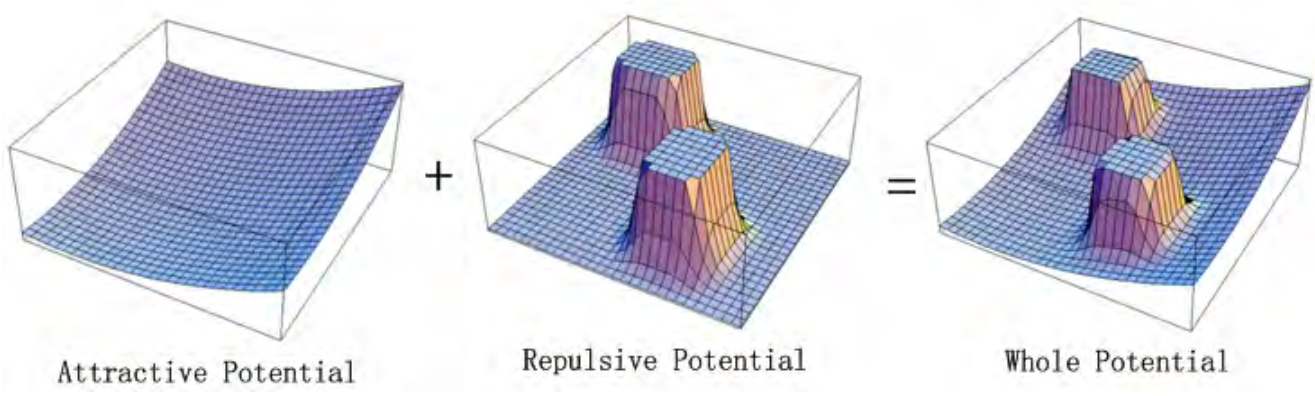


Figure 2.4: Figure showing an example of the potential field algorithm. The first image shows the attractive potential field provided by the goal in the bottom left corner. Costs will decrease as the goal is approached. The second image in turn gives the repulsive potential generated by objects in the path. The final image shows the total potential and gradient descent can be run on this to find a minimum cost path. (Taken from [23])

The path is generated by following the path of the negative gradient which corresponds to the direction which moves towards the goal the fastest and compared to the search algorithms is computationally simpler because no searching occurs.

Teach and Repeat methods

The teach and repeat methodology offers a different approach from most of the path-planning systems found in the literature in that it has two distinct stages. In the first stage, the teach stage, the robot is presented with a path to a goal from a certain starting position and in the repeat stage the robot is made to repeat this path to the goal. These methods are agnostic to how the original path is created in the teach section. Therefore any of the traditional path-planning methods as well as human-aided methods such as steering the robot or some kind of line-following can be used to teach the robot.

What the repeat stage does is eliminate the need for any further explicit path-planning

to be carried out in an area where planning has occurred before. This comes with a huge computational benefit as path-planning, especially in low-cost robots, requires a huge chunk of computational power. This computational power is not always available when the robot is performing other tasks during normal operation which generally leads to sub-optimal performance due to task scheduling. Task scheduling occurs when multiple tasks are competing for limited computational power leading to some of them being prioritised over others. This translates to sub-optimal performance of the system. The reduction in computational cost by using teach and repeat therefore has real implications in the robots operation. Firstly it frees up computational power that can be used for other tasks increasing the overall system performance. Secondly its performance in planning remains similar as the same path taught is implemented. Two different ways of implementing teach and repeat methodologies are presented below.

Antonelo and Schrauwen [24] extended their work of localisation using an Echo State Network (ESN) to path-planning. They developed and trained two ESN's on a room environment, the first ESN performed the localisation task while the second ESN performed navigation by giving outputs to the motors of the robot.

The second ESN was trained using a controller that moved the robot from an initial position in the room environment shown in Figure 2.5 to a goal position in any of the other rooms.

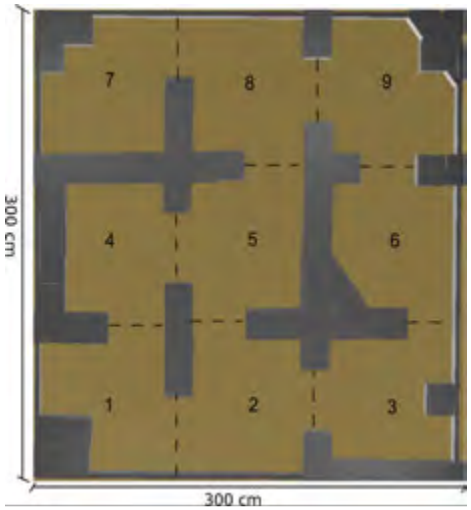


Figure 2.5: Figure showing the room environment used for the path-planning. (Taken from [24].)

The two ESN's were then trained using gathered data. Once the robot was returned to its initial position the ESN was able to navigate it to a goal as can be seen in Figure 2.6 below.

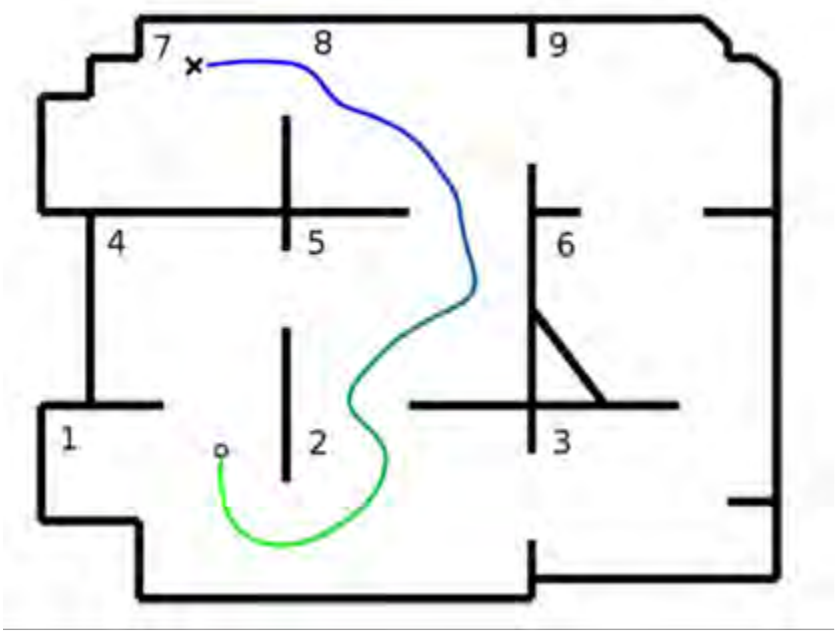


Figure 2.6: The rooms plan with the trajectory navigated by the robot shown by the green/blue line using the ESN from room 1 to 7. The circle represents the starting point while the x represents the final position. (Taken from [24].)

The navigation ESN was able to reconstruct the path through the environment that was given by the controller and could be thought of as 'memorising' the paths taught to it. During the execution stage of this experiment the ESN effectively functions as a path planner that reconstructs paths from memory.

Berenson et al. [25] proposed a similar path-planning model that learnt from experience. Although in their case they used a library to store paths that had already been created and then retrieved them using a special module using some defined heuristics. This approach leads to faster paths being produced by the model for the two simulations they carried out.

2.1.3 Proposed Approach

As was stated previously a bio-inspired approach to navigation of mobile robots is taken in this work following the same premise reported by Antonelo et al. [13].

The SLAM approach to localisation, though considered essentially a solved problem in structured indoor environments, is not only computationally complex but also requires expensive and quite accurate sensors to be accurate. This, while suitable for large robots, makes it difficult to implement effectively in small low-cost mobile robots. The bio-

inspired approach however has been shown to work effectively on small robots with low-cost and low-accuracy sensors [13].

For path-planning a teach and repeat approach was adopted, using one of the traditional path-planning methods to construct plausible paths in an environment and then teaching the robot to repeat these paths using a neural network. While the approach by Berenson et al. [25] offers similar results to neural networks the use of a library makes it difficult to implement in a low cost embedded system.

Therefore neural networks, specifically those with the same architecture as the Echo State Network, were chosen to implement the navigation system of a small low-cost mobile robot. The following section serves as introduction to this architecture commonly known as reservoir computing [26].

2.2 Reservoir Computing

Reservoir computing is a fairly new approach to training recurrent neural networks. This approach first came to light through the work of Jaeger [15] as well as that of Maass et al. [27] who invented this idea independently. The Echo State Network proposed by Jaeger had its roots in machine learning while the Liquid State machine was firmly footed in computational neuroscience.

The reservoir computing architecture can be seen in Figure 2.7.

The idea behind reservoir computing is that because a recurrent neural network contains certain generalised properties, it is not necessary to train all the weights in the network to produce a specific output. Training of a memoryless readout from the network suffices to give good performance. This means that only the weights from the reservoir to the output layer of the network shown in Figure 2.7 need be trained. The training of recurrent neural networks before this, had proved to be almost always non-converging and even when they did converge it was slow, computationally expensive and difficult to tune which greatly limited their use [28]. New approaches such as convolutional neural networks and deep learning have been able to improve the training process but this comes at a significant design cost as compared to the traditional recurrent networks [29].

The two different variations of reservoir computing reported in the literature are discussed in the section below.

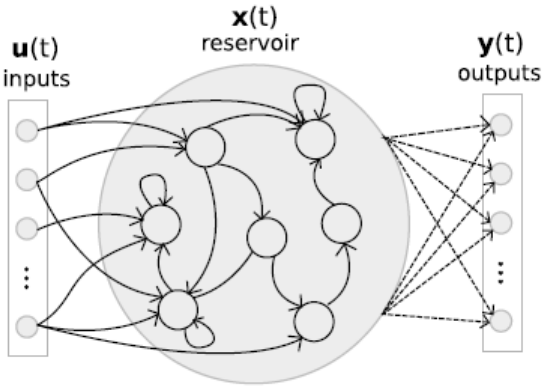


Figure 2.7: Figure showing the reservoir computing architecture. This consists of three layers, an input layer, reservoir layer and output layer. The input layer has connections to the reservoir layer, with the reservoir layer having connections to the output layer as well as connections internally. Only the weights of the connections from the reservoir layer to the output shown in dotted lines are trained in this architecture.

2.2.1 Echo State Network

The Echo State network is built up of artificial or firing rate neurons. From a mathematical perspective firing rate neurons are numbers therefore an echo state network maps its inputs which are numbers to its outputs, other numbers. It is defined by the following equations as described by Jaeger [15].

$$\mathbf{x}(n+1) = f\big(\mathbf{W}_{in}\mathbf{u}(n+1) + \mathbf{W}_{res}\mathbf{x}(n)\big) \tag{2.6}$$

In equation 2.6, the inputs to the network at time *n+1* are represented by $\mathbf{u}(n+1)$ with the matrix representing the weighted connections between the input and the network represented by $\mathbf{W}_{in}$. The states of the reservoir shown in Figure 2.7 at time *n+1* are given by $\mathbf{x}(n+1)$, with the recurrent connections between nodes in the reservoir described by the matrix $\mathbf{W}_{res}$. The function $f(\cdot)$ is a nonlinear activation function for the neurons.

It is necessary, however, for the reservoir to have an echo state or fading memory property. This implies that if the inputs are the same then the reservoir states should converge after some time period irrespective of the previous history. This ensures that the reservoir states do not grow without bound. It can be shown that if the largest absolute eigenvalue (spectral radius) of the system is smaller than one, then the reservoir has a fading memory and fulfils the echo state property. For most applications best performance is achieved when the spectral radius is just less than one $|\lambda_{max}| = 0.98$ [26].

The output of the network is given by the following equation which is a linear combination of the reservoir states.

$$\mathbf{y}(n+1) = \mathbf{W}_{out}\mathbf{x}(n+1) \tag{2.7}$$

To train the network an optimum value of weights $\mathbf{W}_{out}$ that minimise the squared error between the predicted output and the real output should be found. This training can de done easily using techniques such as linear regression that compute this optimum weight from the data directly. It must be noted that this is a type of supervised learning and takes place offline, therefore a set of training inputs and outputs must be available before learning takes place.

Echo State Network Applications

Due to the ease of their training and relative ease to set up, Echo State networks have found applications in several areas. They have been used in speech recognition applications [30], robot motor control [31], medical [32] and financial applications [33] with success comparable to the state of the art in all these areas.

2.2.2 Liquid State Machine

The liquid state machine was developed for modelling computations in cortical microcircuits in the brain as an alternative to traditional methods such as the Turing machines [34]. Compared to the Echo State Network which maps numerical values to numerical values due to its use of artificial neurons, the Liquid State Machine maps continuous time inputs into continuous time outputs using biologically realistic models of neurons. These inputs $u(t)$ are commonly spike trains with the outputs $y(t)$ going into output spike trains. Therefore, mathematically, the LSM acts as a functional, operator or a filter. The liquid state machine however still functions based on the reservoir principle which allows only the weights from the reservoir to be trained. This formulation guarantees universal computational power under certain idealised conditions [27].

Maass et al. [27] described the operation of the liquid state machine as follows. A function of time injected into the liquid filter L^M, creates at time t, the liquid state $x^M(t)$ which can then by transformed by a memoryless readout map f^M to generate an output $y(t)$. This operation is shown in Figure 2.8.

Maass et al. [27] then showed that a Liquid State Machine can be used to approximate

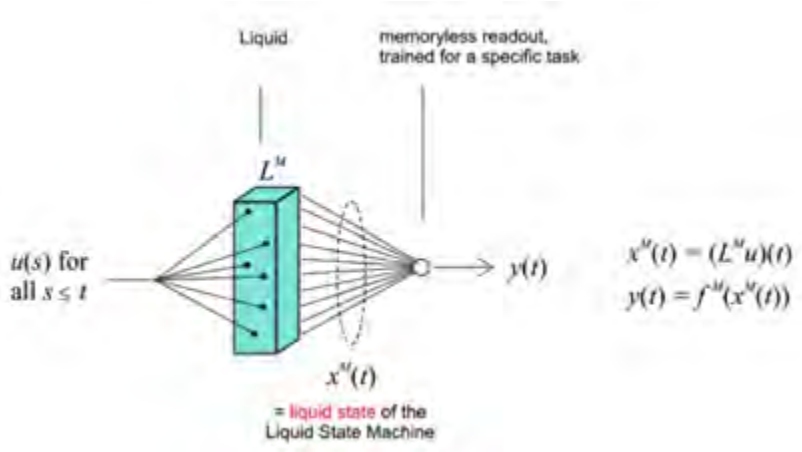


Figure 2.8: Figure showing the Liquid State Machine architecture as described by Maass et al. [27]. When a function of time $u(t)$ is injected into the liquid state machine L^M it creates liquid states $x^M(t)$. These can be transformed by a memoryless readout map to a task-specific output $y(t)$. (Taken from [34])

any time-invariant fading memory filter provided it has the following properties. The first property is the pointwise separation property. If there are two different input streams $u(\cdot)$, $v(\cdot)$ where $u(s) \neq v(s)$ for $s \leq t$, a filter B has the pointwise separation property if $(Bu)(t) \neq (Bv)(t)$. The second property needed is the approximation property, which has that there must be a rich enough pool from which the readout functions should be chosen which ensures that any continuous function on a compact domain can be approximated from this pool. Maass et al. [27] go further to show that if a Liquid State Machine, fulfilling the approximation and separation property, is enhanced with feedback from its outputs it then acquires universal computation power.

The liquid state machine although used widely in computational neuroscience to better understand how the brain works has found some application in various engineering tasks such as speech recognition [35] and robotics [36].

2.3 Proposed approach

The approach taken in this study is to implement both the Liquid State Machine and the Echo State Network as biologically realistic neural networks for the purpose of path-planning of a small low-cost embedded robotic platform.

Chapter 3

Theoretical background

In this chapter, the reader is presented with the mathematical constructs used in this work and how they were developed. Particular attention is placed on the biologically based neural networks as well as the simplification arguments used to create the networks used in this work. This chapter builds up the theory mathematically from the literature review before developing some new theory that can be used in our networks.

3.1 Neurons

In most mammals the nervous system consists of the network of interconnected neural cells or neurons [37]. Neurons communicate using electrical signals commonly referred to as *spikes*, they receive signals from other neurons through the *dendrites* and then transmit their emitted signal through the *axon*. The transmission of these electrical impulses is through specialised transmitters located at the boundaries of the axon of one neuron and the dendrite of the next neuron called *synapses*. An example of a neuron is shown in Figure 3.1 showing two neurons their axons as well as the dendrites and a synapse.

A synapse represents a connection between two neurons that facilitates the transfer of information between them. The synapse therefore divides the two neurons it connects into two categories, the post-synaptic neurons and the pre-synaptic neurons. The classification is based on the direction of information flow through the synapse, the neuron that transmits a message through a synapse is called the pre-synaptic neuron while the neuron that receives this signal through the synapse is referred to as the post synaptic neuron.

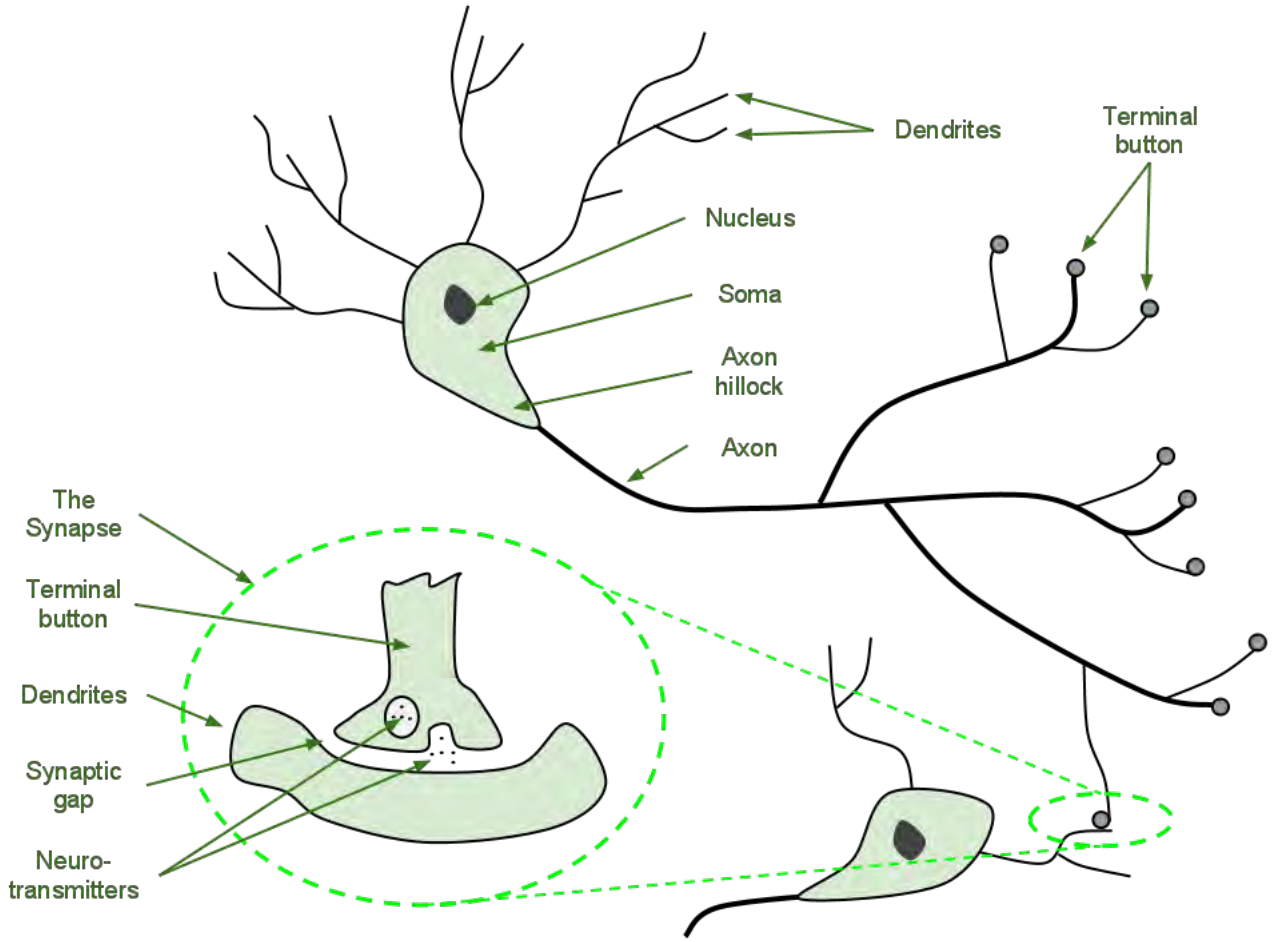


Figure 3.1: Figure showing two biological neurons. The connection between two neurons via the dendrites, the synapse, is highlighted in the dotted green ellipse. The axon of the neuron facilitates the transfer of signals. (Taken from [38])

At the synaptic point incoming messages from the pre-synaptic neurons cause the release of chemical substances known as *neurotransmitters*. Neurotransmitters allow for certain ions to flow into the dendrite of a postsynaptic neuron. These in turn cause a voltage difference between the external environment and the interior of the neuron termed as the neuron's *potential*. If this potential exceeds a certain threshold an electrical signal is propagated along its axon. This signal is referred to as an *action potential* or spike and the process can be seen in Figure 3.2.

One of the most important features of synapses is that they have the ability to change the strength of their connections. This is as a single neuron is connected to multiple input neurons via the synapses. These presynaptic neurons do not necessarily affect the postsynaptic neuron in the same way therefore there is a difference in the synaptic strength across the synapses. This strength changes according to the output needed leading to a characteristic commonly referred to as synaptic plasticity [37]. This is what gives the neurons the ability to learn and the basis for memory in mammalian nervous

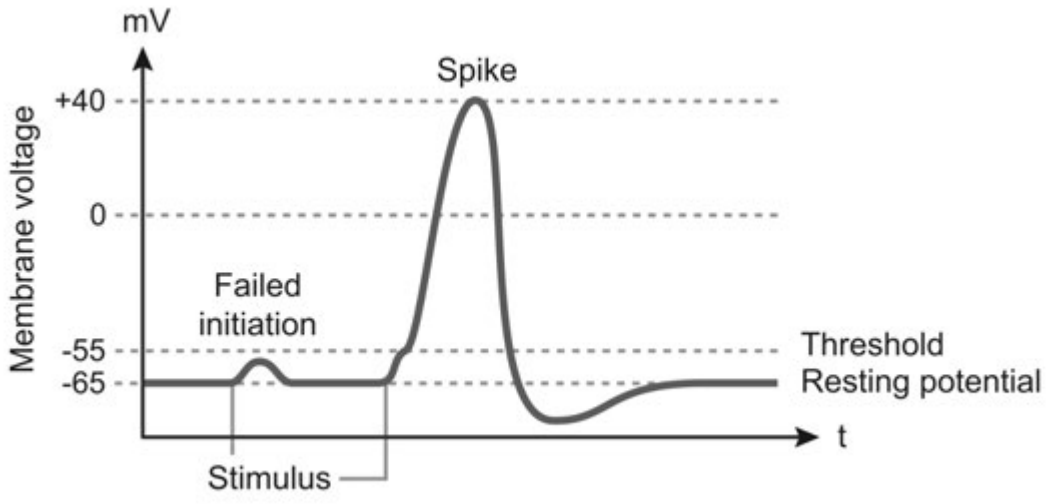


Figure 3.2: Figure showing membrane dynamics. When the membrane voltage exceeds a certain threshold a spike is generated and propagated through the neuron. (Taken from [37])

systems. Synapses can also either cause the post-synaptic neurons to spike more, in this case they will be excitatory synapses, or to spike less in which case they will be inhibitory synapses.

The following sections describe some of the models that have been generated to not only describe neurons and neural networks but to enable their use in software and hardware that attempt to utilise features of biological neural networks.

3.1.1 Neuron Models

The theoretical development of neuron behaviour begins with the Hodgkin Huxley model which is considered as one of the most complete models for neuron behaviour, before going into the several abstractions that were made to create more simple models.

Hodgkin Huxley Neural Model

The Hodgkin Huxley model was one of the first models that was presented to understand neural behaviour [39]. After experiments conducted on a neuron of a squid, Hodgkin and Huxley proposed a mathematical model that quantitatively accounted for the current that flowed through the cell membrane of a neuron.

They proposed that this current, caused mainly by the flow of ions both into and out

of the cell, was responsible for spike/action potential generation in neurons as well as propagation. This current was composed of Sodium (Na) ions, Potassium (K) ions as well as a leakage (L) element. From examination of the membrane current it was seen that the conductance of the sodium and potassium ions does not change according to a change in the total membrane current but rather due to a change in the membrane potential. This change was so strong that a depolarisation of the cell membrane by a few milivolts would lead to an exponential increase in the conductance of sodium ions [39].This led Hodgkin and Huxley to postulate that the transfer of sodium as well as potassium ions occurred due to the activation of certain charged particles in the neuron. These particles, when activated, would allow ions to pass through the membrane. Therefore the movement of these charged particles would affect the rate at which conductance reaches its maximum but do not affect the maximum itself. From this analysis they were able to propose a model that accounts for most of the activity of the neuron [39]. Their analysis begins from the following equivalent circuit that models the neuron membrane as shown in Figure 3.3.

