NTNU
Norwegian University of
Science and Technology

Interfacing Living Neural Cultures to Control a Robot

Neshat Naderi

Master of Science in Computer Science
Submission date: July 2018
Supervisor: Gunnar Tufte, IDI

Norwegian University of Science and Technology
Department of Computer Science

Summary

The idea of benefiting from exceptional capabilities of human cognition has been the inspiration to construct a bio-robotic machine.The human brain is an adaptive complex system capable of performing heavy computations efficiently. Features such as adaptivity and energy efficiency found in biological systems are not present in today's computer technology. The potential of organic systems motivates the NTNU Cyborg project to build a hybrid machine by utilizing unconventional computation paradigms. One of the big challenges is to enable communication between the biological substrate and a digital robotic system.

In this thesis, we take the first step towards interfacing such a hybrid machine that opens the opportunity of integrating biological neural cultures into a mechanical-digital robot. The main goal is proving the functionality of a hypothetical hybrid machine that is constructed using Reservoir Computing methodology. A functioning simulator is implemented as a close imitation of biological neural networks that is able to communicate with a Pepper robot; an off-the-shelf robot. The proposed architecture for this simulator contains three main parts. The first part is a *Pepper* robot that extracts sensor data from its environment, the second is a *Reservoir Computing* (RC) system that utilizes a Random Boolean Network (RBN) to interpret data input stimuli from the robot. The final part contains a controller that can stimulate the robot movements with predictions made by the reservoir system. All three sub-systems are integrated to enable an experimental approach to verify the proposed RC based hybrid cyborg.

The simulator is a functioning system capable of creating dynamics in a digital reservoir, responsive to the external perturbation made by the robot sensory information. Further it can interpret the dynamical changes in the RBN reservoir and pass it as a projection of the original input into a higher dimensional feature space. These new features are used as training data for a linear classification model to predict the robot movements.

Despite the real-world challenges, e.g. noise and non-determinism, the experimental results show that the proposed RC-neuron-robot approach is a sound way toward achieving the goal of the NTNU Cyborg project.

Sammendrag

Ideen om å benytte den eksepsjonelle kognitive evnen til hjernen har motivert oss til å bygge bio-digitale maskiner. Den menneskelig hjernen er et adaptivt komplekst system som er i stand til å utføre tunge beregninger på en effektiv måte. Karakteristiske funksjoner som fleksibilitet og energieffektivitet som finnes kun i biologiske systemer og ikke i dagens teknologi. Potensialet i organiske systemer motiverer NTNU Cyborg-prosjektet til å bygge en hybridmaskin ved bruk av ukonvensjonelle beregningsparadigmer. En av de store utfordringene er å muliggjøre kommunikasjon mellom et biologisk substrat og et digitalt system.

I denne oppgaven tar vi det første skrittet mot å lage et grensesnitt for en slik hybridmaskin som åpner muligheten til å integrere biologiske nevrale kulturer i en mekanisk-digital robot. Hovedmålet er å bevise funksjonaliteten til en hypotetisk hybridmaskin som er konstruert ved hjelp av Reservoar Computing metodikken.

En fungerende simulator er implementert til å være en nær imitasjon av biologiske nevrale nettverk som er i stand til å kommunisere med en programmerbar "off-the-shelf" robot (Pepper). Den foreslåtte arkitekturen for denne simulatoren inneholder tre hoveddeler. Den første delen er en Pepper robot som trekker ut sensor data fra sine omgivelser, den andre er et Reservoir Computing (RC) system som bruker et tilfeldig boolsk nettverk (RBN) for å tolke datainputstimuli fra roboten. Den siste delen inneholder en kontroller som kan stimulere robotbevegelser med predikerte retninger laget av reservoaranlegget. Alle tre delsystemene er integrerte for å muliggjøre en eksperimentell tilnærming for å verifisere en foreslått RC-baserte bio-digitale arkitektur.

Simulatoren er et fungerende system som er i stand til å skape dynamikk i et digitalt reservoar og reagere på eksterne forstyrrelser som blir gjort av robotens sensor data. Videre kan den tolke de dynamiske endringene i RBN-reservoaret og sende det som en projeksjon av den opprinnelige inputdataen til et høyere dimensjonsutfallsrom. Disse nye funksjonene brukes som treningsdata i en lineær klassifiseringsmodell for å forutsi robotbevegelsene. Til tross for reelle utfordringer, f.eks. støy og ikke-determinisme, viser de eksperimentelle resultatene at den foreslåtte RC-nevron-robot tilnærmingen er gjennomførbart og kan være til å oppnå endelige målet til NTNU Cyborg prosjektet.

Preface

I would like to thank my ambitious supervisor, Professor Gunnar Tufte who guided my research with his knowledge. I will also thank him for giving me the opportunity of researching on this field of computer science which has always fascinated me. I am happy to contribute to this project as my final project at NTNU. The last but not the least was the Pepper robot; an interesting teammate to me that made my entire project work both challenging and fun. I would like to thank my Professor again for giving this chance to work with a humanoid robot both as a tool to perform my experiments and also as an opportunity to promote the importance of this field of science to people inside and outside of this community.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Existence being a mysterious and complex concept has caused a countless number of questions in minds through the history of human beings. Reasoning the existence of universe and the purpose of living organisms in the nature involves enormous amount of absurdity and ambiguity, keeping the definition of *life* a complex phenomena and a subset of an unbounded space of uncertainties. Nature is a composite of various living organisms which are in continuous change for adapting themselves to their environment over time. The mechanism of adapting to the environment [4] is known as *evolution* that was introduced by Darwin in the 19th century [5]. Despite the long-lasting process of biological evolution, the nature gradually develops robust, adaptive and fault-tolerant complex systems. In this context, the powerful human cognition distinguishes human brain from these complex systems by being a vastly parallel and dynamic system capable of performing heavy computations with low power consumption.

Despite the fact that ongoing neuroscientific researches have contributed to discover fundamental properties and capabilities of the human brain, underlying mechanism of this dynamical behaviour is still a mystery. But, could we exploit the desirable computational properties of such systems by ignoring the undiscovered principles beneath the complex behaviors? The human brain is a computationally complex biological system that makes its cellular structure to be an exceptional source of inspiration to create efficient and robust computing systems. Building digital systems capable of self-organization and even adaption to new environments or unpredictable external fluctuations requires exploring modern and unconventional computing approaches that can overcome the weaknesses of the traditional Von-Neumann architecture.