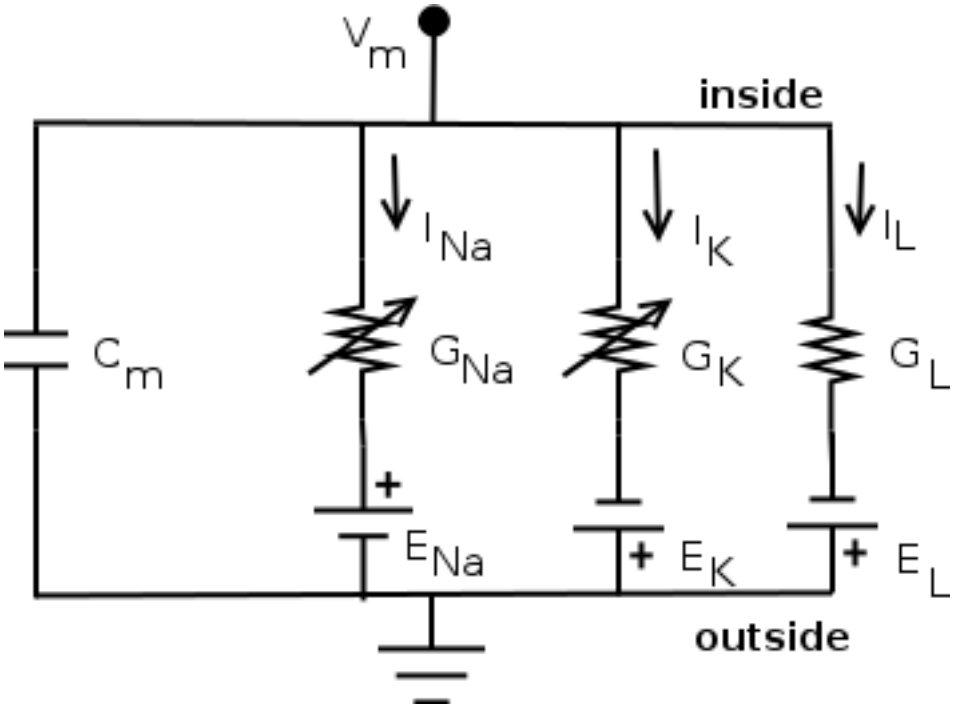


Figure 3.3: Figure showing the equivalent circuit of a cell membrane as postulated by the Hodgkin Huxley model [39]. The membrane has a voltage V_m and capacitance C_m associated with it. Current into the cell comes from three sources sodium ions I_{Na}, potassium ions I_K and a leakage current I_L all controlled by their respective conductances G_{Na}, G_K, G_L and potentials E_{Na}, E_K, E_L.

Leading to the following equation:

$$\frac{1}{\theta^2}\frac{\mathrm{d}^2V}{\mathrm{d}t^2} = C_m\frac{\mathrm{d}V}{\mathrm{d}t} + \bar{g}_K n^4(V - V_k) + \bar{g}_{Na} m^3 h(V - V_{Na}) + \bar{g}_l(V - V_l) \qquad (3.1)$$

This is an ordinary differential equation which can be solved using numerical methods. The solution of this generated spikes/action potentials that closely resembled the ones found experimentally. This therefore created a good computational model for neuron behaviour.

The difficulty of solving an ordinary differential equation however limited its practical use in software or hardware designed to utilise any of the features provided by neural networks. A significant abstraction that was taken in neuron modelling to overcome this is described in the following section.

Integrate And Fire Model

The integrate and fire (IAF) neuron is the one of the most widely used models for spiking neurons [40]. Unlike the Hodgkin Huxley model that concentrated on the biophysical properties of the neuron this model instead focuses on the basic dynamics of neurons by performing leaky integration. This is that for a neuron, once the membrane reaches a certain threshold, a spike or action potential is generated. In the IAF model the neuron is modelled by a resistance R_m and capacitance C_m in parallel receiving an external current I_e as shown in Figure 3.4. The neuron's membrane voltage is referred to as V in the Figure and equations below.

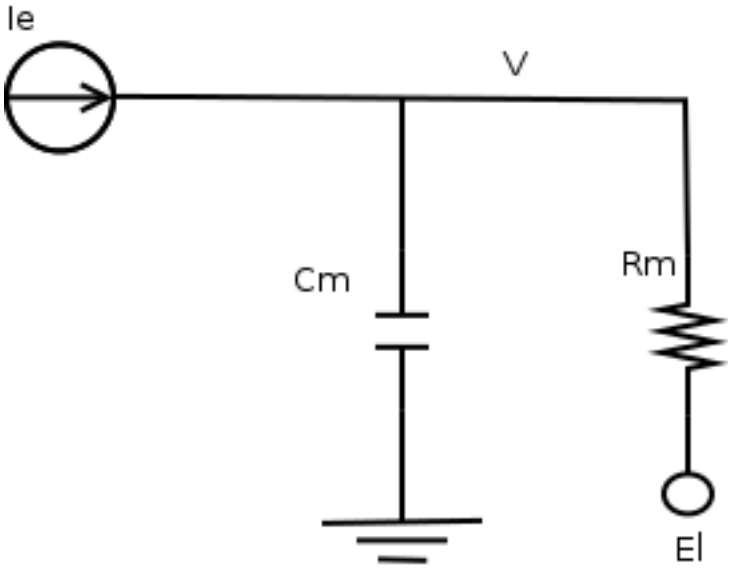


Figure 3.4: Figure showing the equivalent circuit of cell membrane given by the IAF neuron. The cell membrane has a voltage V, capacitance C_m and resistance R_m associated with it as well as a leakage potential E_l. The current I_e through the membrane is considered as coming from an external source.

The derivations presented below of the operation of the IAF neuron together with its

corresponding governing equations are fully described in [40]. Their analysis of this begins with the governing equation for the IAF neuron as:

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -\frac{V - E_l}{R_m} - I_e \tag{3.2}$$

Following this a time constant can be introduced $\tau_m = R_m C_m$ for the membrane and this simplifies to give equation 3.3 below.

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_l) - I_e R_m \tag{3.3}$$

However two additional constraints on the system are added:

1. If $V \rightarrow V_{th}$ a spike is fired.
2. Then $V \rightarrow V_{reset}$.

In this model the membrane voltage is summed up until it reaches a particular threshold value given as V_{th}. When this voltage is reached, a spike is fired from the neuron and the membrane voltage is reset to a value given by V_{reset}. These two constraints capture the basic dynamics of the neuron and lead to a simpler model for the spiking neuron that has been widely used in the literature.

The input current to the neuron however has two sources, the first one being the one directly injected into the neuron by its environment and the second one is the one brought about by the synapses. This is shown in Figure 3.5 below.

The case of a single synapse as shown in the Figure 3.6 is considered first.

In this Figure g_s is the synaptic conductance and E_s is the equilibrium reverse potential of the synapse. The model can then be described by the following equation.

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_l) - g_s(V - E_s)R_m + I_e R_m \tag{3.4}$$

The synaptic conductance is given by the formula below:

$$g_s = g_{smax} P_{rel} P_s \tag{3.5}$$

With P_{rel} being the probability of a transmitter release given an input spike and P_s being the probability of the ion channels opening after the neurotransmitter release. In reality

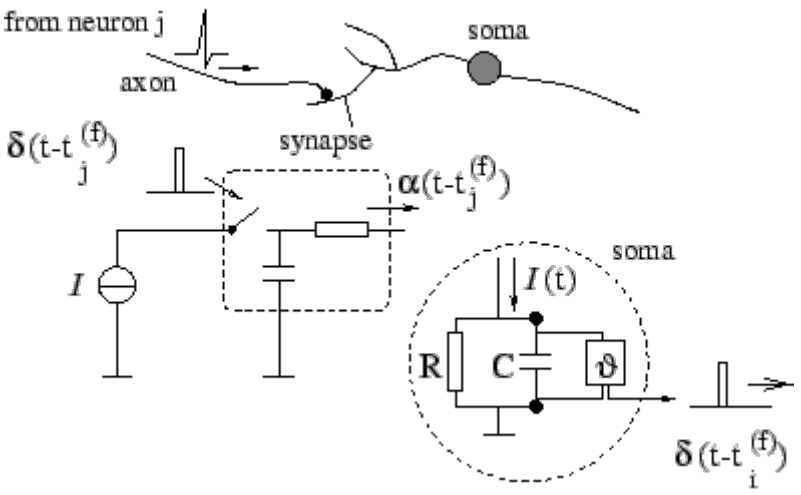


Figure 3.5: Figure showing sources of current into an IAF neuron. The first source which is the soma (cell body) in a biological neuron represents current coming from the environment. The second source, the synapses, represents current coming from spikes of other neurons in this case neuron j. (Taken from [41])

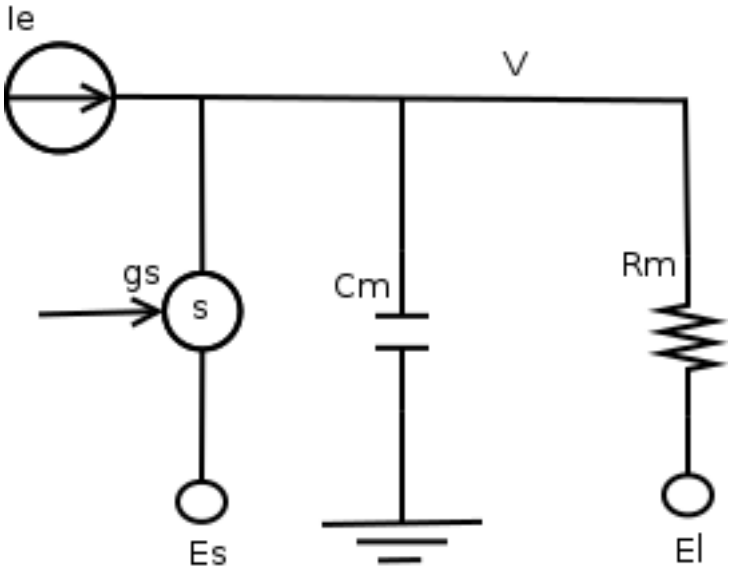


Figure 3.6: Equivalent circuit of cell membrane of an IAF neuron with a synapse. The cell membrane has a voltage V, capacitance C_m and resistance R_m associated with it as well as a leakage potential E_l. The current I_e here is taken as coming from an external source while the synaptic contribution to current is given by the variable synaptic conductance g_s and its potential E_s.

a synapse will not deal with one isolated input spike but with an input spike train and its behaviour will be modified by the input train.

Considering an input spike train as below.

$$p_b(t) = \sum_i \delta(t - t_i)$$

The conductance of the synapse under the influence of this spike train can be modelled according to a particular function $\alpha(t)$ to give:

$$g_s(t) = g_{max} \sum_{t_i<t} \alpha(t - t_i) \tag{3.6}$$

$$g_s(t) = g_{max} \int_{-\infty}^{t} \alpha(t - \tau) p_b(\tau) \mathrm{d}\tau \tag{3.7}$$

These two equivalent equations give the linear filter model for a synapse.

E_s is the reversal potential of the synapse which is much larger than E_l for excitatory synapses and just below E_l for inhibitory synapses. The input current therefore changes according to how high the membrane potential is, the higher the voltage the lower the excitatory effect and the closer the potential is to the resting potential the lower the inhibitory effect.

A further simplification is added in our cases by making the factor $V - E_s$ constant this, as our neurons are rarely at the rest potential. This simplifies equation 3.4 to:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_l) - g_s w R_m + I_e R_m \tag{3.8}$$

and further to:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_l) + R_m(w g_s + I_e) \tag{3.9}$$

Where w is a constant that is equal to $(V - E_s)$. It can be expanded to include all the constants of g_s present in equation 3.6 to give:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - E_l) + R_m\big(w \sum_k \alpha(t - t_k) + I_e\big) \tag{3.10}$$

The current $I(t)$ that flows through R_m for the case of a single synapse on the neuron can be defined as:

$$I(t) = w \sum_k \alpha(t - t_k) + I_e(t) \tag{3.11}$$

Expanding this to include multiple synapse becomes:

$$I(t) = \sum_j w_j \sum_k \alpha(t - t_{jk}) + I_e(t) \tag{3.12}$$

When a presynaptic spike occurs it generates a postsynaptic pulse which decays according to the function $\alpha(t - t_{jk})$. Therefore K is an index of all the spikes that have been produced by a certain neuron j, while J is an index that runs through all the neurons that have

synapses with the target neuron i. The synaptic efficacy is represented by w_{ij} which is a measure of how much and in what way the presynaptic spikes from a particular neuron affect the postsynaptic neurons. For our purposes we define the function $\alpha(t - t_{jk})$ as below.

$$\alpha(t - t_{jk}) = \frac{q}{\tau_s} exp(-\frac{t - t_{jk}}{\tau_s})\mathcal{H}(t - t_{jk}) \tag{3.13}$$

In which $\mathcal{H}(.)$ is the Heaveside step function.

Firing rate models

Using the models presented previously, networks of connected neurons can now be modelled. Beginning with a single post-synaptic neuron connected to multiple pre-synaptic neurons through multiple synapses as shown in the Figure 3.7.

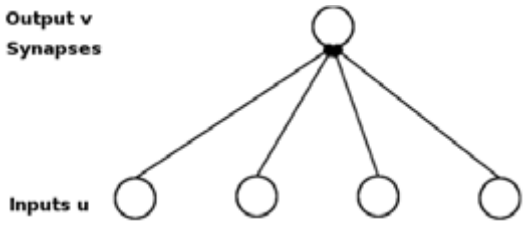


Figure 3.7: Figure showing one output neuron connected to multiple inputs. The output neuron v has connections to all of the inputs $\mathbf{u}$ via the synapses.

The total synaptic current can be given by the equation below:

$$I_s(t) = \sum_{u=1}^{N} I_u(t)$$

Where N is the number of pre-synaptic neurons and I_u refers to the synaptic current from the u-th presynaptic neuron. Using the linear filter model the equation for the current can be rewritten as:

$$I_s(t) = \sum_{b=1}^{N} w_b \int_{-\infty}^{t} \alpha(t - \tau) p_b(\tau) \mathrm{d}\tau \tag{3.14}$$

We can then replace the spike train model of the pre-synaptic neurons with that of the firing rate of the pre-synaptic neurons if the neurons are not correlated or in synchrony. Leading to a new equation below.

$$I_s(t) = \sum_{b=1}^{N} w_b \int_{-\infty}^{t} \alpha(t - \tau) v_b(\tau) \mathrm{d}\tau \tag{3.15}$$

Supposing that α is an exponential where $\alpha(t) = \frac{1}{\tau_s} e^{-\frac{1}{\tau_s}}$. Equation 3.15 can be differentiated to give:

$$\tau_s \frac{\mathrm{d}I_s}{\mathrm{d}t} = -I_s + \sum_b w_b u_b \tag{3.16}$$

These equations produce a new model commonly known as the firing-rate model for a neuron. This models give analog neurons as compared with the spiking neurons previously dealt with. In this model the output is a function of the synaptic current. If the same dynamics for the output firing rate as the input current are assumed the following equation holds:

$$\tau_r \frac{\mathrm{d}v}{\mathrm{d}t} = -v + F(I_s(t)) \tag{3.17}$$

If the time constant τ_r for the output firing rate is much larger than that of the input current $>> \tau_s$ it can be shown that $I_s = \mathbf{w}.\mathbf{u}$ reducing Equation 3.17:

$$\tau_r \frac{\mathrm{d}v}{\mathrm{d}t} = -v + F(\mathbf{w}.\mathbf{u}) \tag{3.18}$$

For most static inputs where $\frac{dv}{dt}$ goes to zero quickly this gives:

$$v_{ss} = F(\mathbf{w}.\mathbf{u}) \tag{3.19}$$

Equation 3.19 has been the governing equation used in artificial neural networks. This represents a significant level of abstraction as the voltage on the firing-rate models of neurons can be solved by summing up its weighted inputs and placing them in a non-linear function while in the Hodgkin Huxley model it required a solution of an ordinary differential equation. This abstraction made it possible to create and simulate neural networks as explained in the following chapters.

3.1.2 Artificial Neural Networks

Building on the work discussed in the previous section, networks with multiple synapses and multiple neurons can be created.

Feedforward Neural Networks

The first network is the feedforward network where information is transferred from one layer to another therefore there are no synaptic connections between neurons in the same

layer creating a network as shown below.

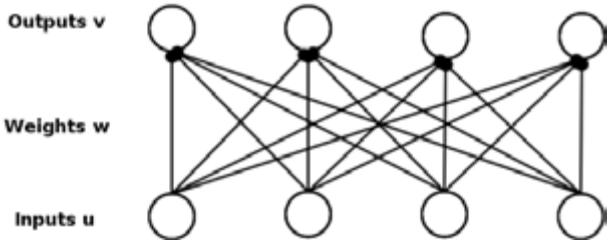


Figure 3.8: Figure showing a Feedforward Artificial Neural Network with two layers. The outputs $\mathbf{u}$ can be generated from the inputs $\mathbf{u}$ and the weights $\mathbf{u}$ using the formula governing artificial neural networks $\mathbf{v} = F(\mathbf{w}.\mathbf{u})$.

Recurrent Neural Networks

The model described in the model above is however not completely accurate, as in real networks in the human brain the outputs are fed back into the network. This leads to an element of feedback which can be modelled by Figure 3.9.

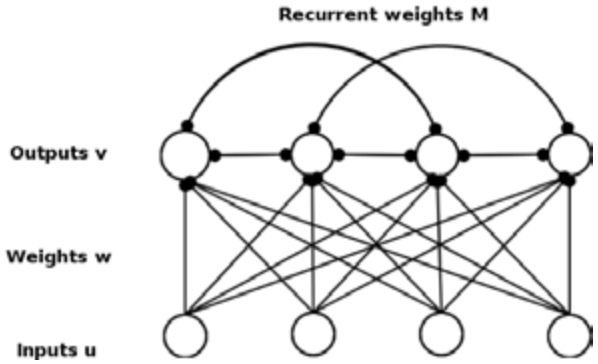


Figure 3.9: Figure showing a Recurrent Artificial Neural Network with two layers. The connections between the outputs $\mathbf{v}$ and the inputs $\mathbf{u}$ is given by the weights $\mathbf{w}$. The internal recurrent connections between the outputs themselves are given by $\mathbf{M}$.

Using this Figure and equation 3.17 a new equation for the firing rates of the outputs can be described as:

$$\tau_r \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbf{v} + F(\mathbf{w}.\mathbf{u} + \mathbf{M}.\mathbf{v}) \tag{3.20}$$

If the function F is linear, a linear Recurrent Network can be described by the following equation:

$$\tau_r \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbf{v} + \mathbf{w}.\mathbf{u} + \mathbf{M}.\mathbf{v} \tag{3.21}$$

If the matrix $\mathbf{M}(N \times N)$ is symmetric then it has N orthogonal eigenvectors e_i and eigenvalues λ_i as shown below.

$$\mathbf{M}e_i = \lambda_i e_i$$

The vector $v(t)$ can be written in terms of eigenvectors of $\mathbf{M}$ as:

$$v(t) = \sum_{i=1}^{N} c_i(t) e_i$$

With this $c_i(t)$ can be solved for by substituting it into equation 3.21. Producing:

$$c_i(t) = \frac{h.e_i}{1-\lambda_i}\left(1 - exp(\frac{-t(1-\lambda_i)}{\tau}) + c_i(0)exp(\frac{-t(1-\lambda_i)}{\tau})\right) \tag{3.22}$$

From this equation we can see that if any of the eigenvalues in $e_i > 1$, $v(t)$ diverges and the network is unstable.

Recurrent neural networks form the basis of both of the networks investigated in this work and lie at the background of reservoir computing which is discussed in the following section.

3.2 Reservoir Computing

Reservoir computing is a fairly new concept that refers to both the Echo State network and the Liquid State machine. They both share the same architecture and only vary in their implementation. Inputs are projected into a high-dimensional space, the reservoir, which consists of recurrently connected neurons and these inputs generate the reservoir states. A linear readout of these states can then be trained to produce a desired output. For the Echo State Network the neurons consist of artificial or firing-rate neurons while for the Liquid State Machine they consist of spiking neurons.

There are broadly speaking three ways to look at the behaviour of the reservoir and each of the sections below looks at them in turn.

3.2.1 The Reservoir as a Kernel

The first aspect we must consider is how the inputs get projected into the reservoir. The reservoir space is usually created as a high-dimensional space and therefore the projection of inputs is from a low-dimensional space to a high-dimensional space. The reservoir thus functions as a kernel in a similar manner to Support Vector Machines and projects the inputs through a nonlinear map to a high-dimension space, this boosts the computational

capabilities of this system.

The kernel functionality can be simply explained by considering two data points that lie on the same plane in low dimensional space and are thus not linearly separable. By projecting these inputs to a higher-dimensional space a hyperplane can be generated which separates the data. This is because if data are represented in a space with more dimensions the probability of the data being linearly separable increases [42].

3.2.2 The Reservoir as a Dynamical system

The second aspect to be considered is the reservoir itself and how its behaves due to its recurrent connections. For this it behaves like a nonlinear dynamic system and aspects from control theory can be used to explain its performance. The first point is that the reservoir must be dynamic enough to different input signals to ensure that the states are linearly separable and hence a linear readout can be constructed for a classification task. However it must not be too dynamic such that small changes in the inputs arising due to noise are amplified and the information about the input is lost in the wild and chaotic dynamics. Therefore to achieve optimal computation the reservoir must lie at the *edge of chaos* which is the edge between the dynamical regime and the chaotic regime [43].

3.2.3 The Reservoir as a Linear Filter

The third aspect has to do with how outputs are generated by reservoir computing methods and this is done by a linear readout of the reservoir states. Therefore the reservoir can be considered as a preprocessing filter that generate states in a similar manner to most signal processing techniques and the theory from this field can be applied to reservoir computing. This theory includes learning methods such as linear regression/ least mean squares which have found repeated and successful use in the signal processing domain.

Building up on these aspects we can then dig deeper into each of these networks in the following sections.

3.3 Echo State Network

An Echo State Network is a modified version of a recurrent artificial neural network that has been modified to ensure that the learning stage is simple. It uses the principle of a reservoir, which in this case is a recurrently connected network of firing-rate neurons. Projecting an input to this reservoir is effectively projecting it into a high dimensional space and therefore the reservoir can be thought of as a non-linear kernel. The desired outputs can then be read off from this reservoir.

Echo State Network Theory

Due to the kernel property of this reservoir, all the information and past history of the input will be stored in the reservoir. Therefore this makes it possible to only train the outputs to match the data needed. Therefore the weights in the reservoir as well as those from the input to the reservoir need not change when we the network is trained. Only the weights from the reservoir to the output are changed in the training phase.

The equations that govern the operation of the ESN have been described in section 2.2.1.

3.4 Liquid State Machine

A liquid state machine (LSM) is a similar architecture to the Echo State Network except that instead of using artificial neurons in its reservoir as the Echo State Network does it uses biologically plausible neurons. This is in an attempt to mimic the behaviour of neural microcircuits in the brain of humans and other mammals. In an LSM a stream of continuous time inputs are processed by a recurrent network of integrate and fire neurons and due to the high-dimensionality of the recurrent network. An output can be readout by transforming these internal states.

Liquid State Machine Theory

As was stated previously the architecture of a LSM closely resembles that of the ESN. A full description of its operation can be found in section 2.2.2.

The output of a LSM can be expressed mathematically as follows:

$$y(t) = f^M(x^M(t)) \tag{3.23}$$

Where $x^M(t)$ represents the reservoir states of the LSM and f^M is a memoryless readout map. Since the readout maps are by definition memoryless then all information needed to produce a target output is within the reservoir states $x^M(t)$. This same condition has been shown to hold for the reservoir states of the Echo State Networks. It therefore follows that only the readout map needs to be trained to produce a specific output. Multiple readout maps can also be created to generate different target outputs from the same liquid states.

Neural Application using IAF neurons

To use IAF neurons for an implementation of a liquid filter some preconditioning steps have to be taken to ensure that this network still retains the properties of an LSM [27]. The first step is that a continuous time input is injected directly into a small percentage of the liquid neurons which have been chosen randomly. The amplitudes of these inputs are also chosen from a Gaussian distribution ($\mu = 0, \sigma = 1$) so that each neuron receives a slightly different input.

The liquid state at time t of an LSM implementation using IAF neurons was defined by Maass et al [27] as *"all the information that a readout neuron can extract at time t from the circuit"*, which is the output of all the neurons at time t. Mathematically this translates to the output of linear filters with exponential decay applied to the outputs of the LSM's neurons.

The readout map could be defined in two different ways. Firstly by using a population P of IAF neurons receiving only input from all the liquid neurons. The firing activity of this population P i.e. the fraction of neurons firing during a time bin of a certain time period, can be interpreted as the output of the LSM. Secondly it could be implemented by a single neuron. In cases where the target output was binary valued, a single IAF neuron could be trained for the classification task with the output being the spikes of the neuron. The firing rate of the single IAF neuron can also be used to produce a slowly varying continuous value if required.

3.5 Learning

As mentioned before learning or adaptation is one of the major features of the nervous systems in animals. It is this feature that enables modification as well as development of behaviours that allow organisms to survive in new and dynamic environments. As was shown in section 3.1, learning is a function of synaptic plasticity which occurs when a synapse changes its strength to enhance or reduce a certain type of behaviour. This change of weight is inspired mostly by Hebb's rule which can be described as below [44].

$$\Delta w_{ij} = \eta y_i x_j \tag{3.24}$$

Where Δw_{ij} is the change in the connection strength of w_{ij}, η is the learning rate and y_i, x_j are the activation values of postsynaptic and presynaptic units respectively. In the case of spiking neurons the Hebb rule becomes a function of the time difference between the reception of a spike from a presynaptic neuron and the emission of a spike from the postsynaptic neuron. Therefore it is commonly termed as Spike Time Dependent Plasticity (STDP) and can be shown by the following equation

$$s\Delta w_{ij} = \begin{cases} A_+ e^{s/\tau_1} & s < 0 \\ A_- e^{-s/\tau_2} & s > 0 \end{cases} \tag{3.25}$$

Where $s = t^{pre} - t^{post}$ is the time difference between arrival of the presynaptic spike and emission of the postsynaptic spike and A_+, A_-, τ_1, τ_2 are neurophysiological constants. Although spiking neurons are used in the LSM, we neglect the use of STDP algorithms and concentrate on learning algorithms that were created for artificial neural networks for this report.

Broadly speaking, learning algorithms for artificial neural networks can be put into two different categories. Unsupervised learning algorithms and supervised learning algorithms. In unsupervised algorithms the network extracts statistically important information from the distribution of input patterns or memorises and reconstructs these patterns. In the case of supervised learning algorithms the desired output is used to guide the process of training the network. The problem in this work could only be solved by a supervised learning algorithm and two different approaches to do so are discussed in the next sections. These different approaches are comprehensively detailed by Floreano and Matussi [37] with work here presented for completion.