Herein, the goal is to establish a proof-of-principle for an unconventional computing approach that can exploit the underlying processes of biological systems, i.e. the neural networks. This thesis is a part of the NTNU Cyborg project [6] that aims to integrate living neural cultures as a control system into a *hybrid* or biological-digital-electric-mechanical system. The NTNU Cyborg project targets unknown problems within computer science, neuroscience and cybernetic. Whereas, in this project we want to take the first step toward interfacing a hybrid machine and utilizing living neural cultures as a controller for a robot.

This thesis project aims to explore computational paradigms that can include living neural networks and determine how such a bio-digital hybrid system can interfere with a physical robot.

Today, the Von-Neumann architecture faces several challenges caused by increasing complexity of systems that grow in size, distribution, parallelism, compute power and amount of processed data, limited performance because of high energy consumption, entanglement between physical (real) world and virtual (digital) world that requires computers aware of their physical environment that can make decisions appropriated to changes and even reconfigure their surroundings in the physical world [7]. This is when *evolution in materio* (EIM) comes into play; a technique that entails manipulating materials directly to perform computations [4]. The objective of evolution in materio is to create computer controlled artificial evolution [4] that exploits the properties of the physical systems. EIM aims to create a hybrid system to exploit physical processes that are unrevealed by humans or overlooked by the designer [8].

Research Objective

This thesis focuses on exploiting the biological neural networks and applying its capabilities to inorganic systems. The human brain is an organic complex system that acts as a highly efficient machine in terms of solving complex tasks, operating in a vastly parallel computing environment that consumes low energy (an estimate of 20 watts) compared to the modern computers. It is self-organized and fault tolerant; at the time of a single neuron failure, the faulty neuron is replaced by the rest of the system to fix the damage with minimal loss and interruption of the ongoing operation.

We claim that exploiting these powerful properties is possible by constructing a hybrid machine composed of the biological neural cultures as a physical reservoir to control a digital-mechanical robot. To prove that such a hypothetical machine can be built, a simulator is required to imitate the behaviors of living neural cultures to communicate with a robot. The aim of simulating a hybrid machine - *a cyborg*- is to reach the ultimate goal of NTNU Cyborg project which is developing a social interactive hybrid bio-robotic machine. Briefly, enabling the communication between living nerve tissues and a robot.

The simulator will consist of a Pepper robot as a digital-mechanical robot and a digital reservoir as the robot controller. To build this simulator, *Reservoir Computing* paradigm is utilized as a substitution to the biological neurons to imitate their intrinsic computational capabilities. Reservoir Computing is a modern approach in the artificial intelligence community to deal with the temporal and sequential data (see 2.4). *Random Boolean Networks* are used to generate a heterogeneous reservoir to constitute a neural network model. Whereas, the robot will execute a predefined task to collect information from its environment. This information will be injected as input data to the Reservoir Computing system to stimulate the reservoir. The input data interpreted by the reservoir and mapped into a higher dimensional feature space and finally passed to a linear machine learning model to predict the robot behavior. Employing this strategy in the simulation will enable us to interface sensors and motors of the robot toward the next step which is integrating the living neural cultures in the loop.

Chapter 2

Background Theory

2.1 Complex Systems

A complex system can be referred to a structure of interacting units displaying a collective behavior. When the collective behavior cannot be deduced by tracing back to each single component [9, p.10] the system exhibit emergence. Emergent properties appear in many systems, e.g. magnetic fields [10, p.5], organic cells, crystallization [11], and even in artificial systems e.g. Cellular Automata [12, p.9]. All of these systems exhibit complicated behaviors that makes order emerge from chaos [13]; a concept emphasized by Strogatz in his book Sync.

Many systems in the nature are non-linear and contain a large number of mutually related elements. Non-linear problems are hard to solve and difficult to analyze compared to linear systems. Linear systems can be broken down into parts and each part can be solved separately and recombined to get the answer [14]. Several systems such as human body, ecosystem and economic system are complex and carry diversity as a common property [15]. Complex systems are non-linear dynamic systems with emergent property. According to Jerome Singer, *"An adaptive complex system involves numerous interacting agents whose aggregate behaviors are to be understood. Such aggregate activity is nonlinear, it cannot simply be derived from summation of individual components behavior"* [16, preface].

Associative memory as a neural network model is a good example of a complex system that is known as *Hopfield* or *attractor* network providing cognitive abilities such as thinking and problem-solving operations. The neural network model consists of correlated simple elements called neurons wherein state correlations of neurons cause emergence in form of local collective behavior across the elements and moreover a global collective behavior that pertains to the entire system. Such a system exhibits rich dynamics in the biological brains [9, p.11]. Briefly, the Hopfield network can operate as an associative memory. It is a fully connected network of threshold elements that represent the neuron firing function. The continuous interaction between elements does the role of synapses to produce a correlation in the firing pattern. Assuming a pre-selected set of patterns, the

interactions can be set to establish self-consistent states of the network. The network in these firing patterns is in a stable state and replacing some of the neurons do not destroy the structure, yet the original pattern is recovered [9, p.11].

2.1.1 Dynamics of Complex Systems

Dynamics is a subject that is used to study the chaotic and complicated behaviors of complex systems. It deals with systems that evolves in time, either they end up in an equilibrium state or does something complicated [14, p.2]. Briefly, we use this term when talking about complex systems and their behaviors. When a deterministic system does not enter a periodic trajectory, the dynamic of this system is defined to be chaotic. Such systems are dependent on the initial conditions [17] and usually unpredictable. Herein the target is to exploit the properties of a dynamical system with dynamic structure that is adaptable to external and internal fluctuations e.g. human brain and its neural networks.