3.5.1 Least Squares Regression

In our first approach to supervised learning, the case where there is a certain dataset available that the ESN or LSM should replicate can be considered. The approximation is through a linear combination of the states of the LSM or ESN as seen in the following equation:

$$\tilde{y}(t) = X(t)\mathbf{W} \tag{3.26}$$

Where $X(t)$ is a vector of the network states at time t and $\mathbf{W}$ consists of the weights needed to approximate a certain function. Assuming the function $y(t)$ was to be approximated a good approximation would result in a minimisation of the squared error.

$$R = \sum_{t=1}^{n} (\tilde{y}(t) - y(t))^2 \tag{3.27}$$

This value R is often termed the residual and is the term to be reduced. Collecting all the terms of y in the dataset, the vector $\mathbf{y}$ can be created:

$$\mathbf{y} = (y_1, y_2, y_3, \ldots, y_n) \tag{3.28}$$

The states of the network can be collected into the following matrix:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix} \tag{3.29}$$

Equation 3.27 can be rewritten as below:

$$\mathbf{R} = (\mathbf{y} - \tilde{\mathbf{y}})^2 \tag{3.30}$$

$$\mathbf{R} = (\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}}) \tag{3.31}$$

$$\mathbf{R} = (\mathbf{y} - \mathbf{XW})^T(\mathbf{y} - \mathbf{XW}) \tag{3.32}$$

The optimum weight will be found where the partial derivative of the weights is zero. Partial differentials are taken on both sides to give:

$$\frac{\partial}{\partial \mathbf{W}}\mathbf{R} = \frac{\partial}{\partial \mathbf{W}}[(\mathbf{y} - \mathbf{XW})^T(\mathbf{y} - \mathbf{XW}) \tag{3.33}$$

$$0 = \frac{\partial}{\partial \mathbf{W}}[\mathbf{y}^T\mathbf{y} - \mathbf{XW}\mathbf{y}^T - \mathbf{X}^T\mathbf{W}\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{W}^2] \tag{3.34}$$

$$0 = \mathbf{X}\mathbf{y}^T - \mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{W} \tag{3.35}$$

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.36}$$

Equations adapted from [45].

Equation 3.36 gives the optimum weights to reduce the squared error between $y(t)$ and $\tilde{y}(t)$. Sometimes it is also of interest to reduce the derivative error to ensure that the $\tilde{y}(t)$ changes at the same rate as $y(t)$ did. To do this the residual can be modified as below:

$$\mathbf{R} = (\mathbf{y} - \mathbf{X}\mathbf{W})^T(\mathbf{y} - \mathbf{X}\mathbf{W}) + (\dot{\mathbf{y}} - \dot{\mathbf{X}}\mathbf{W})^T(\dot{\mathbf{y}} - \dot{\mathbf{X}}\mathbf{W}) \tag{3.37}$$

Giving a new value for $\mathbf{W}$ as:

$$\mathbf{W} = (\dot{\mathbf{X}}^T\dot{\mathbf{X}} + \mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{y} + \dot{\mathbf{X}}^T\dot{\mathbf{y}}) \tag{3.38}$$

The residual can be modified further to ensure that the norm of the weights stay close to one as below:

$$\mathbf{R} = (\mathbf{y} - \mathbf{X}\mathbf{W})^T(\mathbf{y} - \mathbf{X}\mathbf{W}) + (\dot{\mathbf{y}} - \dot{\mathbf{X}}\mathbf{W})^T(\dot{\mathbf{y}} - \dot{\mathbf{X}}\mathbf{W}) + \lambda\mathbf{W}^2 \tag{3.39}$$

This gives a new value of $\mathbf{W}$ as:

$$\mathbf{W} = (\dot{\mathbf{X}}^T\dot{\mathbf{X}} + \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{y} + \dot{\mathbf{X}}^T\dot{\mathbf{y}}) \tag{3.40}$$

The weights produced by this function will minimise the squared error between the original function $y(t)$ and its approximation $\tilde{y}(t)$, the derivative error between them as well as ensuring that the norm of the weights is close to one. This third feature is necessary to ensure the robustness of the approximation function when it is being used after training to small changes in the inputs as well as ensuring similar performance when new inputs not in the dataset are introduced.

3.5.2 Least mean squares

In the case of least squares regression the whole dataset is presented before beginning the learning process. However this is not always possible and it reduces the adaptability of the learning process as all possible samples must be present in the dataset. Learning algorithms that update the weights when presented with a new weight offer a possible solution to this problem. To do this some theory from adaptive filters is used and introduce the least mean squares algorithm. This approach was proposed by Widrow

and Hoff [46] and has been used widely in the signal processing field as well as in some neural applications.

Let us consider the following real valued signals as proposed by Widrow and Hoff [46]. The states of the LSM and ESN can be put together into a vector as below:

$$\mathbf{x} = \begin{bmatrix} x_0 & x_1 & \cdots & x_n \end{bmatrix} \tag{3.41}$$

A weight vector that performs a linear combination of these states can be defined as:

$$\mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_n \end{bmatrix} \tag{3.42}$$

These two can be multiplied to produce the output:

$$\tilde{y} = \mathbf{w^T x} \tag{3.43}$$

The error produced by this combination of states as compared to the actual output can be defined as:

$$e = y - \tilde{y} \tag{3.44}$$

A performance function can be defined as:

$$\xi = E[e^2] \tag{3.45}$$

and the autocorrelation of the input as:

$$\mathbf{R} = E[\mathbf{x}\mathbf{x}^T] \tag{3.46}$$

as well as the cross-correlation between the input and the actual output:

$$\mathbf{p} = E[\mathbf{x}\mathbf{y}] \tag{3.47}$$

The performance function can be rewritten as:

$$\xi = E[y^2] - 2\mathbf{w}^T\mathbf{p} + \mathbf{w}^T\mathbf{R}\mathbf{w} \tag{3.48}$$

The performance function ξ is a quadratic function of the weight vector and it has a global minimum that is obtained by finding the optimal weights using the formula:

$$\mathbf{R}\mathbf{w}_{opt} = \mathbf{p} \tag{3.49}$$

This can be expanded and it returns the same equation as equation 3.36. However in this

case $\mathbf{R}$ and $\mathbf{p}$ are not always available. Therefore an iterative search is used to find the optimal values for the weight vector using a gradient descent algorithm. The gradient of ξ is given below as:

$$\nabla\xi = 2(\mathbf{Rw} - \mathbf{p}) \tag{3.50}$$

The iterative search is begun by using an initial guess of the weights namely $\mathbf{w}(0)$ with the weights given after k iterations as $\mathbf{w}(k)$ as below.

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta\nabla_k\xi \tag{3.51}$$

The value η governs how much the weights change and is known as the learning rate. Expanding this equation gives

$$\mathbf{w}(k+1) = \mathbf{w}(k) - 2\eta(\mathbf{Rw} - \mathbf{p}) \tag{3.52}$$

For adaptive filters the real values of $\nabla\xi$ do not usually exist and have to be estimated. A practical scheme for estimating it is referred to as the Least-Mean-Square algorithm.

This acts to minimise the error in the root mean sense by providing a stochastic implementation of the gradient descent algorithm by replacing the performance function $\xi = E[e^2]$ by its instantaneous estimate $\tilde{\xi} = e^2$. This gives:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta\nabla e^2 \tag{3.53}$$

and

$$\nabla = \left[\frac{\partial}{\partial w_0} \quad \frac{\partial}{\partial w_1} \quad \cdots \quad \frac{\partial}{\partial w_N}\right] \tag{3.54}$$

taking the i-th element of the gradient vector ∇e^2 gives:

$$\frac{\partial}{\partial w_i}e^2 = 2e\frac{\partial}{\partial w_i}e \tag{3.55}$$

Substituting for e from 3.44 gives:

$$\frac{\partial}{\partial w_i}e^2 = 2e\frac{\partial}{\partial w_i}(y - \tilde{y}) \tag{3.56}$$

Substituting for $\tilde{y}$ gives:

$$\frac{\partial}{\partial w_i}e^2 = 2e\frac{\partial}{\partial w_i}(y - \mathbf{w}^T\mathbf{x}) \tag{3.57}$$

Taking the partial derivative of this then gives:

$$\frac{\partial}{\partial w_i}e^2 = -2ex(i) \tag{3.58}$$

and the gradient vector becomes:

$$\nabla\xi = -2e\mathbf{x} \tag{3.59}$$

The final equation is then :

$$\mathbf{w}(k+1) = \mathbf{w}(k) - 2\eta e\mathbf{x} \tag{3.60}$$

Modifications

This least mean squares equation in a similar manner to the equation 3.38 does not cater for generalisation as well which would lead to poor performance when small changes occur in the input. To do that an extra factor is added that intends to normalise the weights in a similar way to that in equation 3.40. This extends the LMS equation to below

$$\mathbf{w}(k+1) = \mathbf{w}(k) - 2\eta e\mathbf{x} - \lambda(\mathbf{w}^2 - 1)\mathbf{w} \tag{3.61}$$

3.5.3 Performance Evaluation

Once the weights have been retrieved through any of the processes described above it is important to evaluate how well their performance is in predicting the correct output. There has been a lot of theory developed in the field of machine learning concerning this.

One of these concepts from machine learning involves the generation of two sets of data a test set and a training set. The training set contains the set of samples from which the weights are generated, training is done until the error on the training set is acceptable. The test set however does not feature in the learning process and therefore can give an objective measure of how well the system will perform as the weights are not directly tailored to reduce error on the test set.

The test set error was defined as:

$$Error = \sum_{k=1}^{T_d} \sqrt{\frac{(\hat{y}_d(k) - y_d(k))^2}{\sigma_{d,y}^2}} \tag{3.62}$$

where $\sigma_{d,y}$ denotes the variance of the desired output signal y in example d.

The introduction of these two sets and consequently their errors leads to some interesting results. The first being that very good performance on the training set can lead to very

poor performance on the test set, a phenomena known as overfitting. This means that the network is learning the noise on the training set data rather than learning the statistical properties of the data or generalising and will have a very poor performance on unseen data, in this case the test set. There must be a tradeoff between the error on the training set error and the test set error.

As was stated before overfitting results from a lack of generalisation during the training stage. To ensure a good tradeoff between the training set error and the test set error we introduce a concept known as regularisation. This is a concept in which some constraints are imposed on the model during training to control the tradeoff. In the case of reservoir computing where we have linear models the penalty term introduced is usually related to the norm of the weights. Therefore in the equation for online learning:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - 2\eta e\mathbf{x} - \lambda(\mathbf{w}^2 - 1)\mathbf{w} \tag{3.63}$$

and that for linear regression

$$\mathbf{W} = (\dot{\mathbf{X}}^T\dot{\mathbf{X}} + \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{y} + \dot{\mathbf{X}}^T\dot{\mathbf{y}}) \tag{3.64}$$

the factor λ controls the regularisation by trading off between the training set error and the weight norm. It must be noted that regularisation can also be obtained by adding noise to the training set data.

Another important concept from machine learning is that of cross validation. Cross validation aims to give a better evaluation of performance by eliminating misleading results that were due to a poor choice of the training and test data sets. This can occur especially in a classification task when the training set does not cover the full space of classes.

Cross validation therefore involves splitting the dataset which would have D samples into a number of K subsets each with the same number of samples. $K-1$ subsets are then selected for training and the remaining subset is used for testing of the model. This is repeated K times with a different subset for testing used every time. The performance is then averaged out across all the test performances.

This can be extended to the choosing of the optimal regularisation parameter λ by performing a nested cross-validation. Here there are three sets; the training set, the validation set which is used for evaluating performance in the inner loop and the test set used to evaluate performance in the outer loop. The nested cross validation with K subsets proceeds as follows, $K-1$ subsets are selected for training and the remaining

subset is used as the test set this forms the outer loop. Every time there is an iteration of the outer loop λ is set to a different value, the performance of this value is then evaluated using the $K-1$ subsets, $K-2$ subsets form the training set for this with the other falling into the evaluation set. The performances are then averaged to give the performance of that λ value.

3.6 Concluding Remarks

This chapter introduced the biological neuron and the attempts made to model it and simplify it. The first model presented was the Hodgkin Huxley model which captured the biophysical parameters of a neuron. This however results in an ordinary differential equation that needs to be solved for the membrane potential of the neuron via a numerical method. As such it does not allow for the simulation of large numbers of neurons that our purposes require.

Two further simplifications of the model were considered the first creating the Integrate and Fire model. This captures the basic dynamics of a neuron and forms the basis of the Liquid State Machine. The second further simplification creates the firing rate model of neurons which forms the basis of the Echo State network. These two networks were then discussed along with some of the learning rules proposed to train them.

Chapter 4

Experimental Background

4.1 Introduction

In this chapter we describe the experiments taken to investigate whether certain biologically based neural architectures can be used in the robotics community for navigation of small mobile robots. We restrict the investigation of these architectures to two different aspects of robotic navigation, the first aspect being semantic localisation.

For the purpose of this report we define semantic localisation as the ability of the robot to determine its current position in a previously viewed environment through labels of the environment itself, i.e. what room it is in in a building. While maintaining the same concept of determining a current location, this semantic localisation is different from that usually found in the SLAM literature, in the latter case localisation refers to the determination of the exact position and orientation (6 Degree of Freedom pose) of a robot in an environment.

The second aspect of navigation considered is that of motion planning i.e can the robot create and successfully implement a trajectory from its current position to a goal position. For the motion planning a teach and repeat framework is utilised. In this framework a demonstration of a possible path, created from some expert information, from a current position to a prescribed target position is performed. The goal of this framework is to attempt to recreate the demonstrated path when the robot returned to the initial position. As in the case of semantic localisation the position we refer to is one that comes from a semantic understanding and not the exact pose, and the output of semantic localisation is fed into the motion planner.

These two separate tasks are first tackled through a simulation experiment before being transferred to a stand-alone embedded robotic platform.

4.2 Simulation Experiment

From simulation we present experiments for the semantic localisation as well as navigation tasks. These are implemented via a small mobile robot in a virtual room environment. The goal of the biological based neural network semantic localisation within this experiment is to learn to accurately determine what room the robot is currently in. This semantic information is then used in the motion planning task, where the biological neural network is used through a teach and repeat framework to plan and execute the motion of the robot from its current location to one determined by an end-user. The details of the implementation of these tasks are described below.

4.2.1 Simulation Environment

The virtual environment for the experiment was created using the software Virtual Robotic Experimental Platform (VREP) version 3.0.2 [47] developed by Coppelia Robotics. VREP is an open-source, free to use full physics simulator that uses an ordinary differential equation engine to perform its calculations and can be used to simulate robotic systems interacting with a realistic environment. For the experiment, an environment consisting of four rooms shown in Figure 4.1 was created.

The simulated robot used in this experiment was based on the e-puck robot [48] shown in Figure 4.2. This particular robot was also used in the experiments conducted by Antonelo et al. [13] and therefore represented a good baseline for comparison of results, a fully functioning model of the e-puck robot is included in the VREP software.

The e-puck robot has two independently driven wheels actuated by stepper motors which can be controlled by the simulator. It is equipped with eight proximity sensors of range [0-100] cm placed around its body and two line sensors placed at the bottom of the robot.

Although Vrep was used to create the experimental environment and run the simulations, code development was done in Matlab V2008_a. Code was therefore developed to work in and around the simulation environment using the inbuilt VREP Application Program

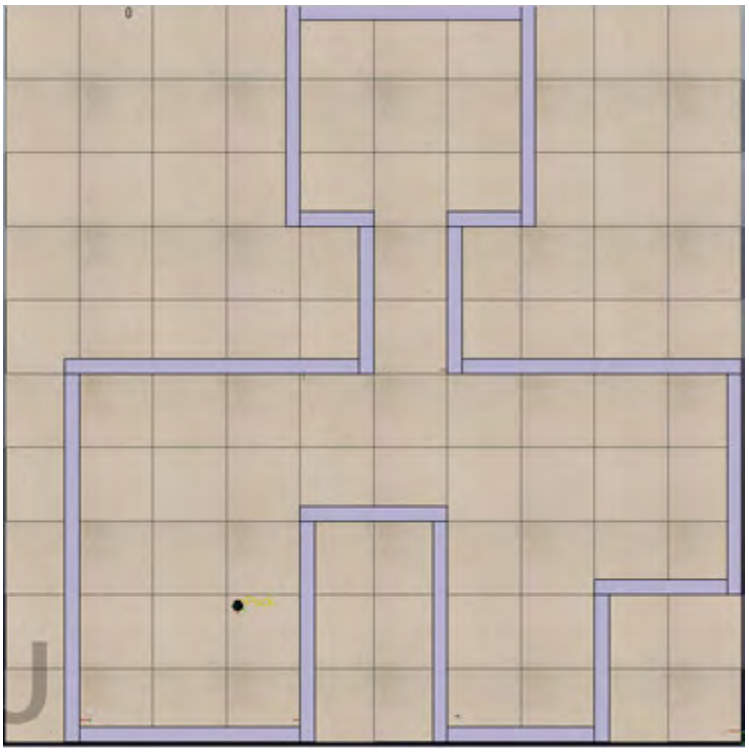


Figure 4.1: Virtual room environment as created in VREP viewed from above. The blue rectangles represent the walls of the rooms with the black dot represents the robot.

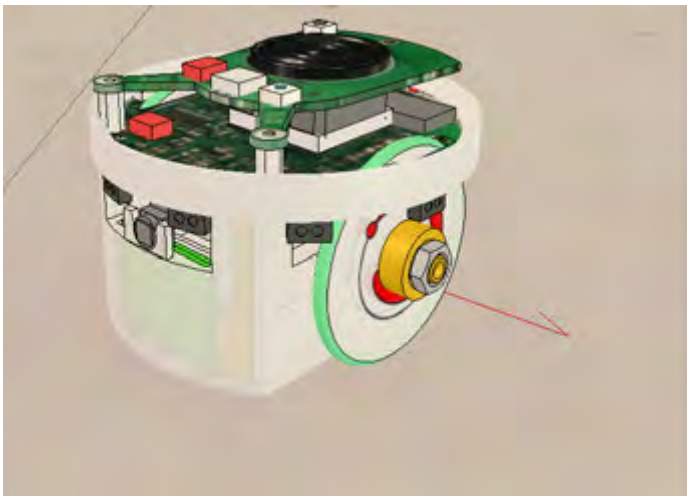


Figure 4.2: Figure showing the model of the e-puck robot created in Vrep. The robot consists of two wheels actuated by stepper motors, 8 infrared proximity sensors placed along its circumference and 2 line sensors placed underneath it.

Interface (API) that allows for communication with other programming languages. The room environment from VREP captured in Matlab can be seen in Figure 4.3, the dimensions of the simulation experiment (5 by 5) metres as well as the boundaries of the rooms in the environment are also highlighted in this Figure.

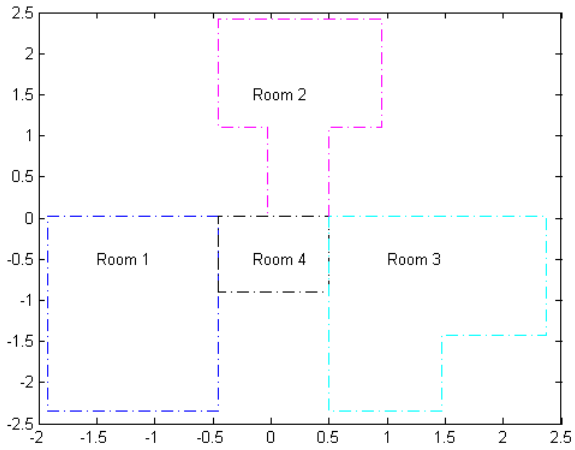


Figure 4.3: Figure showing the virtual environment captured in Matlab with the boundaries of each of the four rooms outlined.

4.2.2 Simulation Task

With the environment and robotic platform defined, we designed an experiment that encapsulates both tasks of this report. In this experiment the robot placed in any one of the rooms is to navigate from its starting position to a goal location dictated by the end-user. The goal location also coincides with the room where a target cylinder is located. One of the permutations of this experiment, showing the robot in room 1 while the target cylinder is in room 2 is shown in Figure 4.4.

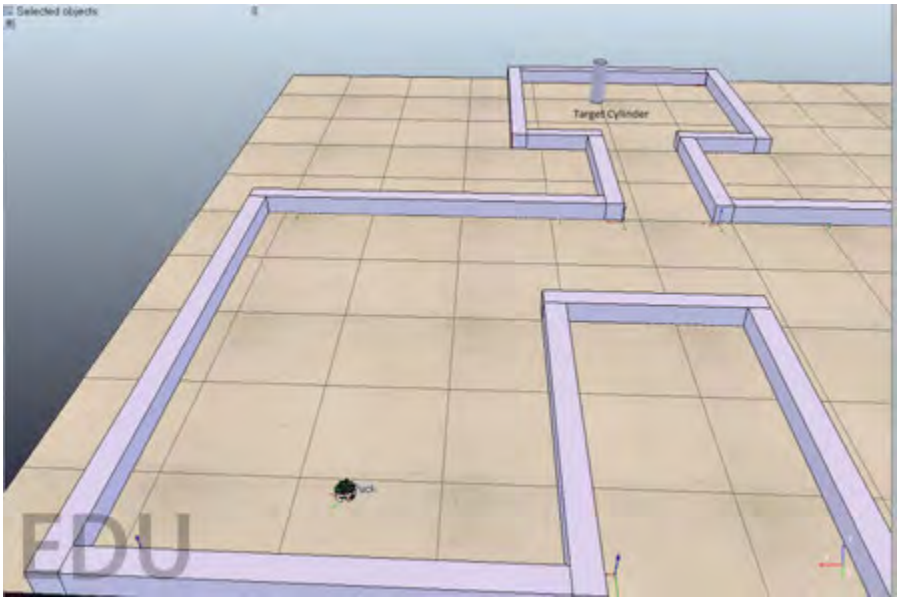


Figure 4.4: Figure showing the virtual room environment with the robot's initial position in room 1 while the target cylinder lies in room 2.

Within this experiment the robot must through the biological neural network learn to perform semantic localisation by determining what room it is in, followed by navigation through a teach and repeat framework implemented in the same biological neural networks learn paths to the target cylinder.

For the teach pass of navigation a path was generated from the initial robot's position to the target cylinder using the A* algorithm [21]. This algorithm calculates an optimum path from the initial position to the target that does not traverse any solid obstacles. Details of the exact algorithm implemented can be found in Appendix A. The algorithm was developed and executed in Matlab and after the path-planning was complete, the path was sent to Vrep. An example of an optimal path generated by this algorithm in both Matlab and Vrep is highlighted in Figure 4.5 and Figure 4.6 respectively.

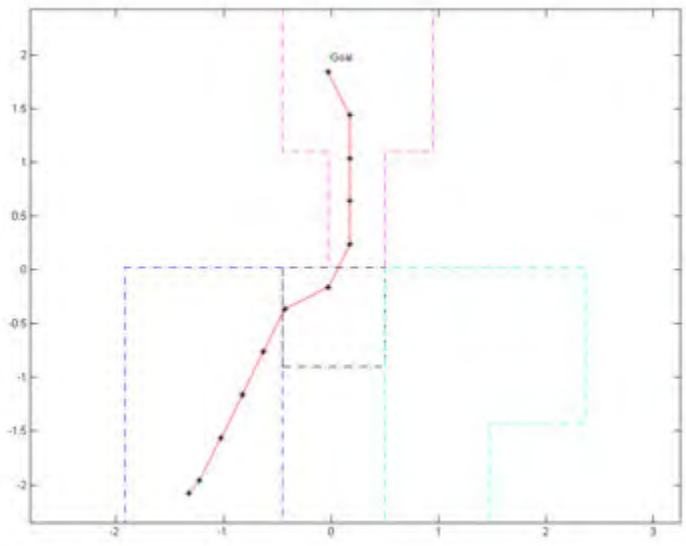


Figure 4.5: Figure showing an optimal path in the virtual environment created in Matlab from a starting position in room 1 up to the target cylinder in room 2.

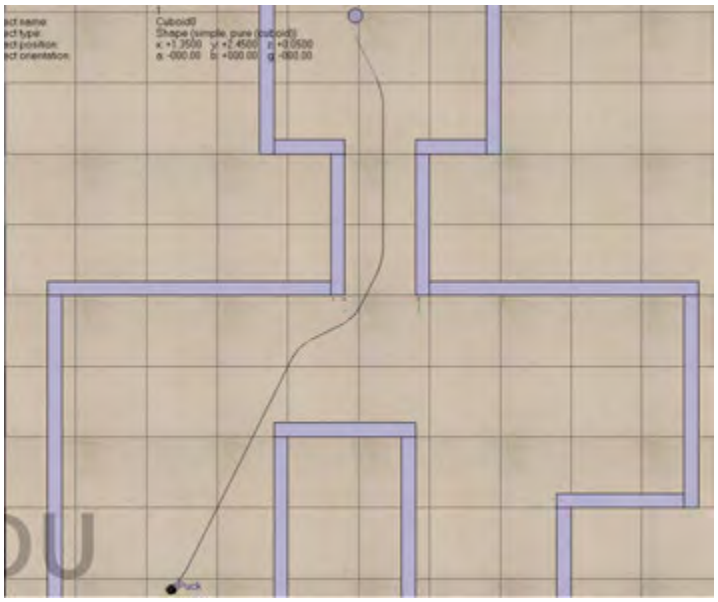


Figure 4.6: Figure showing the executable optimal path in Vrep from a starting position in room 1 up to the target cylinder in room 2.

With the optimal path constructed as a line on the ground of the room environment in Vrep, a simple line-following controller using the two line sensors on the robot was used to traverse the path and complete the navigation to the goal. Pseudo code for this controller is shown in Algorithm 3.

Result: Path Following
while *Target Cylinder not detected* **do**
    RightMotorSpeed=100%;
    LeftMotorSpeed=100%;
    if *Line Detected by LeftLineSensor* **then**
        RightMotorSpeed=25%;
        LeftMotorSpeed=100%;
    end
    if *Line Detected by RightLineSensor* **then**
        RightMotorSpeed=100%;
        LeftMotorSpeed=25%;
    end
end

Algorithm 1: Line Following controller

This simple controller was designed to ensure portability to a physical implementation of the robot with minimum cost and design effort. While the robot is guided along the path by the controller in the teach stage data from the environment is collected, this data includes the current room location, the outputs from the proximity sensors and the

inputs to each of the motors. With this information the neural networks can learn the to perform the semantic localisation and navigation tasks and the full implementation details of first the ESN and then the LSM are discussed below.

4.2.3 Echo State Network

Semantic Localisation

The first task considered is that of semantic localisation where the robot through the ESN determines what room its in. From the input consisting of the outputs of the proximity sensors, the ESN is trained to output both the current and previous room locations as shown in Table 4.1.

Table 4.1: Localization Network

Number of Inputs $u(t)$	8 proximity sensors
Number Of Outputs $y(t)$	8 location predictions
Number of neurons N	400
$\lvert\lambda_{max}\rvert$	0.97

The number of inputs correspond to the number of proximity sensors that were present in the simulated robot, while the location predictions correspond to double the amount of the rooms available. This is predictions are made for both the current and previous room locations. The value N of 400 was used for consistency with the number of states used by Antonelo et al. [13] where a similar experiment into semantic localisation, utilising a similar e-puck robot in a similar room environment, was carried out. From their reported results, the size of the network was suitable for this experiment and to enable cross-comparison of results was chosen in this experiment to be 400.

The states $x(t)$ of the ESN used for the semantic localisation are given by the following equation:

$$\mathbf{x}(t+1) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t) + \mathbf{W}_{biasres}) \tag{4.1}$$

with its output given by the following equation that has been modified from Equation 2.7

$$\mathbf{y}(t+1) = \mathbf{W}_{out}\mathbf{x}(t+1) + \mathbf{W}_{biasout} \tag{4.2}$$

As this is a classification task, the output is modified by taking it through the non-linear winner takes all function $G(\cdot)$. This function gives a value of 1 to the highest input and

-1 to all the others. The predicted current location is retrieved by:

$$\mathbf{y}(t+1)_{Cloc} = G(\begin{bmatrix} y_0 & y_1 & y_2 & y_3 \end{bmatrix}) \tag{4.3}$$

and the predicted previous room location by:

$$\mathbf{y}(t+1)_{Ploc} = G(\begin{bmatrix} y_4 & y_5 & y_6 & y_7 \end{bmatrix}) \tag{4.4}$$

Training for $\mathbf{W}_{out}$ and $\mathbf{W}_{outbias}$ is done by one of the schemes described before, either linear regression or least mean squares.

For linear regression, the actual outputs from the simulation that are used to train the network are recorded and gathered in the matrix $\mathbf{Y}$:

$$\mathbf{Y} = (y_1, y_2, y_3, \ldots, y_m) \tag{4.5}$$

The states of the network are recorded into the matrix below:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,N} \end{pmatrix} \tag{4.6}$$

The values for $\mathbf{W}_{out}$ will then be given by the equation below.

$$\mathbf{W}_{out} = (\dot{\mathbf{X}}^T\dot{\mathbf{X}} + \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{Y} + \dot{\mathbf{X}}^T\dot{\mathbf{Y}}) \tag{4.7}$$

For online learning the weights are updated as the robot is moving to the goal. The weight update equation is given below

$$\mathbf{W}_{out}(k+1) = \mathbf{W}_{out}(k) - 2\eta e\mathbf{x} - \lambda(\mathbf{W}_{out}^2 - 1)\mathbf{W}_{out} \tag{4.8}$$

Where e is given by the equation below.

$$e = \mathbf{y} - \mathbf{W}_{out}^T\mathbf{x} \tag{4.9}$$

Using one of these schemes, the ESN can be trained to fulfil the localisation task.

Teach and Repeat

With the semantic localisation information obtained from the ESN, the robot can begin the teach and repeat motion task. This is performed using input from the proximity sensors, the semantic localisation information as well as goal information provided by the user. This network then executes the motion by providing outputs to the motors of the simulated robot. There are 3 goal locations as for the purpose of the experiment we consider room 4 as an intermediary room and not a final location. A similarly sized network as in the semantic localisation task is used here.

The various variables used in this network are shown below.

Table 4.2: GoalSeeking Network

Number of Inputs	8 proximity sensors, 8 predicted locations, 3 goal locations
Number of Outputs	2 motor speeds
Number of neurons	400

Final Network and Training

One can quickly notice that all of the inputs needed for teach and repat are included in the goal-seeking task. Therefore it is possible to train the network to also output the location positions that are outputs of the first task. These outputs are in fact inputs into the second task and can thus be fed-back into the network again, this ensures that only one network is needed to perform both the tasks described which translates to a saving in computational power. There is however an increased cost when it comes to the training stage.

The approach used in the final network is called teacher forcing. In this approach the weights of the outputs that are to be fed-back into the system are first trained using either linear regression or online learning. To simulate the inputs that would have been fed-back into the system noisy versions of the ideal inputs are used, this "forces" the network to be robust enough to use some of its outputs as inputs. Once these weights have been trained there are set as constant and the feedback in the system is enabled. With the feedback enabled another dataset is obtained and used to train the other outputs while keeping the other weights constant.

For the case above the training stage proceeds as follows. Only the case of linear regression

is shown here where the states are first generated as below.

$$\mathbf{x}(t+1) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t) + \mathbf{W}_{biasres}) \tag{4.10}$$

The vector $\mathbf{u}$(t) contains the following variables:

$$\mathbf{u}(t) = \begin{bmatrix} sensvect & locvect & goalvect \end{bmatrix} \tag{4.11}$$

Where **sensvect** is a vector that contains the outputs of the eight proximity sensors:

$$\mathbf{sensvect} = \begin{bmatrix} sensor_1 & sensor_2 & \cdots & sensor_8 \end{bmatrix} \tag{4.12}$$

locvect is a vector that contains noisy versions of the current as well as predicted room locations:

$$\mathbf{locvect} = \begin{bmatrix} Croom_1 & Croom_2 & \cdots & Proom_4 \end{bmatrix} \tag{4.13}$$

and **goalvect** is a vector that contains the chosen goal position:

$$\mathbf{goalvect} = \begin{bmatrix} Room_1 & Room_2 & Room_3 \end{bmatrix} \tag{4.14}$$

The states are gathered into the matrix $\mathbf{X}$ and their corresponding outputs into the matrix $\mathbf{Y}$, and the weights are then trained using the following equation:

$$\mathbf{W}_{out1} = (\dot{\mathbf{X}}^T\dot{\mathbf{X}} + \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{Y} + \dot{\mathbf{X}}^T\dot{\mathbf{Y}}) \tag{4.15}$$

The output is then given by:

$$\mathbf{y}_{location}(t+1) = \mathbf{W}_{out1}\mathbf{x}(t+1) + \mathbf{W}_{biasout1} \tag{4.16}$$

After this has been taught the other outputs can be trained, which are the motor speeds. The states of these are given by the equation below:

$$\mathbf{x}(t+1) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t) + \mathbf{W}_{biasres}) \tag{4.17}$$

The vector $\mathbf{u}$(t) contains the following variables:

$$\mathbf{u}(t) = \begin{bmatrix} \mathbf{sensvect} & \mathbf{y}_{location} & \mathbf{goalvect} \end{bmatrix} \tag{4.18}$$

The states are gathered into the matrix $\mathbf{X}$ and their corresponding outputs into the matrix $\mathbf{Y}$, and the weights are then trained using the following equation:

$$\mathbf{W}_{out2} = (\dot{\mathbf{X}}^T\dot{\mathbf{X}} + \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{Y} + \dot{\mathbf{X}}^T\dot{\mathbf{Y}}) \tag{4.19}$$

The output is:

$$\mathbf{y}_{motor}(t+1) = \mathbf{W}_{out2}\mathbf{x}(t+1) + \mathbf{W}_{biasout2} \tag{4.20}$$

The final equations are as follows:

$$\mathbf{u}(t) = \begin{bmatrix} \mathbf{sensvect} & \mathbf{y}_{location} & \mathbf{goalvect} \end{bmatrix} \tag{4.21}$$

$$\mathbf{x}(t+1) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t) + \mathbf{W}_{biasres}) \tag{4.22}$$

$$\mathbf{y}_{location}(t+1) = \mathbf{W}_{out1}\mathbf{x}(t+1) + \mathbf{W}_{biasout1} \tag{4.23}$$

$$\mathbf{y}_{motors}(t+1) = \mathbf{W}_{out2}\mathbf{x}(t+1) + \mathbf{W}_{biasout2} \tag{4.24}$$

This network can then be used for the path-planning purposes.

4.2.4 Liquid State Machine

As was mentioned in the previous sections both the Liquid State Machine and the Echo State Network operate on the same reservoir computing principle therefore the processes required to train the network are very similar. In this section we therefore concentrate on the steps needed to generate the states of the LSM. The states of the LSM are given as shown below.

$$x^M(t) = (L^M u)(t) \tag{4.25}$$

The states are formed as a projection of the continuous spike train $u(t)$. To use the LSM we must first convert our sensor inputs into a spike train.

Neural Encoding

Neural encoding refers to the transfer of information from a sensory input into a spike train. The brain typically encodes information in a multitude of ways with the two most studied being frequency of the spike train (rate), and the other being the inter-spike interval between the spikes [49].

For our purposes, information was encoded by using the sensor input as the input current into an integrate and fire (IAF) neuron. This neuron spikes if the current causes its potential to go above a certain threshold. Only the distance sensors in our experiment have a continuous range of values therefore the neurons corresponding to these sensors will also have time-varying frequencies. For the binary valued inputs, for example the room

location, the corresponding neurons will either be silent encoding a zero or will fire at a constant rate. It must be noted that there is a hard boundary on the highest frequency that an IAF neuron can have which is determined by its absolute refractory period therefore during the selection of the frequencies the highest frequency of the neurons connected to the distance sensor had to be lower than this.

Network Structure

To keep consistency with the ESN, the LSM was defined as having 400 IAF neurons. 20% of these were defined to be inhibitory and the rest being excitatory which corresponds with data found from nature [27]. The neurons were also arranged into a cuboid structure of $10 \times 8 \times 5$ as shown in Figure 4.7 below. The use of this cuboid structure is to replicate

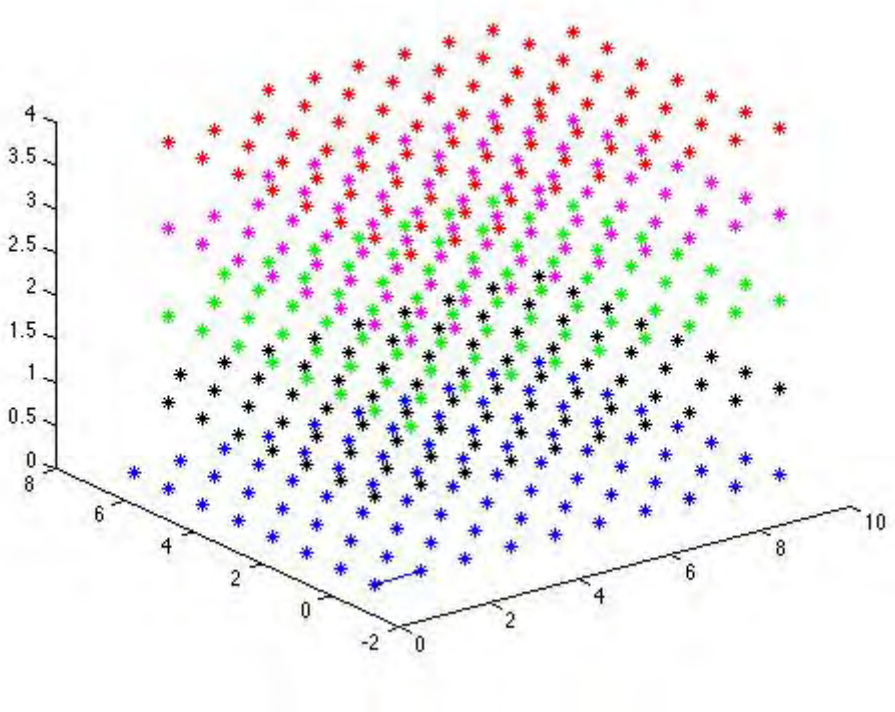


Figure 4.7: Figure showing the arrangement of neurons to form the LSM. The neurons were arranged in a cuboid structure of $10 \times 8 \times 5$. Each colour represents 10×8 neurons placed at a different height.

one of the properties of neural systems observed in nature, whereby we know that there is a strong likelihood that neurons that are physically close together are more likely to be connected together[27]. This structure enables the use of a spatial connectivity pattern where the connectivity of the neurons is a parameter of the distance between two neurons, the probability of two neurons being connected is greater if they are close together than if they are further apart.

For an individual IAF neuron the governing equation is given by

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_l) - I_e R_m \tag{4.26}$$

If $V \rightarrow V_{th}$ a spike is fired.
Then $V \rightarrow V_{reset}$.

With this work investigating the practical suitability of these methods rather than investigating the network itself network parameters from previous work from Maass et al.[27] which have exhibited behaviour closely replicating biological neurons were used in the creation of the LSM as shown in Table 4.3.

Table 4.3: GoalSeeking Network

Parameter	**Excitatory**	**Inhibitory**
Time Constant τ_m	30 ms	30 ms
Absolute Refractory Period	3 ms	2 ms
Threshold V_{th}	15 mV	15 mV
Resting Voltage V_l	0 mV	0 mV
Reset Voltage V_{reset}	13.5 mV	13.5 mV
Background Current I^{ext}	13.5 nA	13.5 nA
Input Resistance R_m	1 MΩ	1 MΩ

We also introduce another element of biological realism to the network used in the LSM. This is by introducing dynamic synapses, these are synapses whose behaviour changes not only according to trained weights as in the case of static synapses but also change according to their use [50]. This effectively translates to the reduction of the capacity of a pre-synaptic neuron to influence a post-synaptic neuron, known as synaptic efficacy, after a spike is fired. Given the equation for current in an IAF neuron as below.

$$I_i(t) = \sum_j w_{ij} \sum_k \alpha(t - t_{jk}) + I_i^{ext}(t) \tag{4.27}$$

A new parameter R which represents the percentage of the synaptic efficacy available after a presynaptic spike is defined as below.

$$R_{n+1} = R_n(1 - u_{n+1})exp(\frac{-\Delta t}{\tau_D}) + 1 - exp(\frac{-\Delta t}{\tau_D}) \tag{4.28}$$

Another parameter U which represents the utilisation of the available synaptic efficacy is also defined as:.

$$u_{n+1} = u_n exp(\frac{-\Delta t}{\tau_F}) + U(1 - exp(\frac{-\Delta t}{\tau_F})) \tag{4.29}$$

As well as a scaling factor A which determines how much a presynaptic synapse affects a

postsynaptic neuron. This gives a definition of the dynamic synaptic weight as:

$$w_{ij} = A.R_n.u_n \tag{4.30}$$

The function α is also defined as:

$$\alpha(t - t_{jk}) = \frac{q}{\tau_s} exp(-\frac{t - t_{jk}}{\tau_s})\mathcal{H}(t - t_{jk}) \tag{4.31}$$

Where τ_s is 3 ms for excitatory neurons and 6 ms for inhibitory neurons. Similarly to Table 4.3 the synaptic parameters were selected from [50] and are dependent on the types of neurons connected. The values of these are chosen from a normal distribution with a mean and standard deviation shown in the table below for all the different types of connections.

Table 4.4: Synaptic Connection Parameters

Synaptic Parameter	EE	EI	IE	II
Use U(mean,SD)	0.5, 0.25	0.05, 0.025	0.25, 0.125	0.32, 0.16
Depression τ_D s(mean,SD)	1.1, 0.55	0.125, 0.0625	0.7, 0.35	0.144, 0.72
Facilitation τ_F s(mean,SD)	0.05, 0.025	1.2, 0.6	0.02, 0.01	0.06, 0.03
Scaling A nA(mean,SD)	30, 30	60, 60	-19,- 19	-19, -19

Using this information the LSM for the localisation task can be created.

Generating the LSM states

With the sensory inputs encoded into spike trains, they can be projected into the column of neurons that forms the LSM. This projection is done via the injection of these spike trains as inputs to the synapses of some of the neurons in the LSM. This projection is however only done to 30 % of the reservoir neurons which are chosen randomly from the column, this number as is shown by Maass et al.[27] guarantees significant synaptic activity and it corresponds with multiple models of the brain in which most of the connections are internal rather than from sensory input. The injection is done via an injection current $I_i^{injection}$ into the reservoir neurons which is then weighted randomly to ensure that two neurons do not receive the same amount of input. The current to the reservoir neurons can be shown below.

$$I_i(t) = \sum_j \mathbf{A}_{ij}\mathbf{R}_{ij}\mathbf{U}_{ij} \sum_k \alpha(t - t_{jk}) + I_i^{background}(t) + I_i^{injection} \tag{4.32}$$

With the governing equations for the neurons as below:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_l) - I_e R_m \tag{4.33}$$

If $V \rightarrow V_{th}$ a spike is fired.
Then $V \rightarrow V_{reset}$.

The LSM states are obtained by taking the reservoir spikes through an exponential filter with a given time constant. This when applied mathematically translates to the equation below.

$$x^M(t+1) = \eta * spikes(t) + (1 - \alpha) * (x^M(t)) \tag{4.34}$$

The equation to obtain η is shown below:

$$\eta = 1 - exp(-dt/\tau_c) \tag{4.35}$$

The value dt represents the smallest time step taken as this system is to be implemented in digital hardware while τ_c is the time constant of the exponential filter.

Final system

Once the states have been generated a linear combination of them can be used to generate a particular output in a similar manner to the ESN. Therefore for the LSM the governing equations are similar to those of the ESN with only one change being the encoding stage.

$$\mathbf{u}(t) = \Big[\mathbf{sensvect} \quad \mathbf{y}_{location} \quad \mathbf{goalvect}\Big] \tag{4.36}$$

$$\hat{u} = (L^M \mathbf{u})(t) \tag{4.37}$$

$$x^M(t) = (L^M \hat{u})(t) \tag{4.38}$$

$$\mathbf{y}_{location}(t) = \mathbf{W}_{out1}\mathbf{x}^M(t) + \mathbf{W}_{biasout1} \tag{4.39}$$

$$\mathbf{y}_{motors}(t) = \mathbf{W}_{out2}\mathbf{x}^M(t) + \mathbf{W}_{biasout2} \tag{4.40}$$

The values of $\mathbf{W}_{out1}$ and $\mathbf{W}_{out2}$ can be obtained by training the network using linear regression or online learning.

4.3 Algorithm Development

This section describes the steps taken to convert the mathematical concepts of the Echo State Networks and the Liquid State Machine described in section 4.2.3 and 4.2.4 into actual algorithms that can be coded and implemented. All of the experiments carried out contained three distinct stages, the first stage is the initialising of the network parameters followed by the training of the optimum weights after which the execution stage follows.

4.3.1 Initialising The Network Parameters

Echo State Network

For the Echo State Network given by the equation below, two parameters need to be defined.

$$\mathbf{x}(t+1) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t)) \tag{4.41}$$

The matrix $\mathbf{W}_{in}$ represents the projection of the inputs to the reservoir. Therefore it is an $n_{states} \times n_{inputs}$ matrix. The next matrix that needs to be created is $\mathbf{W}_{res}$ which represents the recurrent activity in the network, it is an $n_{states} \times n_{states}$ matrix

Liquid State Machine

Compared to the Echo State Network which is very much a mathematical construct, the Liquid State Machine uses biologically plausible spiking neurons and connections and therefore its initialisation and creation on computer hardware is more complex. The flow chart in Figure 4.8 breaks down the theoretical development presented in section 4.2.4 into steps needed to correctly initialise an LSM consisting of N neurons.

4.4 Training the network

Once both the networks have been created and initialised the next step that follows is the training of the network for the specific task, be it semantic localisation or for the teach and repeat framework that we are investigating. For the training stage of both of these tasks as described in Section 4.2.3 the robot is moved, under some form of control,

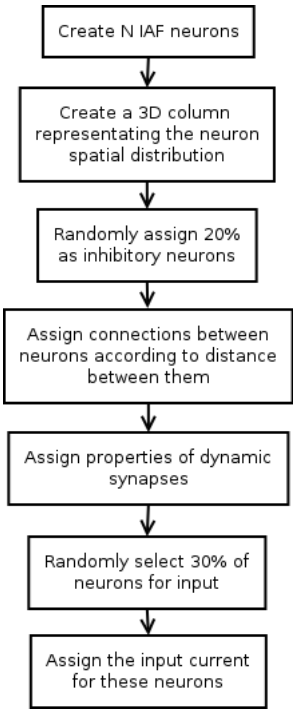


Figure 4.8: Flowchart showing the steps needed to create and initialise a liquid state machine consisting of N neurons as developed in section 4.2.4

through the room environment and it subsequently captures data for either the semantic localisation or teach and repeat. A general flowchart for this training process based on this theoretical development is shown in Figure 4.9.

The difference between an implementation of the ESN and the LSM in Flowchart 4.9 exists in only one step, the generation of states. For the ESN the generation of states is again very mathematical and follows equation 4.42.

$$\mathbf{x}(t+1) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t)) \tag{4.42}$$

The LSM follows a more complicated process as described in Section 4.2.4 and is summarised in the flowchart in Figure 4.10.

4.4.1 Execution stage

Once the weights to the output of either the ESN or the LSM have been trained the network can be used for a particular task, in our case the navigation of a robot to a goal. This can be shown by the flowchart in Figure 4.11.

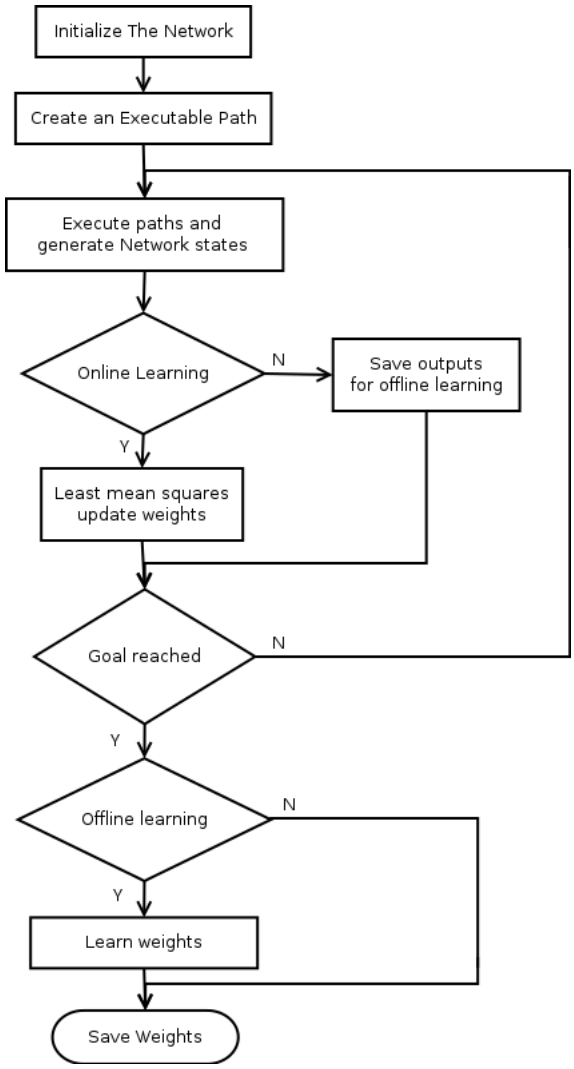


Figure 4.9: Flowchart showing the steps followed to train either a Liquid State Machine or Echo State Network for semantic localisation or teach and repeat. This flowchart follows the theoretical development described in Section 4.2.3.

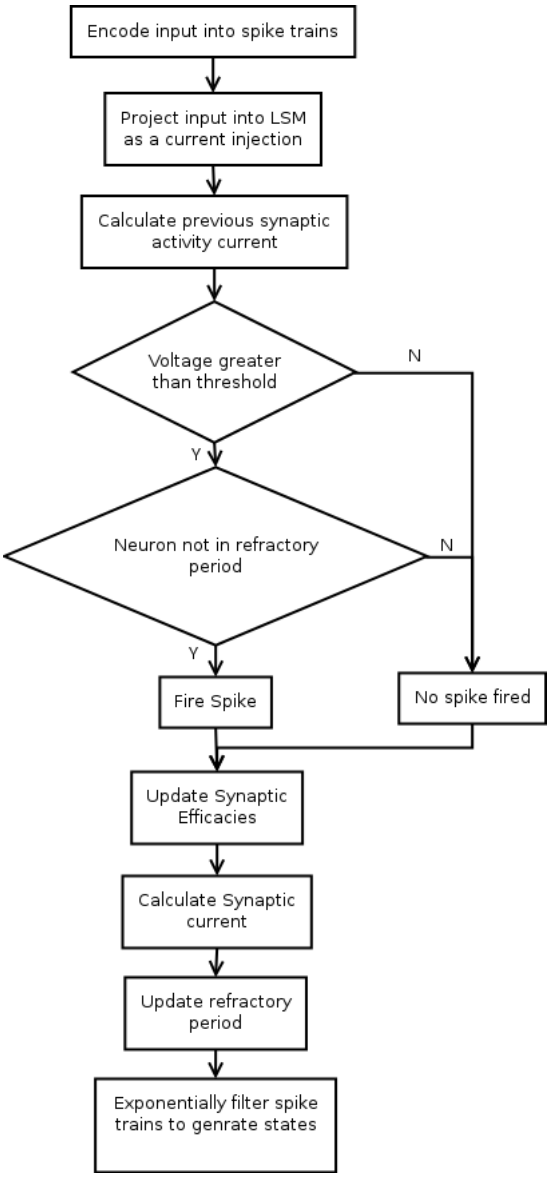


Figure 4.10: Flowchart showing the steps followed to generate the reservoir states of a Liquid state Machine given a particular input. This flowchart summarises the theoretical development provided in Section 4.2.4

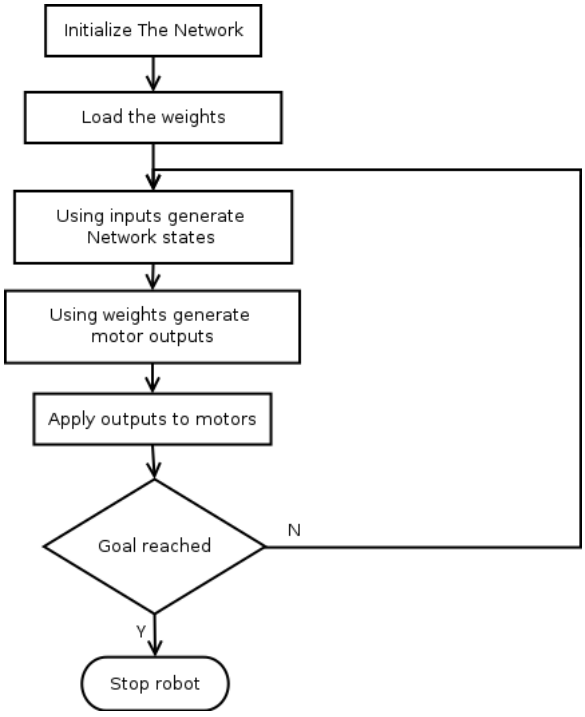


Figure 4.11: Flowchart showing the steps followed to navigate a robot to a goal using an ESN or a LSM as a controller.