2.2 Neural Network Cultures

Even though the mystery beneath the remarkable functionality of the biological brains is still undiscovered, recent technology development has opened possibilities to study the living neurons-a complex dynamical system. *Neural network cultures* are referred to cell culture of neurons in vitro [1] that serve a simpler model of a living brain. The cultured

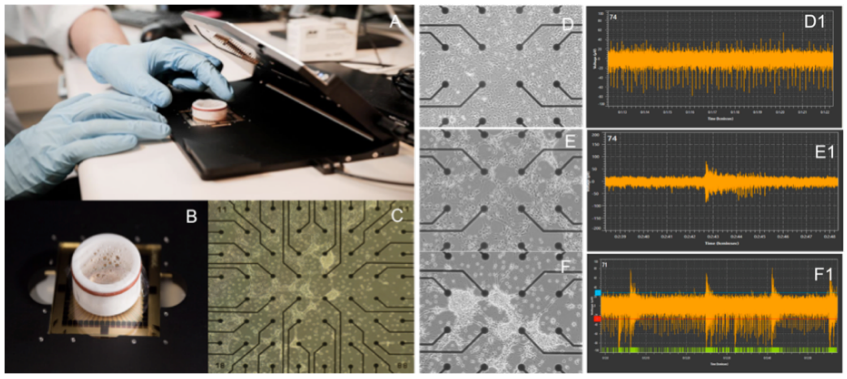


Figure 2.1: MEA overview - (A, B) Neuron cultures on MEA. (C) Microscopic image of neuron cultures. (D to F) Showing the developing neural network structures. (D1 to F1) Shows the neuron activity analysis of for figures D to F.

neural networks are used to study the intrinsic properties of neural network structures and reveal the underlying principles of human cognition; learning, memory, information processing and plasticity. Multi-electrode Array (MEA) 2.1A is an attractive hardware that enables recording cellular activities along with delivering electrical stimuli [18]. This implies a tool capable of measuring the dynamic and complexity of neuron cultures. Biological neural networks embodied in an organic can exploit sensory information and shape

[1] *In vitro*: techniques to perform a procedure in a controlled environment outside of a living organism

their structures [6]. This embodiment can be emulated by connecting neuron cultures to MEAs that embodies the dissociated neurons in a robotic body 2.1B. Figure 2.1C is a microscopic image of neuron cultures on MEA. Figure 2.1D, 2.1E and 2.1F illustrates development progress in neural structures from initial state to the final state of the network. The electrodes can be used to send electrical signals as artificial sensory inputs to stimulates the neural cultures. Then the spiking neuron activities can be measured by the MEA electrodes and analyzed. A visualization of such analysis is available at Figure 2.1D1, 2.1E1 and 2.1F1 showing electrical activities during three different phases of the neural cultures.

2.3 Neural network representation

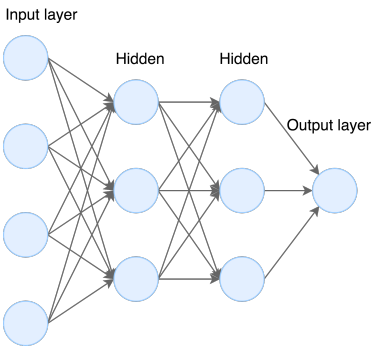
*Artificial Neural Network*s (ANNs) are a collection of interconnected units to simulate specific cognitive functions and are mostly used to perform machine learning tasks. ANNs are inspired by human neural activity consisting primarily of electrochemical activity in a network of brain cells called *neurons* 2.2c. A simple mathematical model was devised by McCulloch and Pitts (1943) [19] mimicking neurons. It *fires* when a linear combination of its inputs exceeds a *threshold*.

Different ways of linking ANN units divide neural networks into feed-forward networks 2.2a and recurrent neural networks (RNNs) 2.2b. The former has connections only in one direction forming a directed acyclic graph, where the latter feeds its outputs back into its own input, building a dynamical system that may reach a stable state or exhibit oscillations or chaotic behaviours. RNNs can be thought as networks with memories in contrast to feed-forward strategy with no loops. Parameter sharing across different parts of the model could make recurrent neural networks advantageous over traditional feed-forward architectures. This property makes it possible to construct models for processing temporal and sequential data, and also generalize to sequence lengths not seen during training or share statistical data across several time steps and sequence lengths [20, p.367]. Sequential data can be processed by RNNs when the data is represented as a sequence of values $x_1, x_2, \cdots, x_\tau$ with a time step $t \in \{1, \tau\}$ [20, p.368]. Figure 2.2c is an image of the NTNU Cyborg living neural networks a clear illustration of a non-feedforward architecture.

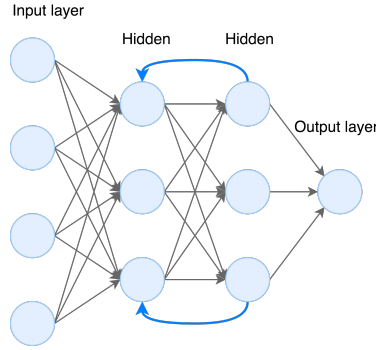
2.4 Reservoir Computing

Reservoir Computing (RC) has its roots in neural network research. It was discovered that the rich dynamics and memory capabilities of RNNs could be exploited without any training of the network. Meaning that it is possible to obtain high performance with a randomly generated recurrent network coupled with a linear readout layer where the model is trained only on the activation of the RNN nodes [21].

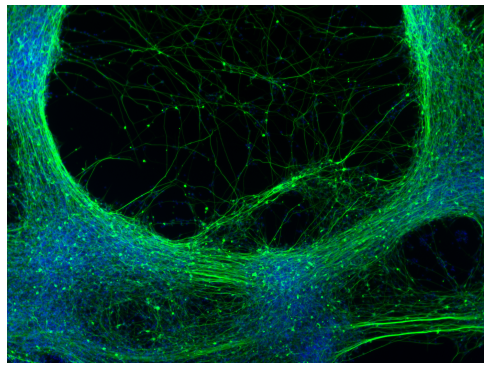
Theoretically, the RNNs are very powerful tools for solving complex temporal machine learning tasks, keeping in mind that they can store and distribute memory. Nevertheless, the hassle of slow convergence and the high time complexity of training the process makes the application of RNNs not always feasible for real world problems [22]. By Reservoir



(a) Feed-forward neural network



(b) Recurrent neural network



(c) Living neuron clusters

Figure 2.2: (a) A feed-forward artificial neural network representation with 2 hidden layers. (b) A recurrent artificial neural network representation with 2 hidden layers and two recurrent links.(c) An image of living neuron clusters used by NTNU Cyborg project. *Photo by Vibeke Devold, INB NTNU.*

Computing there is no need for training the internal weights in the recurrent network, only the readout layer. RC methods are suited for solving temporal and complex tasks because they exploits the advantages of RNNs like storing memory, but they eliminate the challenges with training these networks.