4.5 Physical Experiment

In this section the physical experiment that was done to investigate whether the Echo State Network and the Liquid State Machine could be used in the navigation problem in robotics is described.

4.5.1 Physical Environment

To keep consistency with all the experiments, the physical environment was built to be as similar as possible to the one used in the simulation. To do this the simulation environment was scaled down by a quarter to build the physical environment which had dimensions of 1.25 by 1.25 m and was constructed in the Digital Signal Processing Lab at the University of Cape Town. Wooden blocks with a height of 20 cm were used to create the walls of the various rooms. The completed physical testing environment can be seen in Figure 4.12. The scaling down of the environment had the added effect that most of the room could be seen by the sensors of the robot as compared to the simulation environment where the sensors would sometimes return an out of range reading, resulting in a potentially small boost in performance in the practical experiment.

Figure 4.12: Figure showing room environment that was used for the physical experiment.

4.5.2 Mobile Robot

In keeping with the other experiments a simple two-wheeled robot based on the e-puck robot was designed for use in the physical experiment [48]. It consisted of multiple layers

of Perspex stacked together which house the sensors and the microcontrollers with two DC geared motors at the bottom and used a model developed by Dirck Snoek Hoemanss[**?**]. The robot can be seen in Figure 4.13 below.

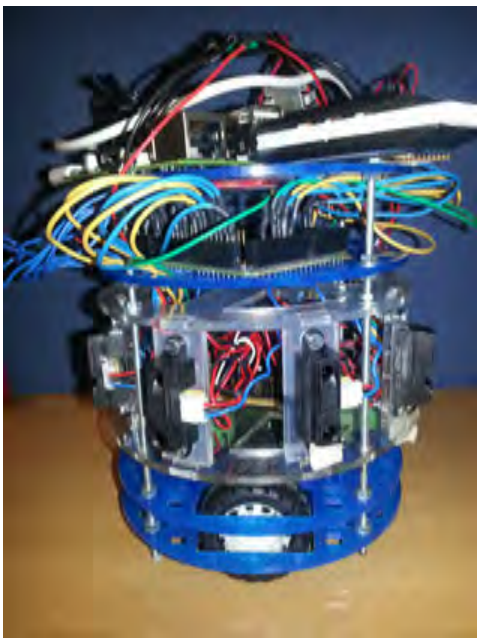


Figure 4.13: Figure showing the mobile robot designed and built for the physical experiment.

The various sensors and microcontroller used are described in the following section.

Raspberry Pi

The Raspberry Pi is a low-cost Linux based computer developed by the Raspberry Pi foundation shown in Figure 4.14 It has a 700 MHz ARM11 ARM1176JZF-S core CPU on-board as well as 512 MB of RAM [51]. It also includes multiple GPIO pins as well as dedicated pins for Serial Peripheral Interface Bus (SPI), I2C and Universal asynchronous receiver/transmitter (UART) communications. It is powered off a 5 V power supply and includes 2 USB ports for attachment of peripherals and a HDMi output to a monitor.

This was therefore the main source of the robot's processing power and was used to perform all the high-level functions in the robot which would be implementing the biologically based networks. The Raspberry Pi was designed to operate as a stand-alone computer with a visual output and therefore to operate the robot autonomously a secure shell (SSH) was used from another computer via a wireless network. Wireless connectivity was provided on the Raspberry Pi via the wireless dongle from D-link shown in Figure 4.15 below.

Figure 4.14: Figure showing the raspberry Pi computer.

Figure 4.15: Figure showing the wireless dongle connected to the raspberry Pi computer.

Arduino

For the low-level control of the robot an Arduino Mega 2560 was used shown in Figure 4.16. This was used to interface with all the sensors as it had multiple I/O pins as well as to control the motors speed and direction. It consists of 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator an ICSP header and a reset button [52].

Line sensing

For line sensing the QTR8RC Reflectance Sensor Array from Pololu [53] was used. This array consisted of 8 Infrared LED and phototransistor pairs. Each pair provides an analog

Figure 4.16: Figure Showing Arduino Mega 2560

reading of reflectance which can then be interpreted to indicate when one of the pairs is over a line or not. Figure 4.17 shows how the sensor was mounted.

Figure 4.17: Figure Showing Mounting of The Line Sensors

Proximity Sensors

The proximity sensors used for the robot were the Sharp GP2Y0A21YK sensors [54]. These had a range of 10-80 cm and produced an analog output that represented this distance. 8 proximity sensors were mounted at equal spacing around the circumference of the robot as shown in Figure 4.18. The line sensors themselves produce a non-linear voltage to distance output and have to be linearised and conditioned before use. The procedure for which is found in [54].

Figure 4.18: Figure Showing Proximity sensors mounted on the robot

Interfacing

The various sensors and microcontrollers were then connected together and the various interfaces needed to communicate with each other established. A diagram showing the connections between components in the robot as well as the communication protocol used between them is shown in Figure 4.19

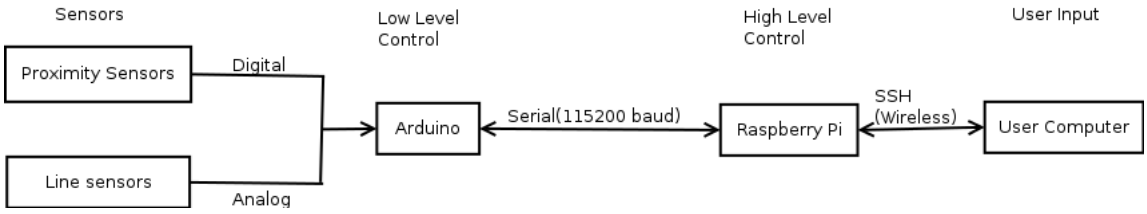


Figure 4.19: Figure showing the interfacing between the sensors and controllers used in the physical robot.

4.5.3 Experiment Task

The task for this experiment was the same as that of the simulation task namely one of simple navigation. The robot, from its starting position in either room 1, 2 or 3, should navigate to a different target room whose location would be given by the user.

For the teach stage of this navigation task, a black line was laid out from the starting room to the goal room. Due to the practical difficulties of laying down multiple paths

in the physical environment, the starting position in any given room as well as the end position in a target room remain fixed for the whole experiment and are not verified to be optimal. This sub-optimality does not however hinder the experiment in any way as teach and repeat as mentioned earlier is agnostic to optimality of the teach path. An example of a path from room 1 to room 2 is shown in Figure 4.20 below.

Figure 4.20: Figure Showing path from Room 1 to Room 2

In this experiment unlike in the simulation however no localisation is done by the robot itself, instead we explicitly state where the robot is to begin with and what target room it should head to as well as how long it should run for. This was done by SSH into the raspberry Pi from a terminal of a Macbook Air with a 1.6 Ghz Intel Core i5 processor and 4 GB DDR3 RAM running OS X 10.9.4. An example of the user input is shown in Figure 4.21 below.

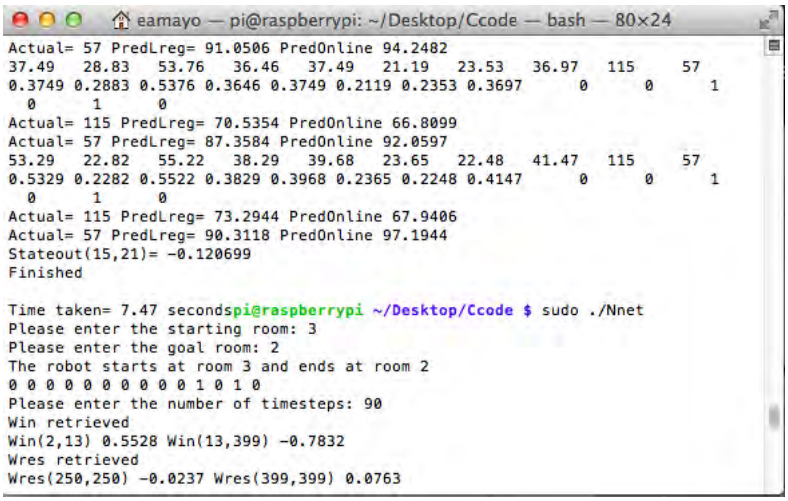


Figure 4.21: Figure showing user input to the robot via SSH connection. The user inputs the starting location of the robot, its target position as well as the length of the run.

Once these values have been input, the robot can begin following the line and executing

the teach section of navigation. To follow the line the following pseudo code was used to control the robot.

```
Result: Line Following
i=0;
while i is less than number of timesteps do
    RightMotorSpeed=100%;
    LeftMotorSpeed=100%;
    if Line Detected by LeftLineSensors then
        RightMotorSpeed=25%;
        LeftMotorSpeed=100%;
    end
    if Line Detected by RightLineSensors then
        RightMotorSpeed=100%;
        LeftMotorSpeed=25%;
    end
    i++;
end
```

Algorithm 2: Line Following controller

The data collection as well as training of the networks progressed in the same way here as in the simulation experiment. The steps needed to train the network for the repeat task can be found in section 4.3.

4.5.4 Code development

One of the greatest challenges in developing an embedded mobile robot lies in the code development. In this robot we are limited to low-cost hardware with limited computational capabilities while in the simulation software a faster computer was used. Therefore more attention has to be paid to various aspects of the code development.

The most important aspect considered was the programming language used to implement the networks. For this the low-level language C++ was used to implement the network. C++ was chosen as it was a low-level language and in addition to this also widely supported Objective-Oriented Programming thus allowing a fast implementation of the algorithms developed to be executed. The Raspberry Pi has a g++ compiler which allows C++ programs to be compiled onboard and also saves on developing time.

Both of these biologically inspired networks rely heavily on numerical calculations such as matrix multiplications and summations. Therefore from the outset it was important to have a fast and reliable numerical operations library and for this purpose the Eigen 2 library was chosen [55]. This not only saves on development time as writing a matrix library for the calculations would not be an easy task but it added robustness as this library has been extensively tested and documented.

4.6 System Design

4.6.1 Simulation Experiment

The finalised system diagram for the simulation experiment can be seen in Figure 4.22 summarising the developmant presented from section 4.1 to 4.5. This Figure shows the sequence of steps involved in moving the e-puck robot in the simulation experiment. The location of the steps, whether in Matlab or Vrep is also showcased.

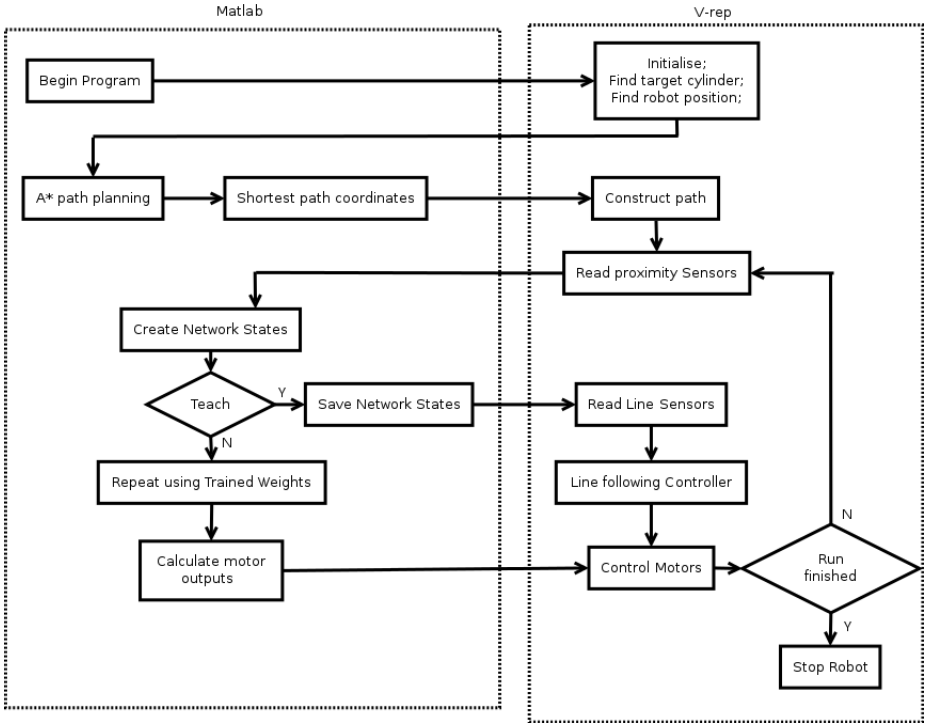


Figure 4.22: Figure showing the system diagram used to perform the simulation experiment. The sequence of steps needed to move the robot in the simulation experiment is shown. The platform that executes these steps, Matlab or Vrep, is also showcased.

4.6.2 Physical Experiment

The finalised system diagram for the physical experiment can be seen in Figure 4.23 and it similarly summarises all the theoretical development presented in section 4.1 to 4.5. This Figure shows the sequence of steps involved in moving the physical robot in the experiment. The platform executing each of the steps is also shown whether it was the user computer, Raspberry Pi or Arduino.

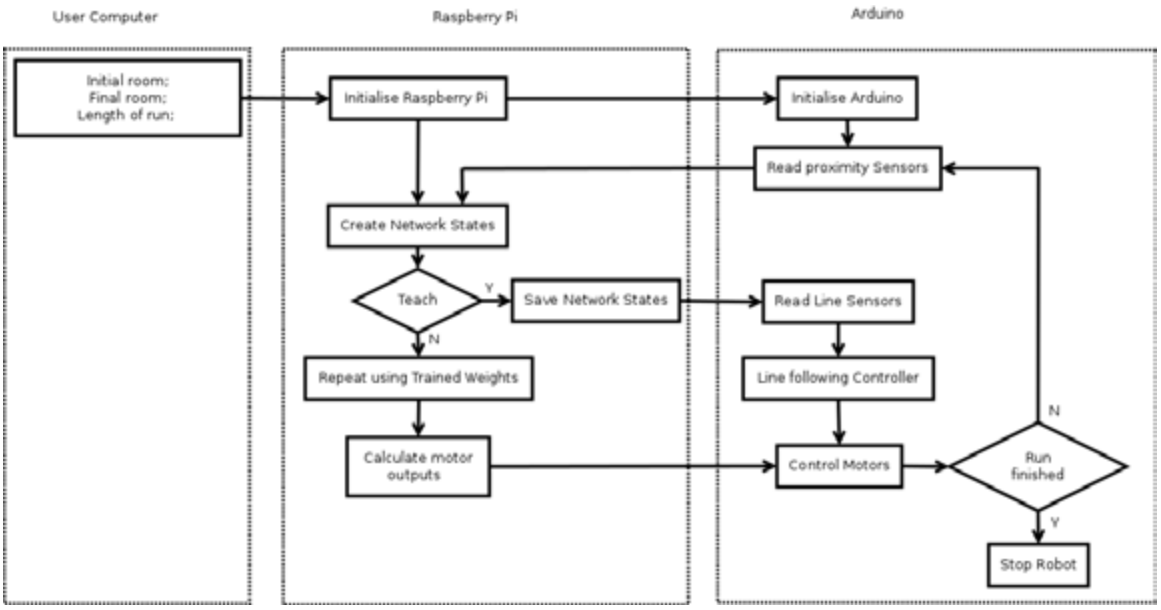


Figure 4.23: Figure showing the system diagram used to perform the physical experiment. The sequence of steps needed to move the robot in the experiment is shown. The platform executing these steps, the user computer, Raspberry Pi or Arduino, is also shown.

Chapter 5

Results

In this section results are presented from both the simulation and physical experiments.

5.1 Simulation Results

5.1.1 Initial Testing

The first step needed in any learning task is the collection of data. Without correct data the networks cannot perform consistently and no evaluation can be carried out. Therefore the first step taken was to establish the ground truth for the simulation task before proceeding to train both of the networks.

To do this the simulated robot was placed in room 1 of the simulated robot environment, with a target goal placed in room 4. The robot then created a path using the A* algorithm and executed it using its line following controller stopping only when it reached its desired target. While the robot was executing its path it also collected data on its current location as well as the proximity sensor data. Figure 5.1 shows the path executed by the robot from the starting position to the end.

Once the robot executed the path to its target the location data was retrieved and evaluated to ensure that the data collected is consistent with the experiment and could be used as ground truth for the training of the neural networks. It must be noted that in the simulation environment we have direct access to the robot's exact location at all

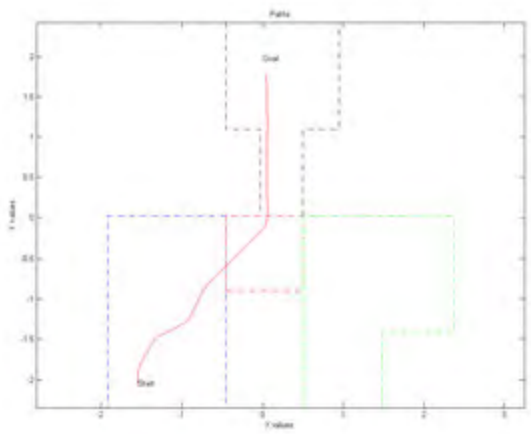


Figure 5.1: Figure showing the path executed by the robot while moving from a starting position in room 1 to a target in room 4. The continuous red line represents the path executed by the robot. The blue, red, green and black dashed lines represent the boundaries of room 1, 2, 3 and 4 respectively.

time while this would not be possible in a physical environment without some form of localisation.

Figure 5.2 below shows the data collected for each of the four rooms throughout the experiment. A one represents the fact that the robot was present in the room at that time while a zero means it was in a different location

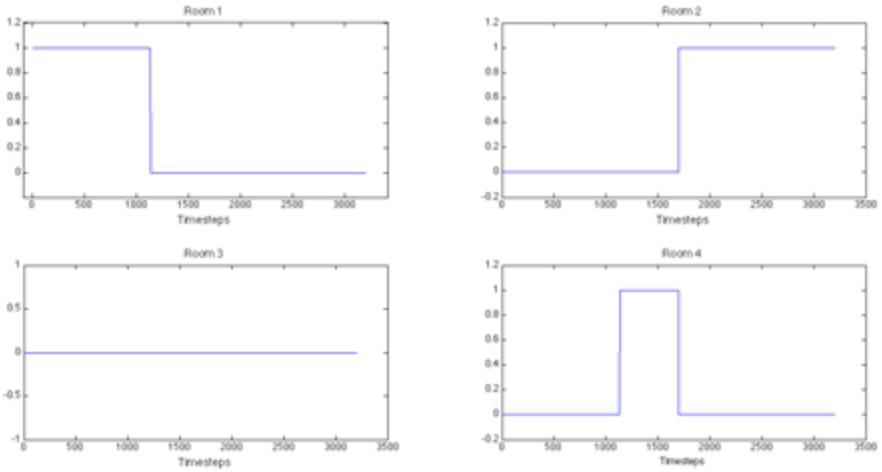


Figure 5.2: Figure showing the ground truth of the location of the robot for the path shown in Figure 5.1. A 1 represents the room the robot is currently in while a 0 means it is in a different room. The robot travels from room 1 to 2 through room 4.

Having established that the data being captured is consistent with what is happening in the experiment and can be used as ground truth the networks were then trained and the outputs compared with the ground truth. Figure 5.3 shows the output of the liquid state

Machine when trained by linear regression to attempt to reproduce the output provided by the ground truth.

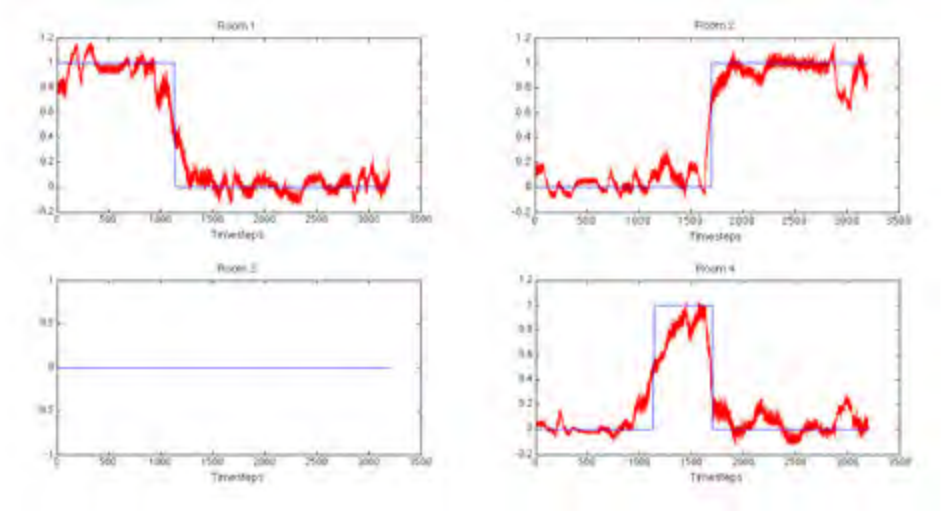


Figure 5.3: Figure of outputs of an LSM after training shown in red compared with the ground truth shown in blue of the robot's location for the path shown in Figure 5.1

Figure 5.4 shows the output of the Echo state Machine when trained by linear regression to attempt to reproduce the output provided by the ground truth.

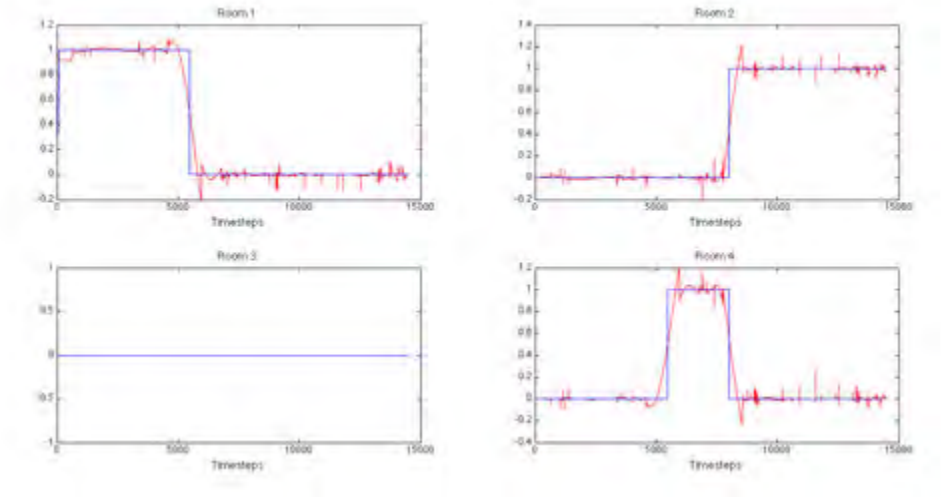


Figure 5.4: Figure of outputs of an ESN after training shown in red compared with the ground truth shown in blue of the robot's location for the path shown in Figure 5.1

From this initial testing shown in Figures 5.4 and 5.3, it can be seen that linear regression in combination with a naive implementation of either the Echo State Network or the Liquid State machine results in an outcome that closely resembles the ground truth. The initial testing serves to confirm that both the ESN and LSM, as they are implemented in this work both have the required computational power needed to perform the classification required in the localisation task. This in the context of reservoir computing means that the ESN and LSM are able to generate internal states from the inputs, here the proximity

sensors of the simulated robot, that are sufficiently different so as to be able to be linearly separable and a memoryless output can be trained to perform the classification.

From these figures it must be noted however that the output of the LSM in Figure 5.3 is however more noisy than the one of the ESN. This can be explained by a closer look at how the states are obtained from the LSM. The states are obtained via an exponential filter of the spiking neurons which have discrete values (either one for spiking or 0 for not spiking) as compared to the continuous numerical states obtained from the ESN. The states of the ESN therefore have more resolution that those of the LSM and are less noisy than the LSM for similar inputs. This translates to less noise in the output as the output is a linear combination of the states.

It is also clear from observation of Figures 5.4 and 5.3 that the ESN operates at a higher frequency as compared to the LSM. While traversing the same path shown in Figure 5.1 the ESN collects almost five times the samples as the LSM. The most obvious implication of this is that, for all the tasks implemented here the Echo State Network will have more data to be trained on than the LSM. While more training data does not necessarily guarantee good results as overfitting of data has been known to happen. It can explain in our case the better performance of the ESN during transitions from one room to another. The higher numerical accuracy of the ESN together with the availability of more data close to the transition points lead to better performance at these points.

5.1.2 Tuning The Echo State Network

From Figure 5.4 an ESN trained by linear regression would appear to be performing well at reproducing the ground truth for the initial task. However, to guarantee optimal performance an investigation into the tuneable parameters of the network was carried out and the effect this has on the network's performance.