In the recent years, a new approach to RNNs was established by *Echo State Networks* (ESNs) [23] and *Liquid State Machines* (LSMs) [24] from machine learning and computational neuroscience fields. These terms are both known as Reservoir Computing methods. These trends stem from the observation that as long as an RNN possesses certain generic properties, supervised adaptation of all interconnection weights is not necessary, and only training a memoryless supervised readout from the RNN is enough to obtain excellent performance in many tasks (Figure 2.3).

In this context, a randomly created recurrent network is viewed as a dynamic system called a *reservoir* [25]. A reservoir behaves as a complex nonlinear dynamic map that projects the input stream into a higher-dimensional feature space [22]. Figure 2.3 illus-

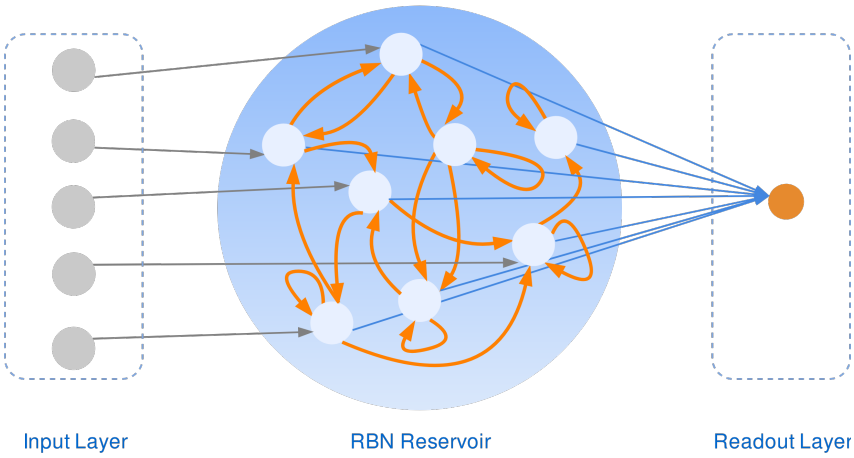


Figure 2.3: Decomposition of Reservoir Computing system. Illustrating the 3 layers, input layer, RBN reservoir and readout layer from left to right.

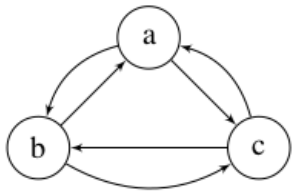
trates a reservoir computing device composed of an input layer, a randomly generated reservoir, and a readout layer. Given a task, the input layer passes input signals to excite the nonlinear dynamics of the reservoir [26]. The dynamic behaviour of the reservoir is used to process input data before feeding that into the readout layer. Subsequently, the input signal traces are interpreted by the readouts. Computing the expected outputs for the current task is harnessed by training on reservoir states using linear methods, thereby eliminating the high computational costs of training RNNs.

Input Layer

In a Reservoir Computing system, the *input layer* has the task of passing input signals into the reservoir. The input signal works as a stimuli to *perturb* the dynamical behaviour of the reservoir system. From other point of view, the input data is processed by the reservoir dynamics and is projected to a feature space with higher dimensions compared to the original input signal.

Reservoir

The *reservoir* in this project is a *Random Boolean Network* (RBN) acting as a nonlinear dynamic map to project an input stream into a more complex feature set. RBNs are a generalization of *Cellular Automata* (CA) where the CA is capable of producing complex dynamics from very simple rules. CA is a network of simple processors with no central control unit having limited communication among its processors [12, p.5]. The level of complexity in decentralized and massively parallel systems such as biological systems was mentioned in section 2.1. Thereby, a computational model is required to study unknown properties of biological systems. Stuart Kauffman developed the Random Boolean Networks (RBNs) as a model of genetic regulatory networks, claiming that living organisms



(a) RBN topology

Ancestor states		New state
0	0	0
0	1	1
1	0	0
1	1	1

(b) Transition table for node a

Figure 2.4: (a) A simple RBN containing 3 nodes, each having one incoming and one outgoing connections to other nodes. (b) RBNs state transition table for node *a*[1].

could be represented with random elements [27]. To simulate the living neurons we take the advantage of generic characteristic of RBNs to attempt revealing living neurons mechanism. The generic characteristic of RBNs come from their random generations. It omits the need of programming every single element in the neural network precisely, so that assuming a particular functionality or connectivity of the nodes would be unnecessary.

Readout Layer

Readout layer receives output of the dynamical reservoir to interpret traces of the input signals. At this part of RC system, a linear model using supervised strategy is trained to predict the expected output of the current input.

2.4.1 RBN reservoirs

A Random Boolean Network or RBN is a collection of N randomly interconnected nodes, each of which having K input connection links. Figure 2.4a illustrates a RBN with $N = 3$ and $K = 2$. At any instant in time t, the node assumes a Boolean value s as its state, where $s \in \{0, 1\}$. This structure provides a 2^N domain of possible functions for each node. The state of a node at time $t + 1$ is determined by the logic functions of its K input nodes. In a classic RBN model, updating the nodes are done synchronously meaning that calculation of node states is done sequentially for all nodes. The state transition table for the RBN node *a* can be seen at Figure 2.4b. Networks with same K connectivity for all nodes are called homogeneous RBNs. If K is not same for all nodes the network is to be heterogeneous and uses a parameter called mean connectivity $\langle K \rangle$. RBNs have finite state space (2^N) that means identical states might appear in dynamic flow of a network. In a deterministic system, the network reaches repeated states that is called an *attractor*. An attractor consisting of only ones state is known as *point* attractor, whereas those consisting of two or more are called *cycle* attractors [27]

Threshold Logic Gates

The type of activation functions of each RBN node is selected from a set of *threshold logic gates* (TLGs) and *XOR logic gates* dominated by TLGs. Threshold logic gates (TLGs) are

very similar to spiking neurons. A Boolean function $f(x_1, x_2, \cdots, x_n)$ is a threshold function if there exist n weights $(w_1, w_2, \cdots, w_n)$ and threshold T such that

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & \text{if} \sum_{i=i}^{n} w_i x_i \geq T \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

The activation $\sum_{i=i}^{n} w_i x_i$ is the weighted sum of inputs. Based on the input pattern, the neuron is activated/fired when the value of activation is equal to or above the value of threshold T [28]. The aim of using threshold logic units is to implement a neural network model close to biological networks and exploit the electrical functionality of the living neuron cells.