Spectral Radius

The first parameter investigated was the spectral radius of the ESN. This, as explained in section 3, is the largest eigenvalue of the $\mathbf{W_{res}}$ matrix in the equation of an ESN shown below.

$$\mathbf{x}(t+1) = \tanh(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}_{res}\mathbf{x}(t) + \mathbf{W}_{biasres}) \tag{5.1}$$

This has an effect on the output of the network and we use the residual of the network which is the squared difference of the actual output given by the ground truth y, and the predicted output from the network $\tilde{y}$ as shown below.

$$\mathbf{R} = (\mathbf{y} - \tilde{\mathbf{y}})^2 \tag{5.2}$$

Different values of the spectral radius were taken and used to create the network states after which training was done, using the ground truth of the path shown in Figure 5.1 to retrieve the weights W_{out} and create the predicted output as below.

$$\tilde{y} = \mathbf{W_{out}} x \tag{5.3}$$

Following this the residual error for different values of the spectral radius was plotted and can be seen in Figure 5.5 below

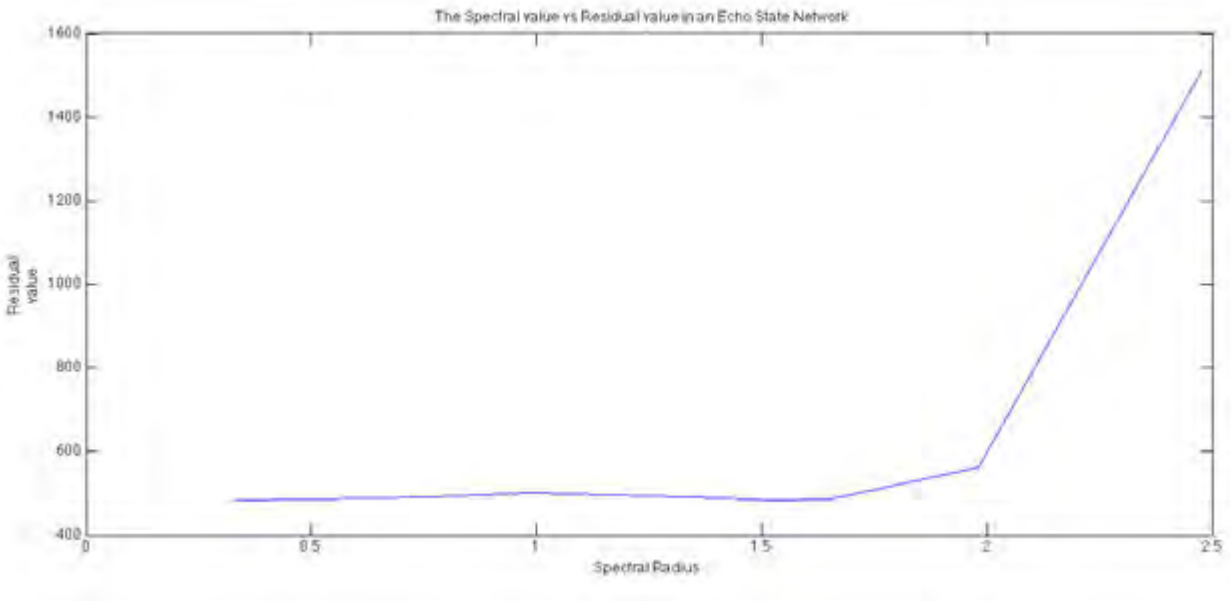


Figure 5.5: Figure showing the Residual value of an ESN for different values of the Spectral Radius

With these results in mind attention was shifted to the tuning of the Echo State Network's parameters. The primary one being its spectral radius. As was mentioned in the methodology theoretically an Echo State Network required a spectral radius of less than one for it to have the "echo state" property. What we see practically from Figure 5.5 is that the optimal value of the spectral radius that would minimise the residual error is greater than one. Values higher than this optimal still result in errors increasing. The significance of this practical implication did not escape Jaeger [15] as he observed that networks without the "echo-state" property can be used as ESNs provided there was no null Input and with sufficient tuning of the Input.

This practical observation relaxes the restrictions on creating an Echo State network. This shows that even a badly initialised Echo State Network still has the potential to perform the localisation task. This shows the strength of this particular architecture as it is robust to changes even in the network makeup itself. This relative large computational power combined with the use of linear regression and mapping from internal states to different external outputs show a great capacity for learning under various conditions.

5.1.3 Tuning the Liquid State Machine

In a similar manner as in the Echo State Network, the LSM contains parameters that need to be tuned to ensure optimal performance.

Tuning the Network parameters

The two parameters considered in the LSM are the network connectivity factor, which changes the probability of two neurons being connected, and the Weight factor which changes the intensity of synaptic activity in the network. A Figure showing the residual value against each of these two factors is shown in Figure 5.6.

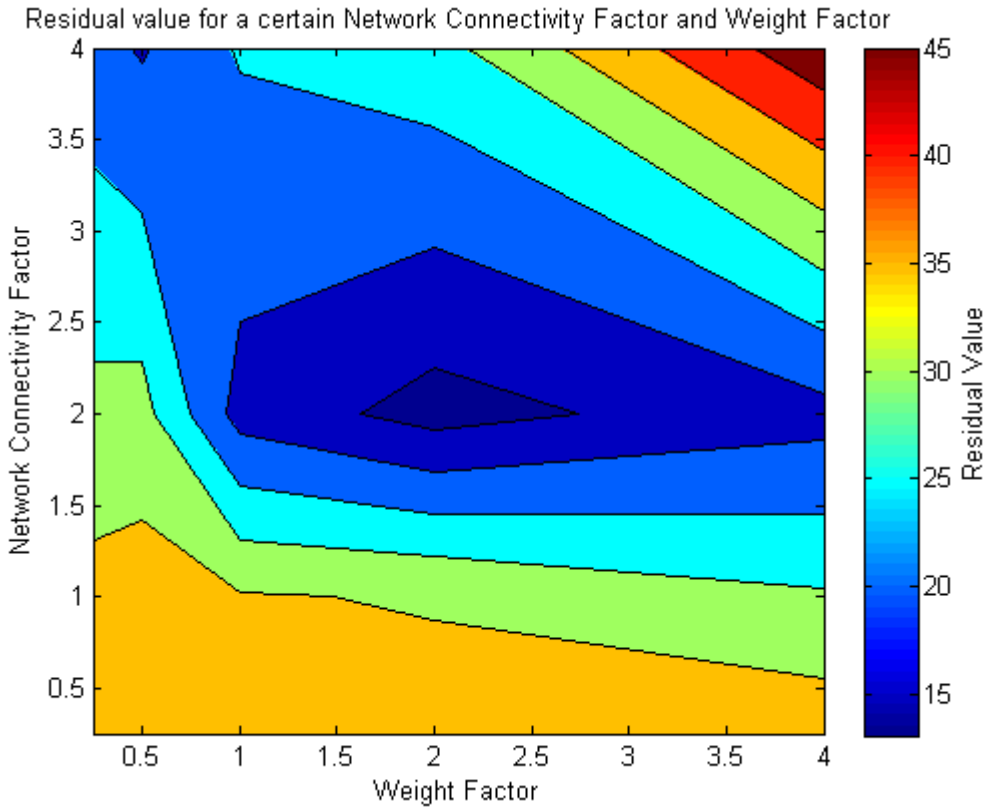


Figure 5.6: Figure showing the residual error obtained when using a LSM for different Network Connectivity and Weight Factors

The tuning of the parameters of the LSM showcases a different problem and one that has not escaped the literature. It is essentially more difficult to tune the LSM, as operating it in the wrong area would lead to bad performance. Therefore a lot of the parameters have been fixed and stay the same. The first parameters investigated were the weight and network connectivity factor which both determined the amount of synaptic activity

in the network. From Figure 5.6 it can be seen that there is a region in which the error is minimised. This corresponds to the region with the maximum computational power or the 'edge of chaos' as referred to in the literature [43].

5.1.4 Tuning the Learning Parameters

Once the parameters of the networks have been tuned, attention can be turned to the learning processes. Both the linear regression and online learning mechanisms were tuned as shown in the section below.

Tuning Linear Regression

For linear regression the only parameter that requires tuning as can be seen in the equation below is the variable λ.

$$\mathbf{R} = (\mathbf{y} - \mathbf{XW})^T(\mathbf{y} - \mathbf{XW}) + \lambda\mathbf{W}^2 \tag{5.4}$$

The value of λ changes how much regularisation occur in the training. Regularisation is important to ensure that the network exhibits good performance when it encounters data that is not in its training set. The error due to lack of generalisation was therefore retrieved using a nested cross validation test as was described in section 3.5.3.

A table of the residual error obtained for different values of λ on an ESN is shown in table 5.1.

Table 5.1: Nested 8-fold cross validation giving the residual error on the room predictions for weights of an ESN retrieved using different values of λ

λ	Room 1	Room 2	Room 3	Room 4
10	113.7694	174.9829	265.0994	438.2374
20	109.4232	166.2128	232.4164	392.3908
80	106.5181	150.1949	189.3857	319.3459
100	106.8859	148.5993	185.6086	312.0099
120	107.4160	147.5983	183.2643	307.1738
160	108.7163	146.6086	180.9313	301.7086
320	114.4255	147.3804	181.8504	299.3152

A table of the residual error obtained for different values of λ on an ESN is shown in

table 5.2.

Table 5.2: Nested 8-fold cross validation giving the residual error on the room predictions for weights of an LSM retrieved using different values of λ

λ	Room 1	Room 2	Room 3	Room 4
0.1	17.9778	47.1584	32.7920	56.4154
0.15	17.9485	46.3301	32.5452	55.4700
0.25	18.1178	45.3704	32.4874	54.6766
0.5	18.7844	44.2994	32.9109	54.7403
0.75	19.4403	43.8239	33.4172	55.5158
1	20.0351	43.5601	33.8853	56.4030
2	21.9406	43.2028	35.4073	59.6473

Once the network itself has been trained, we advance to the training of the learning parameters namely the regularisation for the linear regression as well as the online learning parameters. Table 5.1 and 5.2 show the results of the cross validation tests done on the ESN and LSM respectively. It is clear that an ESN needed a higher regularisation factor than an LSM for the dataset provided. The optimum value that reduced the residual value on the ESN was about 120 while on the LSM it was about 0.5. This can be attributed to the fact that the ESN collects more samples than the LSM and as such any training would run into overfitting when small differences between states, that should be considered as noise, are taught to the model. This affects the performance at generalising to new data which the cross validation test examines.

Tuning The Online learning Parameters

The next parameters that need tuning are the online learning parameters, These are governed by the equation below

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha_{onlearning} 2e\mathbf{x} - \lambda_{onlearning}(\mathbf{w}^2 - 1)\mathbf{w} \tag{5.5}$$

Choosing $\alpha_{onlearning}$

The first parameter tuned is $\alpha_{onlearning}$ which is the learning rate for this method. A graph of various $\alpha_{onlearning}$ values against the residual error on a LSM is shown in the Figure 5.7 below.

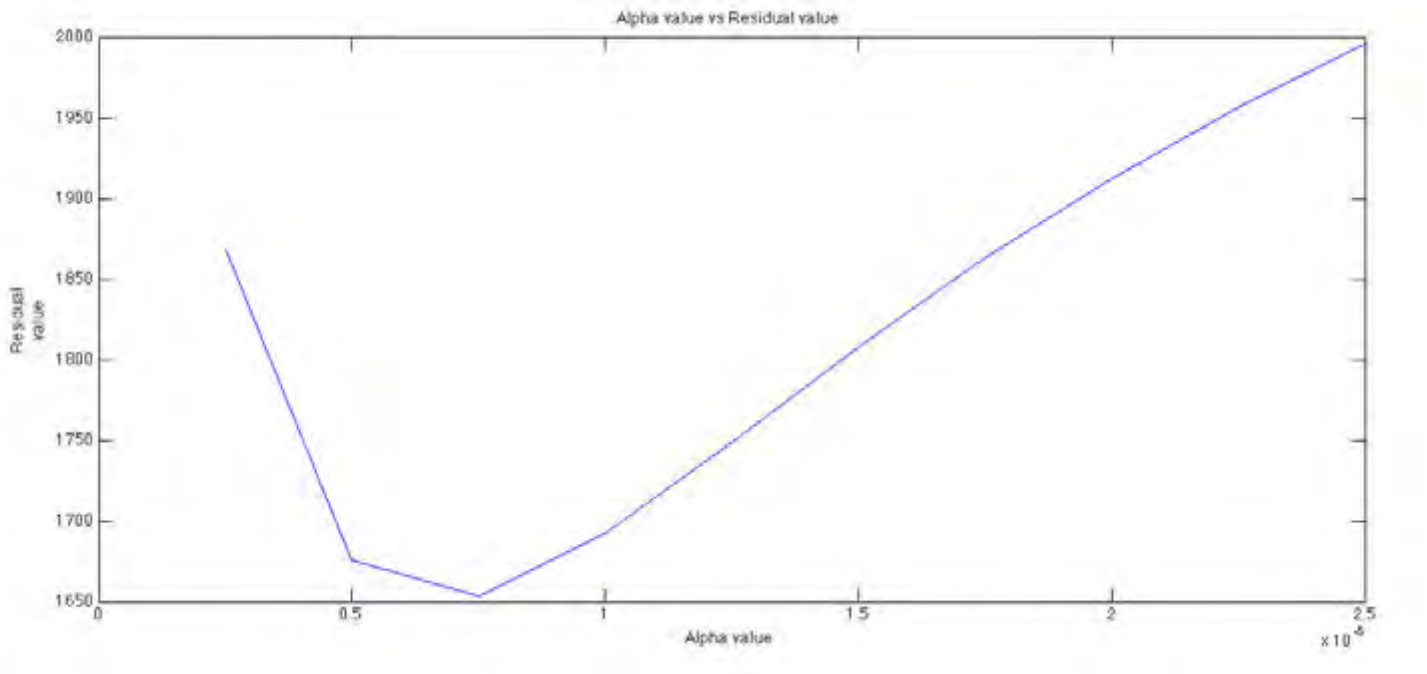


Figure 5.7: Figure showing the residual error for different values of $\alpha_{onlearning}$ on the LSM

Choosing $\lambda_{onlearning}$

While selecting the optimal value of $\lambda_{onlearning}$ it is important that it is also compared to other values of $\alpha_{onlearning}$ to ensure both the parameters are optimal for the data. A Figure showing the residual value for different values of $\lambda_{onlearning}$ and $\alpha_{onlearning}$ is shown in Figure 5.8 below.

In the case of online learning the values $\alpha_{onlearning}$ and $\lambda_{onlearning}$ must be tuned. From Figure 5.7 it can be seen that an optimum value of $\alpha_{onlearning}$ that minimises the residual error on a certain dataset can be found. In the case of $\lambda_{onlearning}$ it becomes much more difficult to tune this value as Figure 5.8 shows, the residual error does not give any information on the optimal value of $\lambda_{onlearning}$. This highlights the difficulty of tuning this particular learning algorithm as a test on regularisation can not be done easily. In the case presented here the entire dataset was available, however practical online learning does not usually involve storing of all the samples. This means that any error metric that involves more than one sample would be difficult to implement.

It must also be mentioned that the rate of convergence for online learning even this optimum value is also very slow, the entire dataset needed to be presented to the algorithm about 100 times before good performance was exhibited. This equates to driving through a path repeatedly about 100 times before the weights are properly trained.

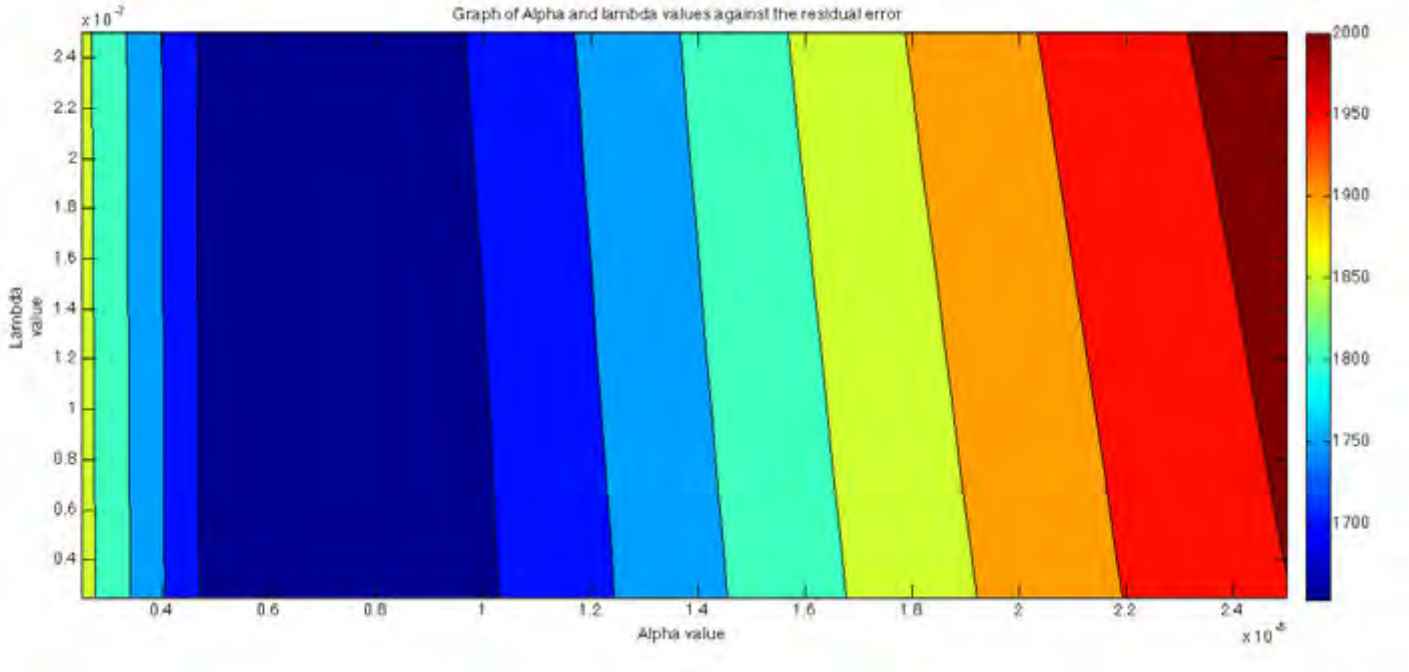


Figure 5.8: Figure showing the residual error for different values of $\alpha_{onlearning}$ and $\lambda_{onlearning}$ on the LSM

5.1.5 Localisation Task

With all the parameters tuned for both of the networks focus shifts to the performance at performing the localisation task. For this a representation of location called the occupancy grid is used which compares the ground truth collected by the robot with the predicted positions coming out of the networks. The first case considered is that of the performance of the ESN on its training dataset. Figure 5.9 and 5.10 below shows the result of this task after the ESN has been trained by linear regression and 100 steps of online learning.

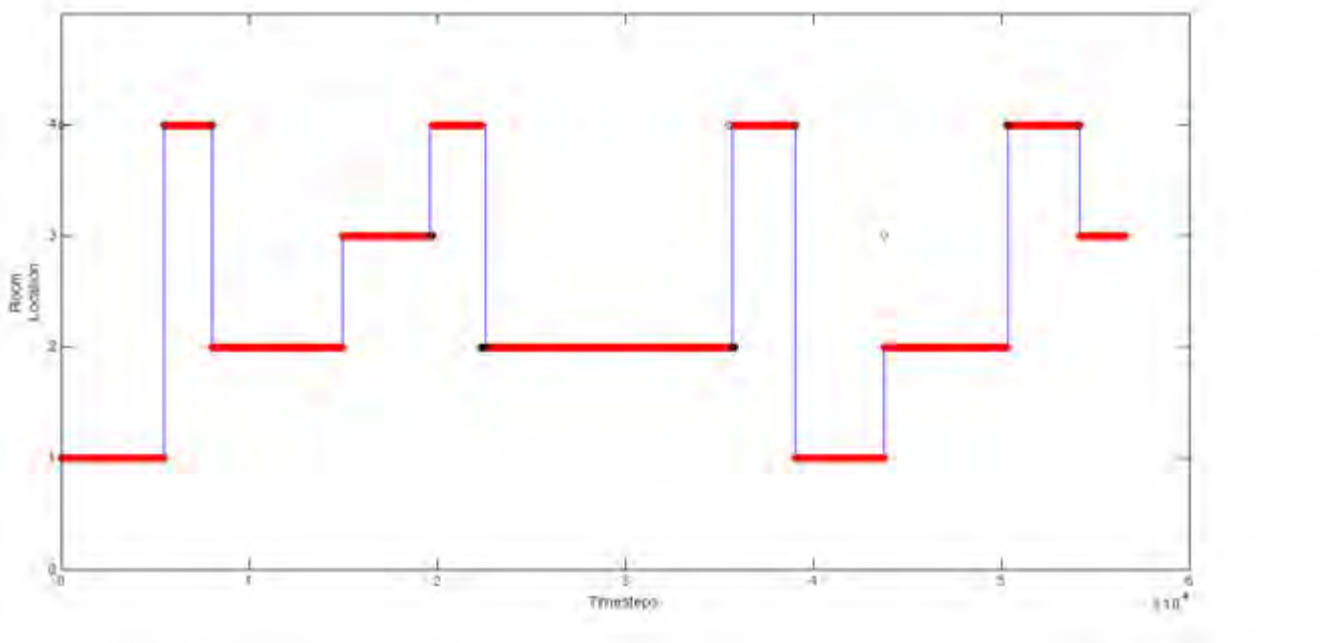


Figure 5.9: Figure of the occupancy grid comparing the ground truth of locations collected by the robot in blue and the outputs of an ESN trained by linear regression. Correct outputs of the ESN are shown by red circles while mispredictions are shown by black circles.

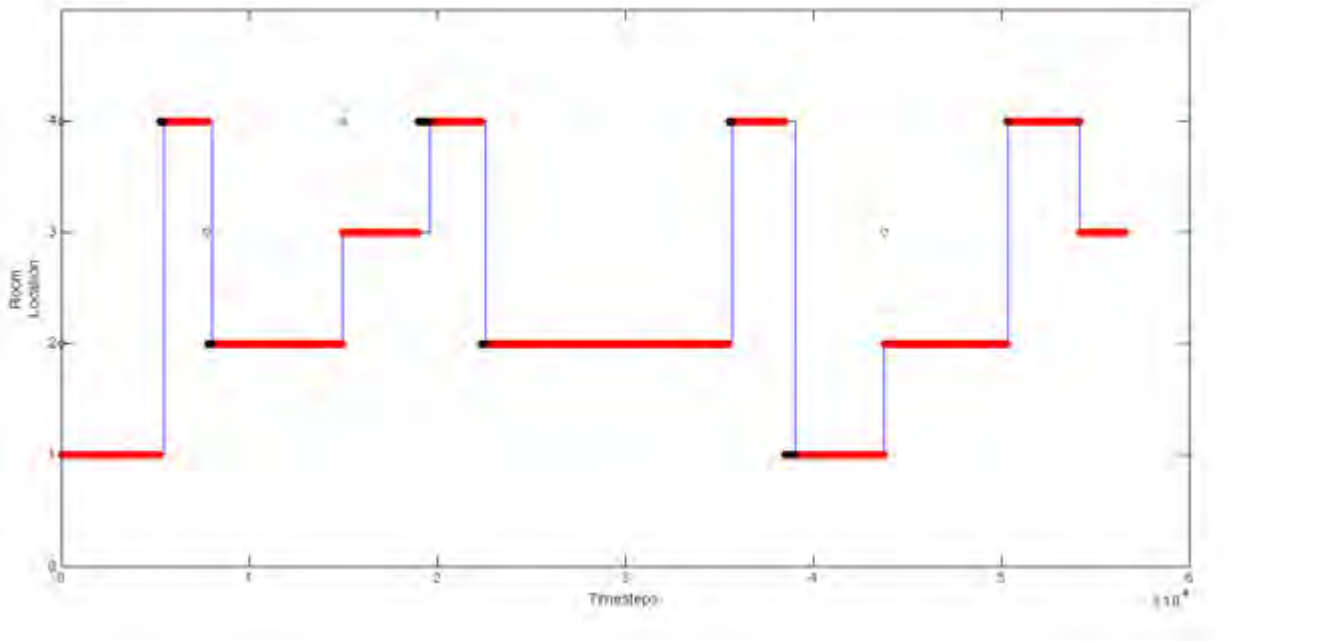


Figure 5.10: Figure of the occupancy grid comparing the ground truth of locations collected by the robot in blue and the outputs of an ESN trained by online learning. Correct outputs of the ESN are shown by red circles while mispredictions are shown by black circles.

The same test is then performed using a LSM and the results shown in Figure 5.11 and 5.12 below.

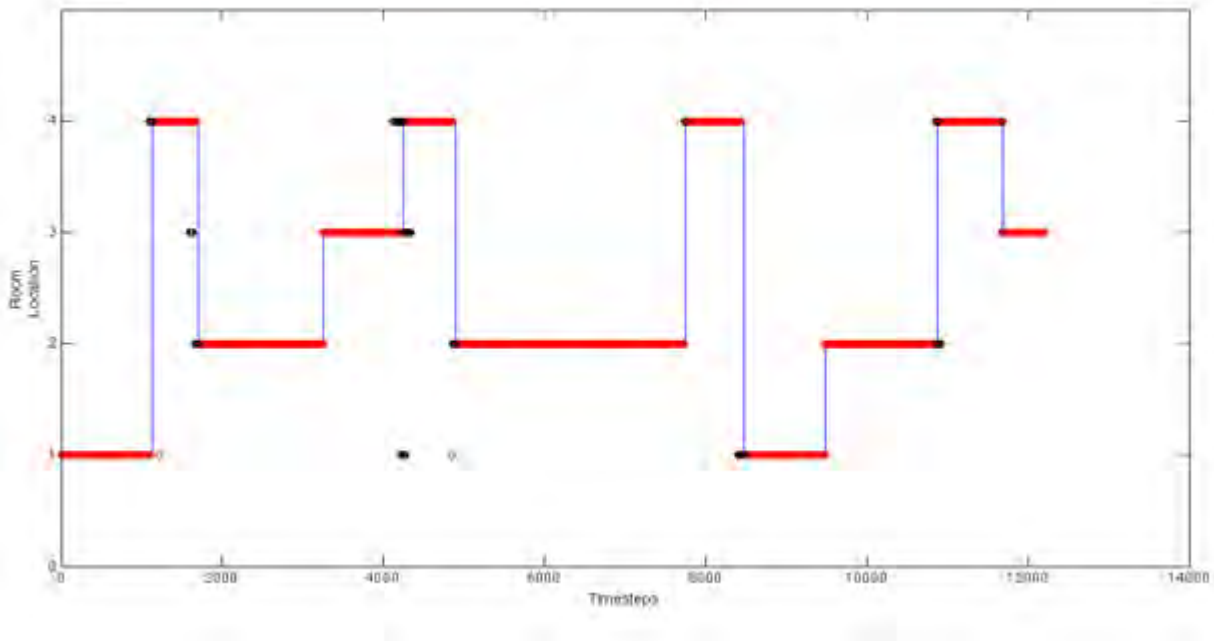


Figure 5.11: Figure of the occupancy grid comparing the ground truth of locations collected by the robot in blue and the outputs of a LSM trained by linear regression. Correct outputs of the LSM are shown by red circles while mispredictions are shown by black circles.

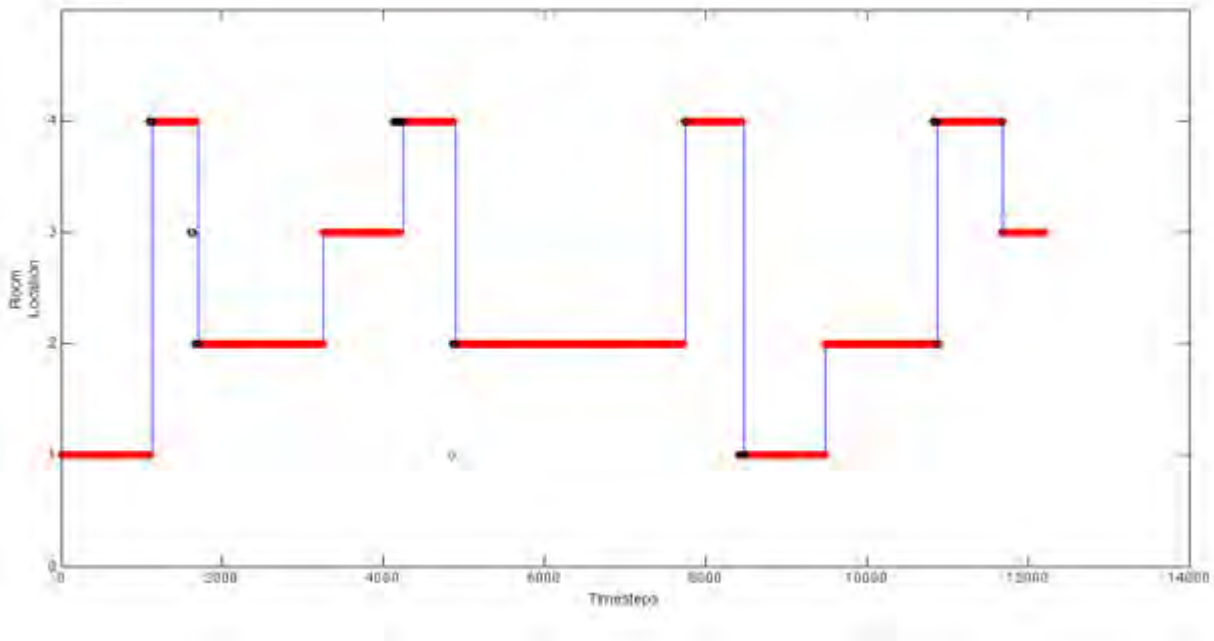


Figure 5.12: Figure of the occupancy grid comparing the ground truth of locations collected by the robot in blue and the outputs of a LSM trained by online learning. Correct outputs of the LSM are shown by red circles while mispredictions are shown by black circles.

With the learning and network parameters tuned to the values presented above, a comparison of the performance of the ESN and LSM was carried out. Figures 5.9, 5.10, 5.11 and 5.12 show the performance on the training dataset of an ESN trained under linear regression, one trained by online learning, an LSM trained by linear regression and one trained by online learning respectively in the localisation task. All the figures showed good localisation performance. It can be seen however that the ESN trained by linear regression outperforms the LSM trained by the same method especially at the transitions between rooms as was postulated in the beginning of this section. The performance of online learning on this training dataset was also very good even though it was still worse than that given by linear regression in each of the systems.

Although the performance of online learning will always be worse than that of linear regression, the opportunities opened up by the availability of an online learning tool more than make up for this. The first gain is in memory storage, with linear regression the entire dataset must be stored, this includes all the states of the network and their corresponding outputs while for online learning only the current states and their outputs must be stored before the training step. With large datasets especially when dealing with low-cost robotic systems it is not usually feasible to include a large bank of memory to store the data which becomes redundant after the training has occurred. Using online learning therefore translates to a large savings in low-costs systems and a greater systems efficiency as all the memory needed is used during normal system operation leading to a better product. The major disadvantage with online learning is that the convergence can not be guaranteed and especially where different behaviours are required it might take multiple runs for the system to achieve reasonable performance. If the conditions vary greatly during each of these runs the final output would also not be a true optimum as it has been taught using continuously varying data and this would affect performance.

The figures above show that both the networks after training were most of the time able to correctly predict their location. These locations and outputs however were part of the training dataset and to ensure performance the same test must be run on data that the robot had not seen before or been trained on. A path was constructed as shown in Figure 5.13 and the task of the robot was to predict its position as it traversed the path using the previously trained weights and networks.

The results of this run on the localisation task using an ESN trained by linear regression are shown on the occupancy grid in fig 5.14.

The results of this run on the localisation task using an ESN trained by online learning are shown on the occupancy grid in fig 5.15.

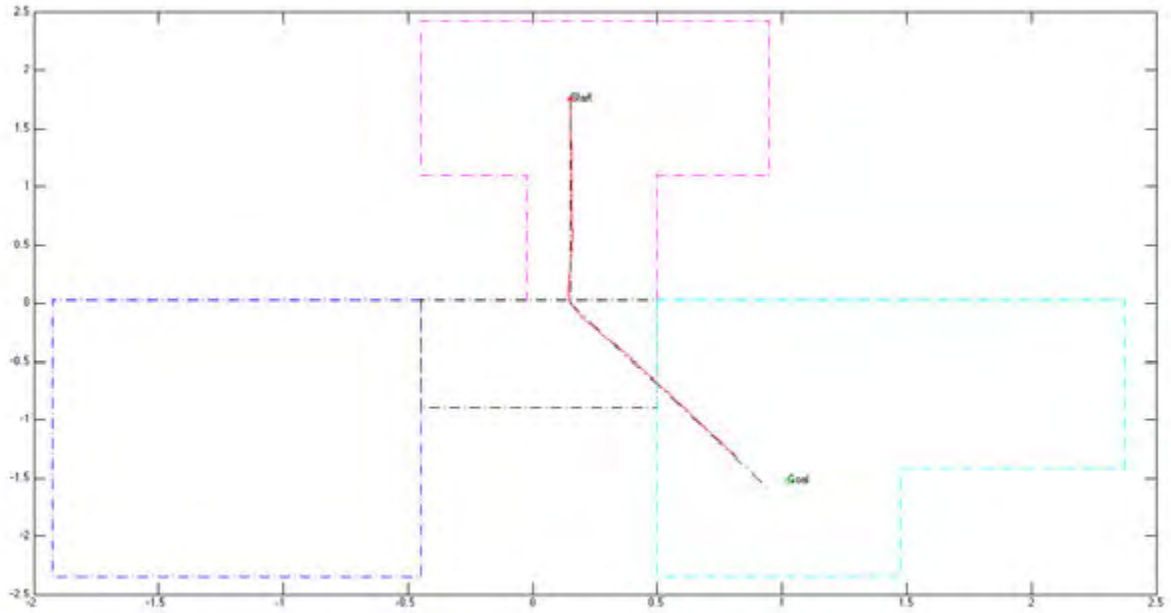


Figure 5.13: Figure of the path taken to validate performance in the localisation task. The robot begins at room 1 and ends up at room 2.

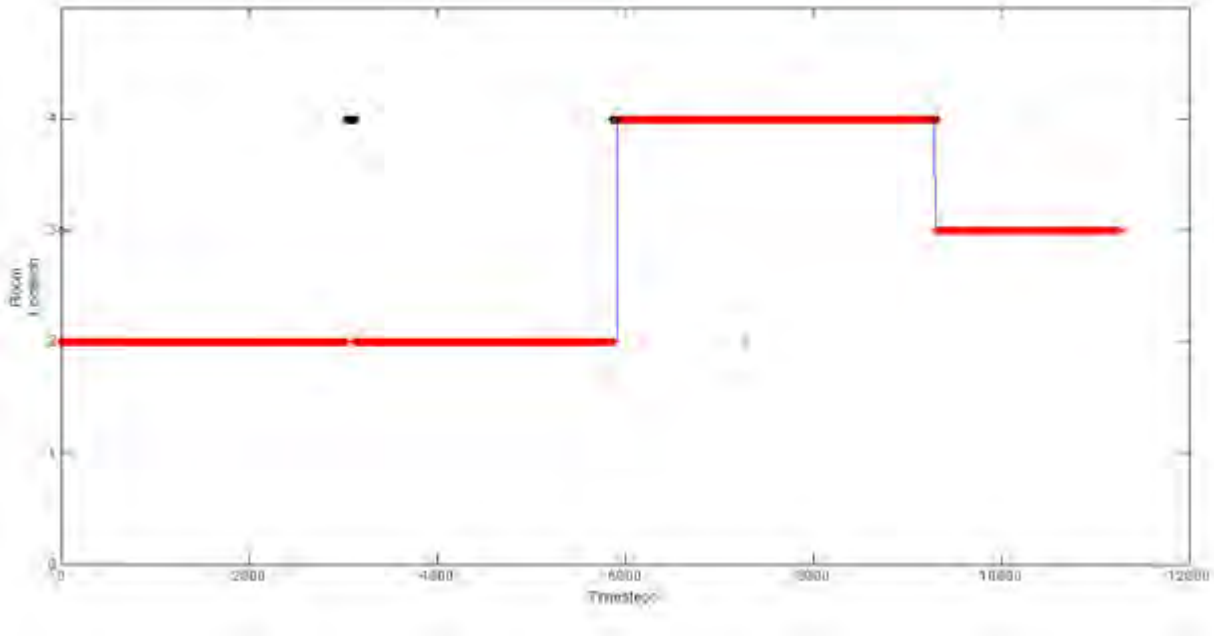


Figure 5.14: Figure of the occupancy grid showing the predicted room location of an ESN trained by linear regression against the ground truth for the path in Figure 5.13. Correct outputs of the ESN are shown by red circles while mispredictions are shown by black circles.

The results of this run on the localisation task using a LSM trained by linear regression are shown on the occupancy grid in fig 5.16.

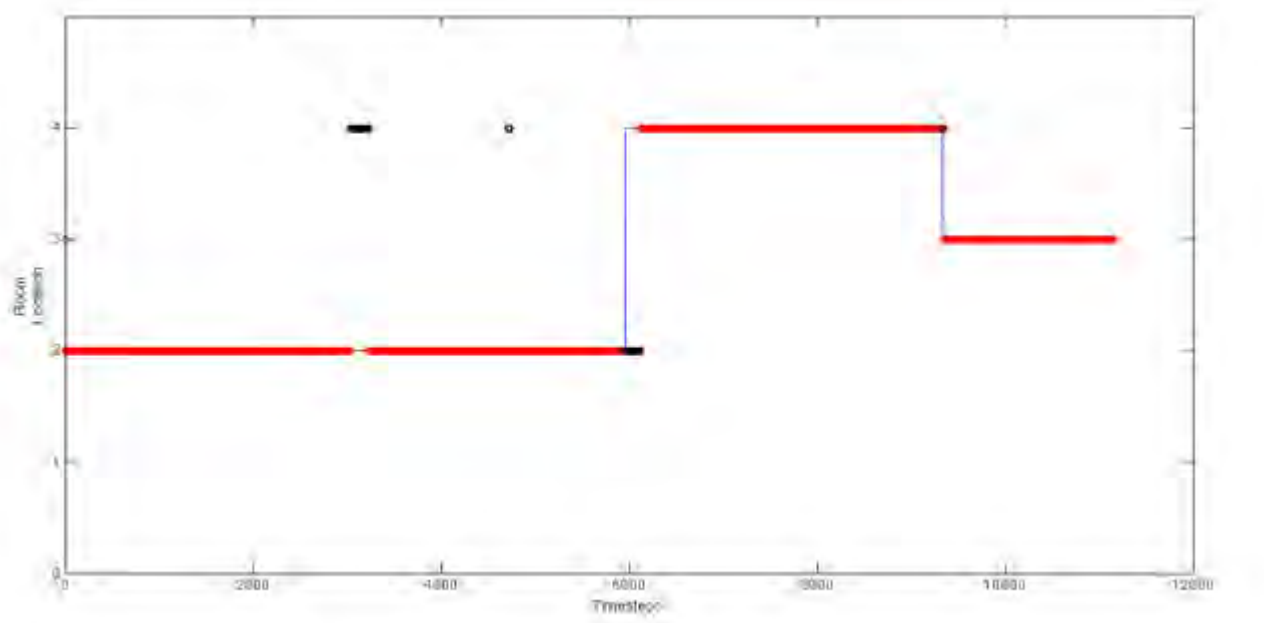


Figure 5.15: Figure of the occupancy grid showing the predicted room location of an ESN trained by online learning against the ground truth for the path in Figure 5.13. Correct outputs of the ESN are shown by red circles while mispredictions are shown by black circles.

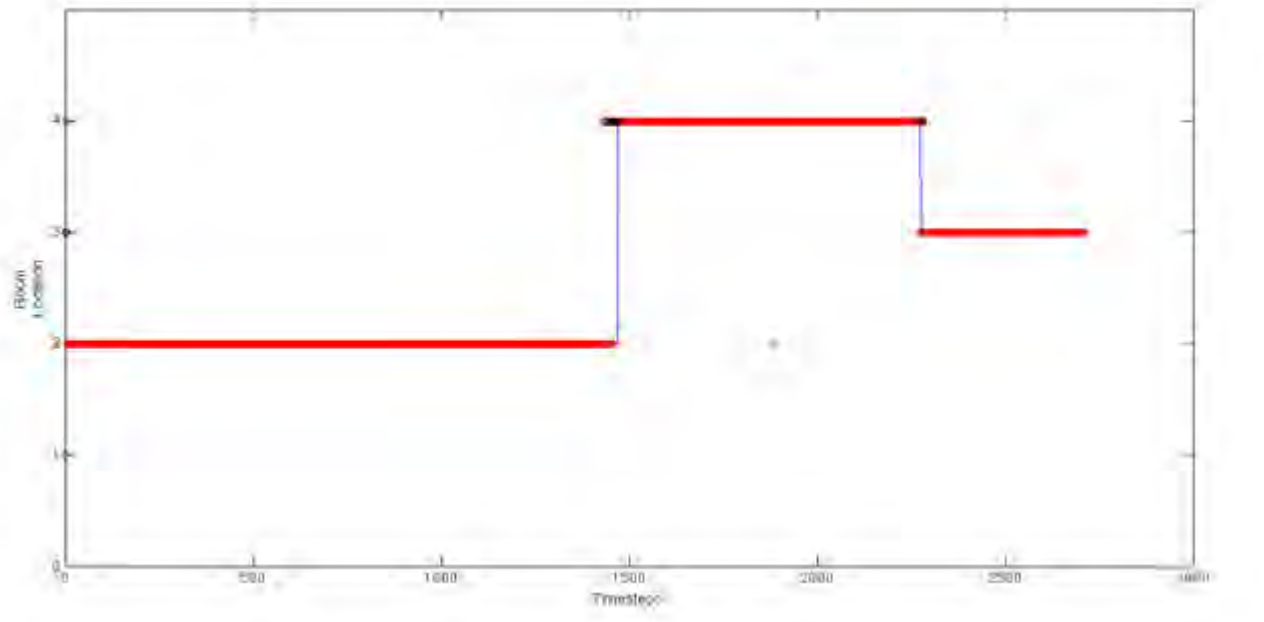


Figure 5.16: Figure of the occupancy grid showing the predicted room location of an LSM trained by linear regression against the ground truth for the path in Figure 5.13. Correct outputs of the LSM are shown by red circles while mispredictions are shown by black circles.

The results of this run on the localisation task using a LSM trained by online learning are shown on the occupancy grid in fig 5.17.

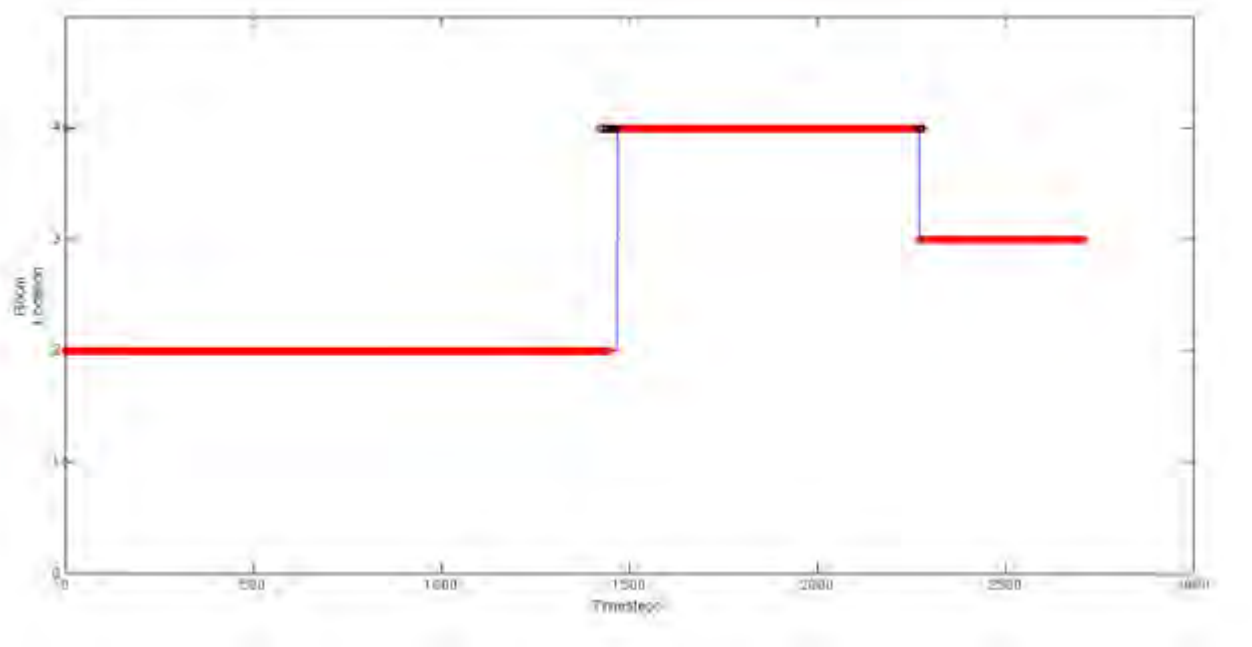


Figure 5.17: Figure of the occupancy grid showing the predicted room location of an LSM trained by online learning against the ground truth for the path in Figure 5.13. Correct outputs of the ESN are shown by red circles while mispredictions are shown by black circles.

Figures 5.14, 5.15, 5.16 and 5.17 show the occupancy grids for the area under simulation with the predictions coming from an ESN trained by linear regression, one trained by online learning, a LSM trained by linear regression and one trained by online learning respectively. From each of these figures it can be seen that the robot is able to localise itself in the room while traversing a path that was not part of the initial dataset used for training. Both of the networks exhibit errors during initialisation as well as transitions but the ESN however had a lower accuracy at localisation than the LSM, this was clear as it misidentified some sections of room 1 as room 4, the ESN trained by linear regression in Figure 5.14 did however outperform the one trained by online learning in Figure 5.15.

That being said the performance of all the networks was very good. The misclassification right at the beginning of the robot's initialisation is due to the fact that the robot does not have sufficient information about its environment, once information builds up through the next states it is then able to localise itself in the environment. This is to be expected in any recurrent network as it takes a few steps for recurrent activity to 'build up' and only then would this translate to accurate results. The worse performance by the ESN in this task even though it outperformed the LSM in the training data set is due to a lack of generalisation. The larger number of samples used in the training cause the ESN to suffer from overfitting much more than the LSM and this reduces its performance when presented with data not in its training set.

5.1.6 Combined Task

With the parameters tuned, path-planning using the networks can now be achieved. The following figures show the path-planned by the ESN and LSM respectively from each of the different rooms to a target in another room.

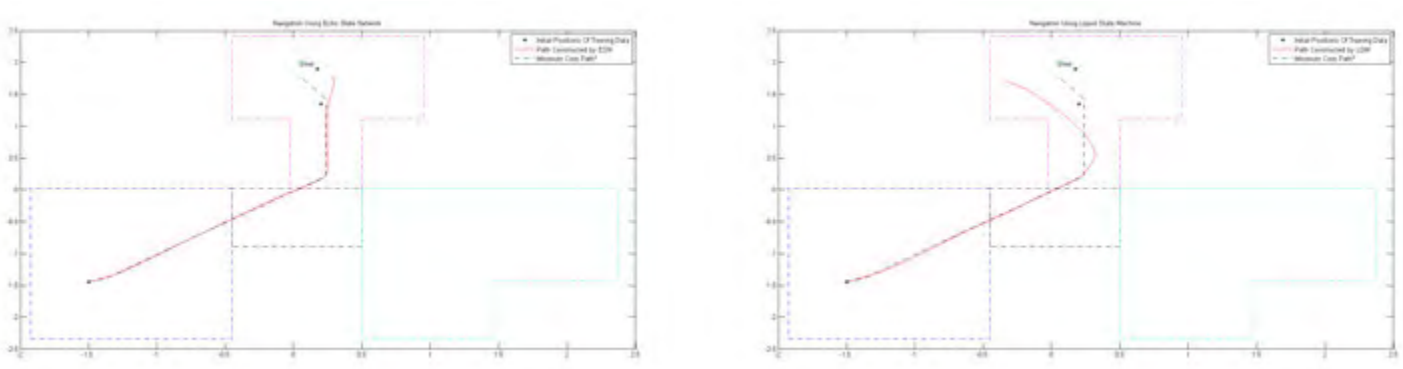


Figure 5.18: Figure showing the path constructed by the networks from room 1 to 2

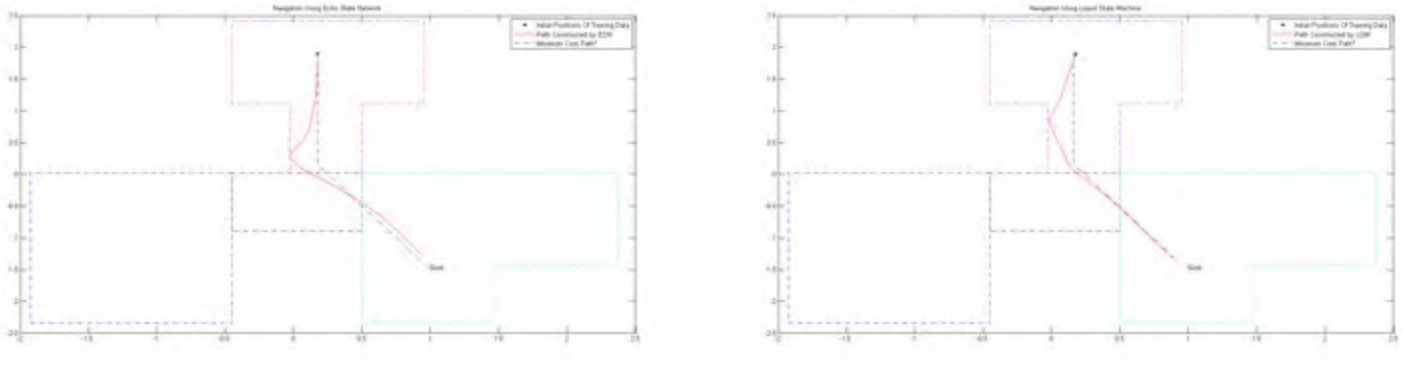


Figure 5.19: Figure showing the path constructed by the networks from room 2 to 3

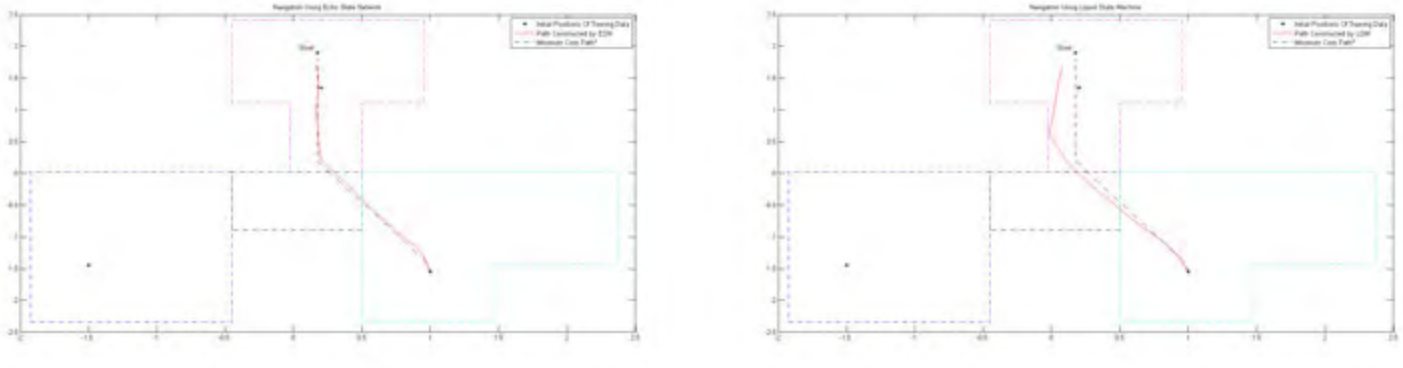


Figure 5.20: Figure showing the path constructed by the networks from room 3 to 2

Attention was then shifted to the other aspect of the experiments which was whether or not the robot was able to fulfil the navigation task that is to be able to move from a start

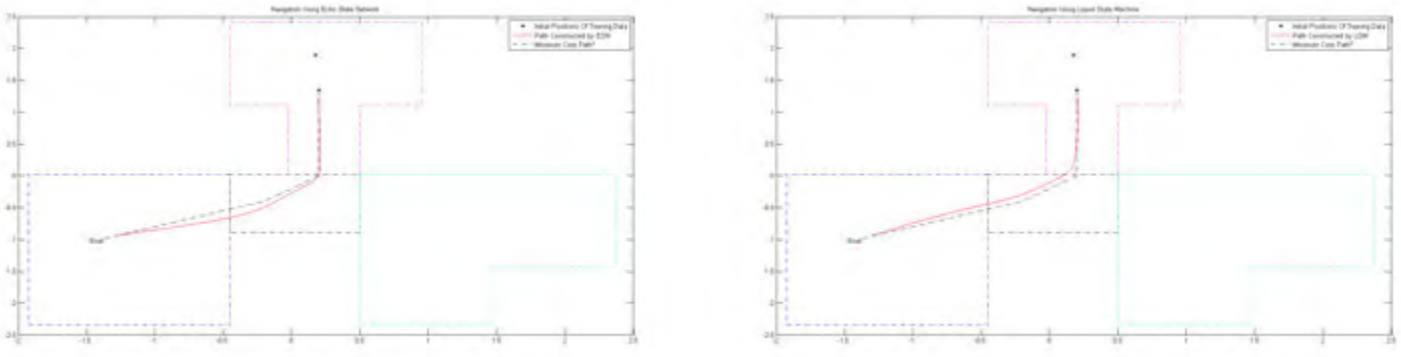


Figure 5.21: figure showing the paths constructed by the networks from room 2 to 1

position in one of the rooms, either simulated or experimental and to be able to move to a target position in another room. In all the experiments shown in Figures 5.18, 5.19, 5.20 and 5.21 the robot was able to successfully move to the target in the other room.