2.4.2 Training at Reservoir Computing

In a *supervised learning* setup, a *training set* of input values (x_i) labeled by their expected values (y_i) is given, where the function $f(x)$ generating the y_i is unknown. The task of this supervised learning method is to approximate a function h for the true function f. In this context, learning is a search through the possible hypotheses that performs well on unseen data. Therefore, a *test set* different from the training set is required to measure the accuracy of the learning algorithm. The algorithm generalizes well if it predicts the value of y correctly tested for novel examples.

Training of recurrent networks is omitted in the reservoir computing method which makes it attractive in contrast with other learning strategies. In case of reservoir computing, the output from a randomly generated recurrent network (the reservoir) could be used as training data for a linear model.

Ridge regression has performed well in earlier RC research projects and therefore used in this project. Ridge regression [29] is a linear model used in the readout layer of the RC-system. This model will predict the direction of Pepper robot in its environment and is to verify the computational capabilities of the reservoir dynamics.

2.5 Reservoir Dynamics and Computational Capabilities

When choosing a RBN as a digital reservoir in the RC system, the target is to develop a dynamical system and achieve the computational capabilities similar to the living neural networks. We usually search for dynamics containing longer attractor length and larger number of distinct attractors. The *dynamics* of a RBN is calculated as discrete time series of the network state. So a RBN with a random initial state is updated over a time interval, we can observe how the network state stabilizes or changes. These changes in the network dynamic are classified as *ordered, critical* or *chaotic* [27] and are illustrated at Figure 2.5. In the ordered phase 2.5a, the dynamics stabilizes quickly where in the chaotic phase 2.5c, the states are mostly in change and does not follow any patterns. The phase transition from ordered to chaotic regime is called the *edge of chaos* or critical phase where some cycles of previous states are appeared in the flow and are seen as visible patterns in the state-space diagram shown in Figure 2.5b.

(a) Ordered phase, K=1. **(b)** Critical phase, K=2. **(c)** Chaotic phase, K=3.

Figure 2.5: State-space diagram showing RBN dynamic flow in three possible phases. The RBN illustrated above has a size of 64 nodes and logic functions for each node is selected with uniform probability from a set $F \in \{t \geq 1, t \geq 2, t \geq 3, XOR\}$. The dynamic grows downwards in y-axis and in the x-axis, the state of the of the network in each time step is printed as a black dot for zero state and white for one state. Plots from experiments in pre-thesis project [2]

Separation and *fading memory* or echo state property can be measured to study computational capabilities of a reservoir system. In order to achieve a good performing RBN reservoir, it should be able to forget states (*fading memory*) and separate between the input streams (*separation*). According to [30] computational capabilities of RBN reservoirs are most significant when transiting into critical dynamics which is achieved when the difference between separation and fading memory is largest. So the computational capability of a reservoir $\mathcal{M}$ over an input stream of length $\mathcal{L}$ and a τ time steps in the past is defined as,

$$\Delta(\mathcal{M}, \mathcal{L}, \tau) = Seperation(\mathcal{M}, \mathcal{L}) - FadingMemory(\mathcal{M}, \mathcal{L}).$$

2.6 Physical Reservoirs as Computing Units

Physical systems such as neural networks in human brain extent *Intrinsic Plasticity* (IP) [26]. Such systems are able to utilize a given dynamic regime despite the network parameters. IP denotes the capability of biological neurons to change their excitability according to stimulus distribution that a neuron is exposed to. J. Steil [26] used this property for the first time in RC and proposed a biologically motivated learning rule based on neural intrinsic plasticity to optimize reservoirs of analogue neurons.

To obtain neurons' desired properties such as self-organization, vast parallelism, robustness and adaptivity without understanding their hitherto unknown underlying complex behaviours, physical mediums like living neuron cultures 2.6 can be used as reservoirs. This is because some physical mediums provide very rich and high dimensional dynamical complex systems. In some previous research projects, a real bucket of water [31], Driven Chua's circuit [21] and Carbon Nanotubes [32] were used as other examples of physical reservoirs and succeeded to perform pattern recognition and classification tasks. Hence, transforming certain substrates into trainable computational reservoirs [32].

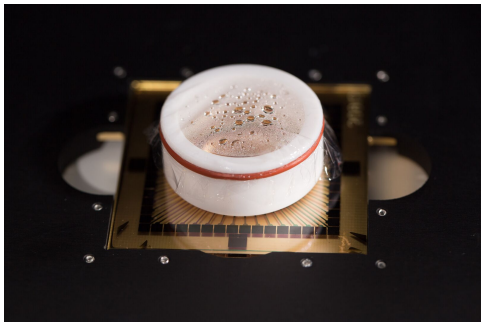


Figure 2.6: *In-vitro* neuron cultures

2.7 A Hybrid Bio-Digital Machine

A *Cyborg* or a cybernetic organism is a combination of machine and living tissue. In 1960, the term Cyborg was introduced by M. E. Clynes and N. S. Kline [33] as a *man-machine* that consciously can integrate external components to extend its self-regulation in order to adapt to new environment. Kevin Warwik [34] defines a Cyborg, *a cybernetic organism, part human part machine* [35]. Moreover he applied experiments on his own body and planted a chip in his arm for the first time in 1988.

The NTNU Cyborg project is a research project within the Norwegian University of Science and Technology (NTNU) to develop a social and interactive hybrid bio-robotic machine. The aim of the NTNU Cyborg project is to enable the communication between living nerve tissue and a robot [36]. This thesis project is a sub-project of NTNU Cyborg and its main target is developing an interface to interpret data from biological neuron cultures to control a robot. Chapter 3 gives a deeper description of the Cyborg project and its platform.

Chapter 3

Cyborg Project

The development of the Cyborg project is inspired by Reservoir Computing; a method to transform input data into a feature space through the dynamics of a network, and also *Evolution-In-Materio* (EIM) [4]; manipulation of a physical system by computer controlled evolution to obtain useful properties. Such properties may be unknown or poorly understood from the underlying physics, i.e. the system may be model-free [4] . Currently, the NTNU Cyborg system is composed of in-vitro neuron cultures embodied in a Micro-Electrode Array 2.6, an artificial neural network interface and a robot. The three main components are illustrated at Figure 3.1. The idea behind building a Cyborg is to construct a hybrid computer and to create a transition from biology into a mechanical system. In connection with chapter 3, the Cyborg project aims at demonstrating the system by applying wall avoidance tasks. Hence, the first attempt is integrating biological neural networks into a robot system [37].

3.1 Cyborg Platform

The initial structure of the Cyborg platform is sketched in Figure 3.1. It is simply an interface between neurons and a computer. It consists of three parts, each separated by an electrical interface; a biological network, an Artificial Neural Network (ANN) readout, and a robot control subsystem [37].

The biological network is neural cultures grown on a *Micro Electrode Array* (MEA) 2.1B to interface neurons using an electrical stimuli 2.6. An image of neurons structure was captured in microscopic size 2.1C coupled with spiking neurons activities 2.1D1-F1. More details about the cultured neurons are available in chapter 2 coupled by Figure 2.1. These neural cultures are used as physical reservoirs fed by sensory information from a robot [3]. The ANN layer subsystem is a simple feed-forward ANN, together with a processing unit to handle feedback information for biological neural networks. This digital only system is used as a readout layer in the RC system to perform learning and classification tasks. The robot control part in the primary platform is a simple robot made at NTNU that consists of several sensors and two actuators; *Move Right* (ML) and *Move*

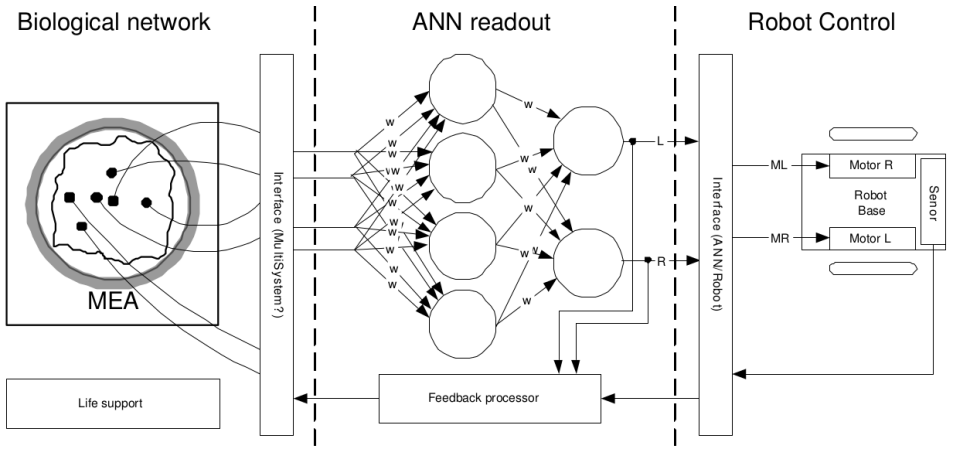


Figure 3.1: NTNU Cyborg platform.

Left (ML) to control the wheels. It can perform simple tasks such as moving and rotating in 2D and sensing distance to walls [37].

3.2 Replacing the Robot Prototype with a Pepper Robot

The robot prototype built earlier by the the NTNU Cyborg team is shown in Figure 3.2. Some drawbacks followed by building a robot control system from scratch. Developing a fully functioning robot system required many working hours of programming and integrating new devices and features. Additional costs were continuously incurred by purchasing new sensors and devices. These challenges led to replacing the robot prototype with a humanoid robot. It was primarily to improve the research process and the testing environment, reduce expenses and time consumption on robot development. As a result, focusing resources on the major motivation of the Cyborg project that is the integration of biological neural cultures into a robot.

A Pepper robot 3.3 was purchased to overcome challenges related to construction of a complete robot system from scratch. These challenges are assumed to be decreased significantly by replacing the old robot prototype with a Pepper. By turning focus away from robot development and towards the application of new ideas and techniques, it is hoped to speed up the progress in the Cyborg project towards its ultimate goals.

3.3 Pepper

Pepper is a humanoid robot built by Aldebaran and SoftBank Robotics to be a human companion. Figure 3.3 shows the appearance of Pepper having a humanlike shape. Its greatest ability is the recognition of human emotions and adapting behaviors based on mood of the interlocutor. Natural behaviour, interactivity and autonomy are major features of the Pepper. These features create a natural and intuitive communication through body

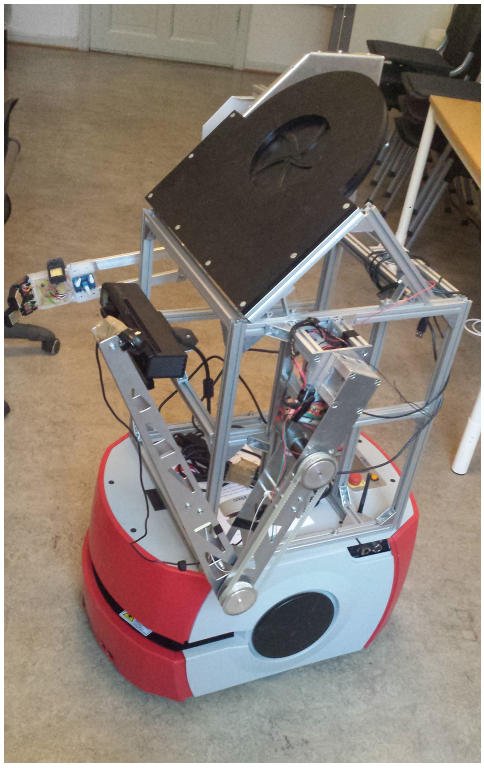


Figure 3.2: NTNU Cyborg robot prototype from previous project phase [3]. This prototype is now replace by a Pepper robot illustrated at Figure 3.3

movements, voice and speech. It can perform tasks such as emotional displays, voice and face recognition, learning, autonomous movements, speaking and hearing. It can also be programmed and personalized and used as a welcoming, informing and entertaining tool [38]. The reason for selecting a Pepper specifically, was because of its existing SW (low-level and programming framework), range of functionality, availability and price.