Looking specifically at the results from the robot moving from room 1 to room 2 when under control from both the ESN and the LSM in Figure 5.18 it becomes clear that the ESN controller works much better than that of the LSM. Both of the controllers are able to make both of the required turns to manoeuvre the robot to room 2, the first being a sharp right into room 4 followed by an even sharper left to room 2, but the ESN is the only one that does this without colliding with the walls in room 2. This can be partly attributed to the fact that the LSM produces outputs at a frequency lower than that of the ESN. This is a lack of realtime action and there is a considerable time difference, about a second in this case, between when the inputs are received by the robot and when the outputs are produced. This means that the LSM lags reality and is therefore unable to correct for most of the finer turns needed to ensure that it does not run into the wall and therefore exhibits lesser control than the ESN.

Another issue with the LSM as implemented above, is that its output will always be more noisy than that of the ESN, this inherent noise has a practical effect on the motor operation in that it causes jerking in the motors and hence the robot movement. This leads to the robot not being able to reproduce a perfect trajectory. A lot must be said however of the generalisation and computational power of the LSM in still being able to navigate the robot from the start to the goal. This is firstly because it operates with a sub-sample of the data available to the ESN due to its lower frequency and also the error introduced by driving the motor with a noisy input. This results in the LSM straying from the path used in the teaching experiment however it is still able to correct for this and recreate a path up to the finish line.

The presence of a correction and a subsequent recovery time needed to be put in place also puts a limit on the speed of the robot. Any speed faster than that allowed with the recovery time would lead to not only a drop in performance from the ideal but that any sort of performance can not be guaranteed. It must be noted that the recovery time as stated here will also change according to the speed at which the robot was trained . The network, through training, would be able to adapt to changes in data as it arrives at any frequency. However if the data begins to arrive at a higher frequency than that at which the robot was trained at this would lead to errors as have been discussed above.

A point must be made that all these shortcomings of the LSM derive particularly to its implementation in an analytical device such as a computer or an embedded microcontroller. If however this was to be implemented in a more biologically realistic architecture such a neuromorphic device it would experience much better performance. Due to this lag in performance in analytic devices it was decided not to implement the LSM on the experimental platform, the Raspberry Pi, as one iteration of the network would produce an output 2 seconds later which would result in system failure from the get-go. Instead the ESN network was implemented and a comparison of experimental and simulation results was carried out.

5.2 Experimental Results

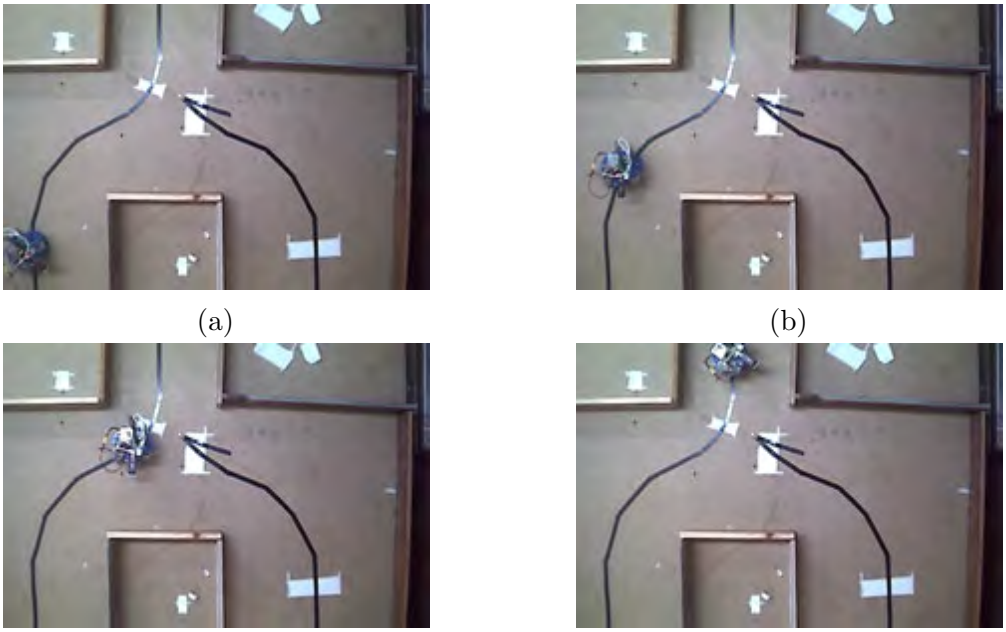
After concluding the simulation experiment focus was then turned to the physical experiment with the aim to replicate the results from the simulation. The robot used for this experiment as well as its operation was described in section 4.5.2. As compared to the simulation experiment where all the data including the robot's position can be easily retrieved from the Vrep the physical environment being indoors does not offer any such information. To have some consistency in the results a webcam was therefore mounted above the experimental platform and videos of the robot were taken while it moved through the environment.

An example image taken from the webcam looking down on the robot environment can be seen in Figure 5.22.

With the experiment set-up as was described in section 4.5.3, the teach and repeat task can be attempted on the robot. This would first and formerly involve teaching the robot a path laid out in the physical environment, using line-following on a black line placed in the environment. While the robot executes this task a video is taken from the webcam

Figure 5.22: Figure of a screenshot taken by the webcam of the physical environment setup.

of its motion for subsequent analysis. Some screenshots showing the robot executing the teach section are shown in Figure 5.23 below



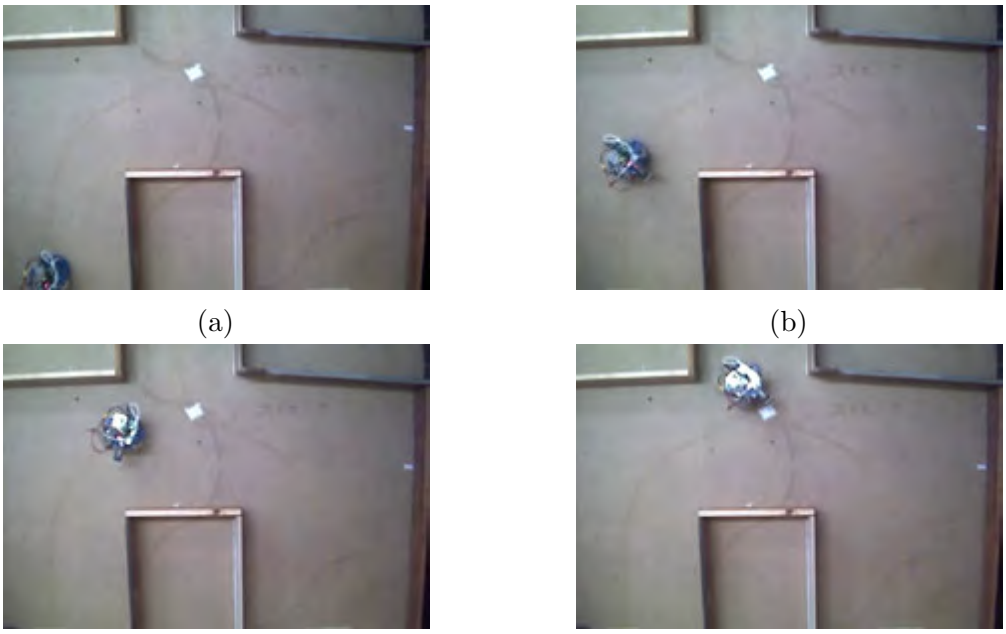
(a) (b)

(c) (d)

Figure 5.23: Figure of screenshots taken as the robot followed a line from room 1 to room 2. taken from video ESNtraining

Following this the network is trained using data from the teach stage to repeat the path given to it. The black line is removed from the environment and the robot is given a target room to navigate to. The results of this repeat section are shown below.

The first results presented are the repeat section as the robot navigates from room 1 to room 2. Figure 5.24 shows some screenshots of this trajectory.

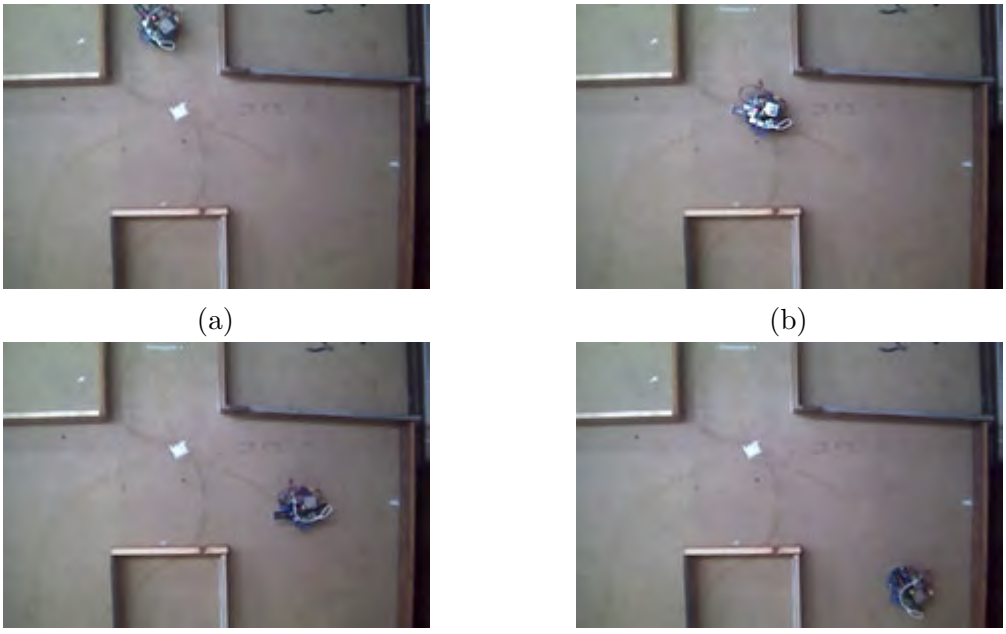


(a) (b)

(c) (d)

Figure 5.24: Figure of screen shots taken as the robot attempted to repeat the path given from room 1 to room 2 using an Echo State Network as a controller. Taken from the video ESNroom1to2

The robot then navigated from room 2 to room 3. Figure 5.25 shows some screenshots as the robot using an ESN as a controller navigated from room 2 to room 3.

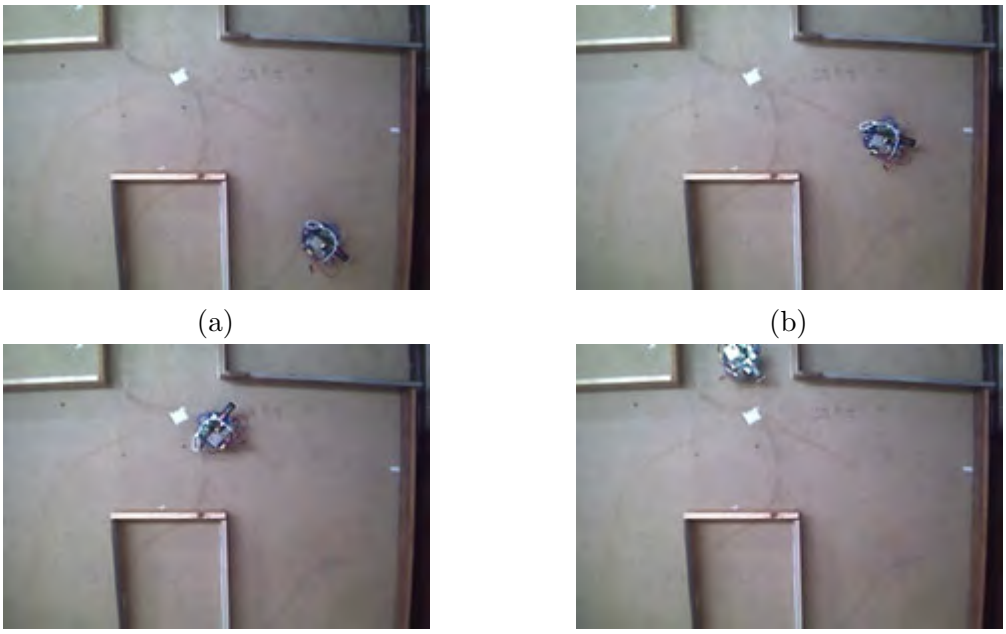


(a) (b)

(c) (d)

Figure 5.25: Figure of screen shots taken as the robot attempted to repeat the path given from room 2 to room 3 using an Echo State Network as a controller. Taken from the video ESNroom2to3

The robot then navigated from room 3 to room 2. Figure 5.26 shows some screenshots as the robot using an ESN as a controller navigated from room 3 to room 2.



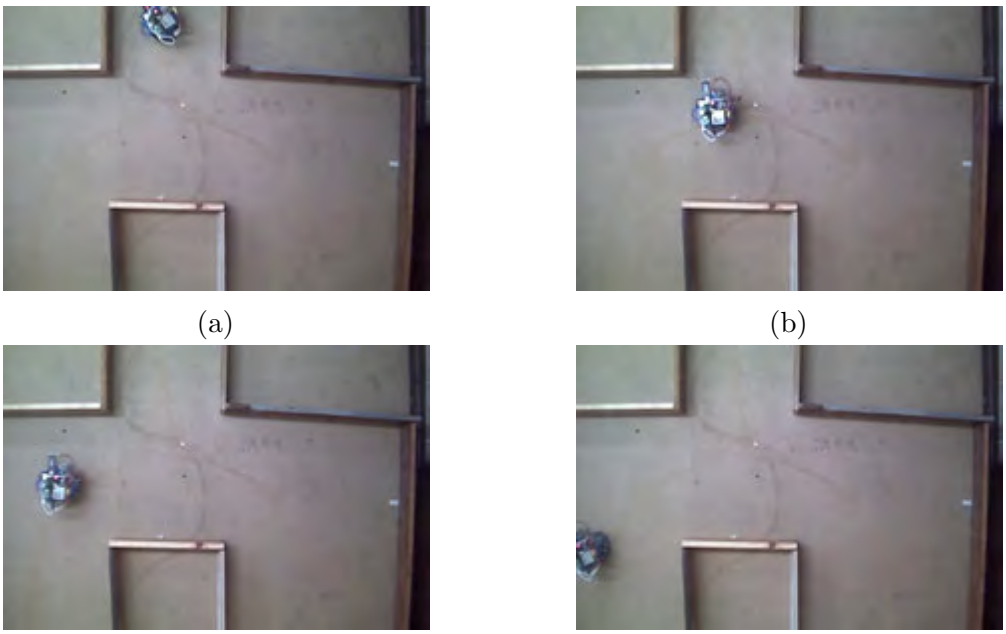
(a) (b)

(c) (d)

Figure 5.26: Figure of screen shots taken as the robot attempted to repeat the path given from room 3 to room 2 using an Echo State Network as a controller. Taken from the video ESNroom3to2

The robot then navigated from room 2 to room 1. Figure 5.27 shows some screenshots as the robot using an ESN as a controller navigated from room 2 to room 1.

What was clear immediately from the physical experiments shown in Figure 5.24, 5.25, 5.26 and 5.27 was that the physical robot succeeded in getting from the initial room to the final room even after all the combination of behaviours had been stored. Therefore the robot was able to recall the past experiences depending on what room it was told it was in and then be able to move to a target in a different room, this was done however without any localisation taking place which made the navigation more complex. This successful navigation in the physical experiment also comes despite the noise exhibited by the proximity sensors that was not seen in the simulation experiment.



(a) (b)

(c) (d)

Figure 5.27: Figure of screen shots taken as the robot attempted to repeat the path given from room 2 to room 1 using an Echo State Network as a controller. Taken from the video ESNroom2to1

Chapter 6

Discussions and Conclusions

From the results presented above it can be concluded that:

- Networks operating on the reservoir computing principle provide a robust and alternative method to solving the navigation problem especially in the context of small and low-cost robotics. This study was seen to confirm the work of Antonelo et al. [13] in simulations before extending this work by testing it on a physical low-cost embedded platform. The matching results obtained from the two experiments verified the suitability of these networks to solving the navigation problem.
- The Least Mean Squares rule from adaptive filters provides a online way of training such networks. While the convergence of this learning rule is difficult to identify, the results obtained by using this rule showed good performance. The availability of such a rule allows the design of more efficient lower cost robots as it reduces the memory requirements of such a system.
- The performance of the more biologically accurate Liquid State Machine although worse than the Echo State Network in some of the tasks goes to show that the biological base for this approach to navigation is strong. Therefore as more realistic models of not only human but rodent brain behaviour are developed in the field of computational neuroscience this could translate directly to better performance in the navigation task for robotics. These developments include the use of neuromorphic devices which eliminate the problems encountered by the Liquid State Machine in this work, namely the simulation of the large number of differential equations in analytical software such as a computer.

Project success

The project was considered a success from a scope perspective as the objectives set out in the beginning were met. The physical experiment results matched what had been seen in the literature and showed that these biologically-based neural networks can indeed be used to solve the navigation problem in low-cost robotics.

Chapter 7

Recommendations

From the discussions and conclusions above the following recommendations were made to improve this work:

- The addition of some aspect of odometry into the system during both the training and implementation stages. Without odometry the controller, either a ESN or a LSM cannot be guaranteed to exactly repeat a specific path as drift which is inevitable to occur cannot be sensed and corrected for. Adding an external sense of odometry to this system will make it more robust and increase its scope of application to areas with high sensitivity to the trajectory of robot.
- Investigate further the Least Mean Squares rule used in this work for online learning. This is to determine how to tune its parameters better as well as investigate methods of increasing its speed of convergence.
- Investigate the performance of a LSM implemented via a neuromorphic circuit. This is to determine the effect and errors caused by simulating large systems of differential equations in analytical machines has on the experiments presented in this work.

Bibliography

[1] M. Milford and R. Schulz, "Principles of goal-directed spatial robot navigation in biomimetic models," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1655, p. 20130484, 2014.

[2] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli, "Local metrical and global topological maps in the hybrid spatial semantic hierarchy," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 4845–4851.

[3] G. Sibley, C. Mei, I. Reid, and P. Newman, "Planes, trains and automobilesautonomy for the modern robot," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 285–292.

[4] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller, "An atlas framework for scalable mapping," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2. IEEE, 2003, pp. 1899–1906.

[5] B. Kuipers and Y.-T. Byun, "A robust, qualitative method for robot spatial learning." in *AAAI*, vol. 88, 1988, pp. 774–779.

[6] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[7] Y. Fuke and E. Krotkov, "Dead reckoning for a lunar rover on uneven terrain," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 411–416.

[8] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[9] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 229–241, 2001.

[10] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *AAAI/IAAI*, 2002, pp. 593–598.

[11] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

[12] C. Lever, S. Burton, A. Jeewajee, J. O'Keefe, and N. Burgess, "Boundary vector cells in the subiculum of the hippocampal formation," *The journal of neuroscience*, vol. 29, no. 31, pp. 9771–9777, 2009.

[13] E. A. Antonelo, B. Schrauwen, and D. Stroobandt, "Event detection and localization for small mobile robots using reservoir computing," *Neural Networks*, vol. 21, no. 6, pp. 862–871, 2008.

[14] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat," *Brain research*, vol. 34, no. 1, pp. 171–175, 1971.

[15] 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.

[16] K. Blum, L. Abbott *et al.*, "A model of spatial map formation in the hippocampus of the rat," *Neural Computation*, vol. 8, no. 1, pp. 85–93, 1996.

[17] M. J. Milford, G. F. Wyeth, and D. Rasser, "Ratslam: a hippocampal model for simultaneous localization and mapping," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 403–408.

[18] A. Barrera and A. Weitzenfeld, "Biologically-inspired robot spatial cognition based on rat neurophysiological studies," *Autonomous Robots*, vol. 25, no. 1-2, pp. 147–169, 2008.

[19] M. L. Seto, *Marine Robot Autonomy*. Springer, 2012.

[20] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[21] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[22] A. Stentz, "The focussed d$^{\hat{*}}$ algorithm for real-time replanning," in *IJCAI*, vol. 95, 1995, pp. 1652–1659.

[23] P. Field. (2013, October) Potential field algorithm. [Online]. Available: http://taylorwang.wordpress.com/2012/04/06/collision-free-path-planning-using-potential-field-method-for-highly-redundant-manipulators/

[24] E. A. Antonelo and B. Schrauwen, "Supervised learning of internal models for autonomous goal-oriented robot navigation using reservoir computing," in *Robotics and automation (ICRA), 2010 IEEE international conference on*. IEEE, 2010, pp. 2959–2964.

[25] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3671–3678.

[26] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[27] 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, no. 11, pp. 2531–2560, 2002.

[28] K. Doya, "Bifurcations of recurrent neural networks in gradient descent learning," *IEEE Transactions on neural networks*, vol. 1, pp. 75–80, 1993.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[30] M. D. Skowronski and J. G. Harris, "Automatic speech recognition using a predictive echo state network classifier," *Neural networks*, vol. 20, no. 3, pp. 414–423, 2007.

[31] M. Salmen and P. G. Ploger, "Echo state networks used for motor control," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 1953–1958.

[32] L. Larger, M. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso, and I. Fischer, "Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing," *Optics express*, vol. 20, no. 3, pp. 3241–3249, 2012.

[33] I. Ilies, H. Jaeger, O. Kosuchinas, M. Rincon, V. Sakenas, and N. Vaskevicius, "Stepping forward through echoes of the past: forecasting with echo

state networks," *Short report on the winning entry to the NN3 financial forecasting competition, available online at http://www. neural-forecasting-competition. com/downloads/NN3/methods/27-NN3 Herbert Jaeger report. pdf*, 2007.

[34] W. Maass, "Liquid state machines: motivation, theory, and applications," *In Computability in Context: Computation and Logic in the Real World*, pp. 275–296, 2010.

[35] 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, no. 6, pp. 521–528, 2005.

[36] H. Burgsteiner, "Training networks of biological realistic spiking neurons for real-time robot control," in *Proceedings of the 9th international conference on engineering applications of neural networks, Lille, France*, 2005, pp. 129–136.

[37] D. Floreano and C. Mattiussi, *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008.

[38] S. Neurons. (2015, January) Biological neuron. [Online]. Available: http://www.spikingneurons.com/snn_biological-neuron

[39] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[40] P. Dayan and L. F. Abbott, *Theoretical neuroscience*. Cambridge, MA: MIT Press, 2001.

[41] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[42] D. Verstraeten, "Reservoir computing: computation with dynamical systems," Ph.D. dissertation, Ghent University, 2009.

[43] N. Bertschinger and T. Natschläger, "Real-time computation at the edge of chaos in recurrent neural networks," *Neural computation*, vol. 16, no. 7, pp. 1413–1436, 2004.

[44] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2002.

[45] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 936.

[46] B. Widrow, M. E. Hoff *et al.*, "Adaptive switching circuits." 1960.

[47] C. Robotics. (2015, January) Virtual robotics experimental platform. [Online]. Available: http://www.coppeliarobotics.com

[48] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proceedings of the 9th conference on autonomous robot systems and competitions*, vol. 1, no. 1. IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[49] A. L. Jacobs, G. Fridman, R. M. Douglas, N. M. Alam, P. E. Latham, G. T. Prusky, and S. Nirenberg, "Ruling out and ruling in neural codes," *Proceedings of the National Academy of Sciences*, vol. 106, no. 14, pp. 5936–5941, 2009.

[50] 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.

[51] Elinux. (2013, September) Raspberry pi. [Online]. Available: http://elinux.org/RPi_Hardware

[52] Arduino. (2015, January) Arduino mega2560. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardMega2560

[53] Pololu. (2015, January) Reflectance sensor array. [Online]. Available: http://www.pololu.com/docs/pdf/0J12/QTR-8x.pdf

[54] Sharp. (2015, January) Sharp gp2y0a21yk. [Online]. Available: http://www.sharpsma.com/webfm_send/1208

[55] Eigen. (2015, January) Eigen2 library. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page

[56] P. Furgale and T. D. Barfoot, "Visual teach and repeat for long-range rover autonomy," *Journal of Field Robotics*, vol. 27, no. 5, pp. 534–560, 2010.

[57] W. Gerstner and L. Abbott, "Learning navigational maps through potentiation and modulation of hippocampal place cells," *Journal of computational neuroscience*, vol. 4, no. 1, pp. 79–94, 1997.

[58] C. Sölver and G. Marcus, "Dead reckoning and the ocean voyages of the past," *The Mariner's Mirror*, vol. 44, no. 1, pp. 18–34, 1958.

[59] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[60] W. Maass, T. Natschläger, and H. Markram, "On the computational power of circuits of spiking neurons," *J. of Physiology (Paris)*, p. 14, 2005.

[61] B. Cessac, H. Paugam-Moisy, and T. Viéville, "Overview of facts and issues about neural coding by spikes," *Journal of Physiology-Paris*, vol. 104, no. 1, pp. 5–18, 2010.

[62] W. Maass, P. Joshi, and E. D. Sontag, "Computational aspects of feedback in neural circuits," *PLOS Computational Biology*, vol. 3, no. 1, p. e165, 2007.

Appendix A

Additional Files and Schematics

A.1 A* algorithm

A* graph search algorithm

Graph search algorithms are one of the oldest methods of finding a minimum cost path between two positions. They allocate costs to each point in the search space referred to as nodes. The costs could be in terms of Cartesian distance, time travelled or any other criteria chosen. By expanding from the start node through the minimum cost nodes, a path can then be generated from the start to the goal node.

The A* algorithm is based on the following formula.

Cost to present node $f(n) = h(n) + g(n)$.

G(n) is the cost from the start node to the present node. The most common cost used is usually the Euclidean distance between the nodes however other costs such as time taken to reach the node or difficulty of reaching the node are also valid costs that can be used depending on the application.

H(n) is an underestimate of the cost from the present node to the goal node and it is usually referred to as the heuristic. By adding this to the cost to the nodes a more realistic view of the cost of expanding a node is obtained. This prevents a "greedy" approach by the algorithm as the node with the minimum cost is then always evaluated. Pseudo code

for the algorithm is presented in Algorithm 1 below.

```
Result: Minimum cost path
Define start and goal point;
Add start point to OPEN list;
Calculate f(n) for start point;
while no path do
    if OPENlist.isEmpty then
        return failure;
    end
    Choose the node on OPEN list with minimum cost;
    if node==goal then
        return path to node;
    else
        Expand current node;
        Calculate f(n) for successors;
        Add successors not on CLOSED list to OPEN list;
        if (Successor is on CLOSED list)&&(f(n) previous >than f(n) current)
        then
            Add successor to OPEN list;
        end
        Remove node from OPEN list;
        Place node on CLOSED list;
    end
end
```

Algorithm 3: A* algorithm[21]

A.2 Information included in CD

1. Matlab code
2. Arduino code
3. Raspberry Pi Code
4. Videos from Vrep
5. Videos from the Physical Experiment