3.3.1 Robot Design

Pepper is managed by an embedded operating system called NAOqi OS. It is a Gentoo based GNU/Linux distribution, developed by the SoftBank Robotics for interacting with the robot hardware. NAOqi OS aims to create new applications and external programs to customize and pilot the robot. NAOqi features Software Developments Kits (SDKs) for Aldebaran robots and support Python, C++, Java, JavaScript and ROS. These SDKs allow developers to use APIs and run applications remotely on the robot [39]. The interactive and emotional interpretation is achieved by 4 directional microphones on robot's head. These microphones detect the sounds to locate their source and to identify emotions through the tone of interlocutor's voice. Movement identification and face recognition is enabled by

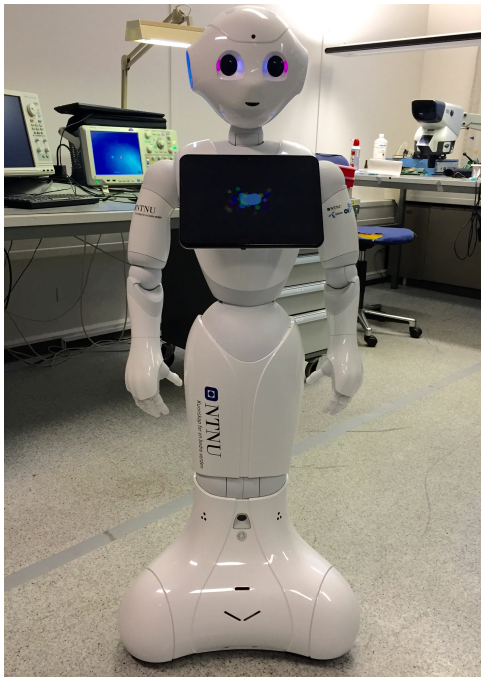


Figure 3.3: The Pepper robot, a replacement for NTNU Cyborg's previous robot prototype.

one 3D and two HD cameras. Pepper is equipped with several sensors to perceive its environment and reduce the risk of falling and collisions in unknown environments. The fall manager and anti-collision system maintain the robot's balance and control. Three multi-directional wheels enable robot movement, and movement is controlled by 20 motors in the head, arms and back. A high capacity lithium-ion battery provides approximately 12 hours of autonomy. Pepper can also be connected to the internet via wireless or wired connection [40].

3.3.2 Pepper as robot control sub-system

Capabilities and functionality of the Pepper robot were studied using the documentation and the software provided by Aldebaran Robotics in the early stages of this thesis project. Later, some simple tasks such as movement to a certain point with a certain velocity using *Choregraphe Suite* [41] software or connecting to Pepper via WIFI using the Python SDK were undertaken. Choregraphe Suite is a desktop application for creating applications such as dialogues, animations and behaviours. This software also has a virtual environment to test applications on a simulated robot or monitor its movements [38]. Python SDK is a NAOqi SDK and is used to implement applications with advanced functionalities and communicate with sensors. These applications can be run on the robot from remote machines using NAOqi APIs.

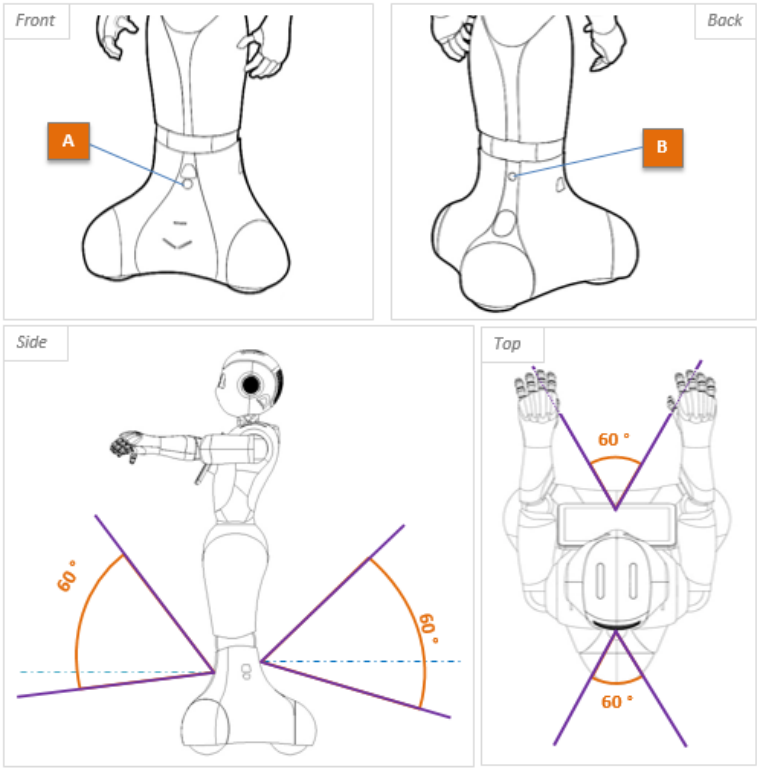


Figure 3.4: Sonar sensors detection range for Sonar A in the front side and B in the back. Effective cone of the Sonars are illustrated from side and top by $60°$.

Technical overview

Pepper has $1210mm$ height, $480mm$ and $425mm$ depth. Its motherboard has Atom E3845 processor, quad core CPU, 1.91 GHz clock speed and 4 GB DDR3 RAM. Ultrasonic (Sonar) sensor specifications are gathered in Table 3.1. Pepper is equipped with two Sonar sensors 3.4 one in the front (A) and the other in the back (B). These sensors measure the distance to the objects in the environment. Distances closer than $0.3m$ are ranged as $0.3m$. Figure 3.4 illustrates the location of the A and B sensors as well as the effective cone of $60°$. A brief summery of Sonar specifications is gathered in Table 3.1. The leg motors ensuring robot's movement in 2D are 3 wheels located front right (WheelFR), front left (WheelFL) and back (WheelB) [42]. Detailed specifications of these motors are shown in Table 3.2.

Pepper application

A program written in one of the supported languages for NAOqi OS is required to enable application execution on the robot. In this project, a program using NAOqi 2.5.5 Python SDK is implemented to communicate with robot OS. This program starts by instantiating

Sonar Specifications	
Frequency	$42kHz$
Sensitivity	$-86dB$
Resolution	$0.03m$
Detection range	$0.3m - 5m$
Effective cone	$60°$

Table 3.1: Sonar sensor specifications.

Wheel Motor Specifications	
Family	BLDC (Brushless DC)
Model	EC45_70W
No load speed (rpm)	6110
Torque constant (mNm/A)	36.9
Stall torque (mNm)	1460
Max Continuous torque (mNm)	128
Speed Reduction ratio - Type B	25

Table 3.2: Motor specifications of motors used in Pepper leg wheels.

a `qi.Session` to get API services such as memory and motion. To run applications from a remote computer, it needs to connect to NAOqi at Pepper IP on port 9559. This connection is established through a TCP handshake.

3.3.3 Sonar Sensing

Sonar or ultrasonic sensors act as transmitters that convert the acoustic energy into electrical energy [43]. They exploit propagation delay of high frequency acoustic waves like $42kHz$ used in Pepper Sonar 3.1 and their echos to extract information from their surroundings [44, p.492]. A Sonar sensor operates by releasing acoustic pulses to the environment and wait for receiving the echo from pulses that were sent out. The travel time of echo waves decide to range the objects in the environment [44, p.493]. This property of Sonars can be used for obstacle avoidance by measuring the distance to closest object in a robot environment.

This sensor seemed to be a simple sensor compared to other sensors installed on the Pepper and is selected to extract information from robot environment in this thesis project. Sonar sensors are also popular in robotics community to measure object ranges. In addition, the output of this sensor is decimal values which requires low computational effort to employ it as input data in other part of proposed system.

3.4 A Recap of NTNU Cyborg Status and the Related Thesis Project

The NTNU Cyborg project aims to construct a hybrid machine. At this stage, NTNU cyborg can grow neuron cultures and connect to MEAs. Through the MEAs, the cultured neural networks can be stimulated with electrical signals. The neural response can be measured as the spiking neuron activities and analyzed. Currently, a functional framework for a bio-digital communication between the living neural networks and a Pepper robot is close to be achieved. This thesis project aims to reach this target by simulating such a bio-digital hybrid machine. This research utilizes the Reservoir Computing approach together with a Pepper robot to prove the viability of such a system. The way of approaching this goal is to communicate with robot and its sensory and motor functions towards performing a navigation task. The sensors in the robot are used to precept the environment and extract data from it. This data will be mapped to utilize in the RC system to function as a controller to the Pepper robot. Briefly, the goal is to construct a simulator and interface the living neural networks to control the Pepper robot.

Chapter 4

Simulator Infrastructure

A simulator is built to provide an imitation of a bio-digital hybrid machine which can contain a physical reservoir. This simulator provides a model to represent the characteristics of the biological neural cultures. It can be utilized as a testing platform to perform experiments on the Pepper robot and the Reservoir Computing system. The simulator can examine the capabilities of physical reservoirs and the possibility of controlling a robot. The simulator architecture includes the implementation of a RBN RC system, software application for low-level motor control and sensor readout from the robot. The integration of robot software and the RC system in a common platform is provided by the simulator implementation.

4.1 Simulator Architecture

Architecture design for the simulator is illustrated at Figure 4.1. It constitutes a tripartite structure compounded of three connected primary components. The links between each component at Figure 4.1 represents the flow of data transmission from one to another component. The simulation starts with a navigation task at the Pepper robot platform to produce data from the sonar sensor (4.2). This data is fed to the reservoir to stimulate the RBN. During the stimulation process, the reservoir accumulates the state changes in the RBN. The readout layer utilizes the state accumulation as training data to predict navigation directions for the robot (4.3). At the final stage, the controller determines the movement commands corresponding to the predictions made at RC level. These commands will perform a controlled move on the robot (4.4). To summarize, the system enables communication and data transfer between each component. The general purpose of this system is to apply a learning process by Reservoir Computing approach to control a robot. The detailed structure of each component is explained in the following sections.

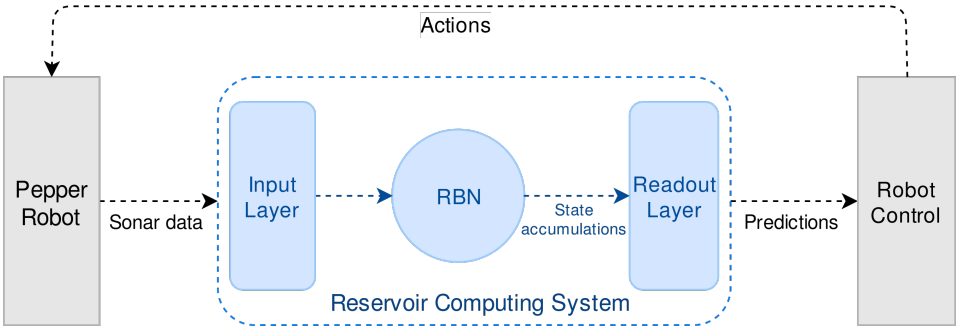


Figure 4.1: Simulator architecture showing the system's main components. Pepper robot, Reservoir Computing system and the robot controller; from left to right.

4.2 Pepper Robot

At the first step of the simulation, the pepper robot is given a simple navigation task to accomplish. The purpose of this task is collecting distance measurements to the closest objects in the robot environment. These measurements are produced by a Sonar sensor during path exploration and obstacle detection. The sensor output records are transmitted to the Reservoir Computing system for further interpretations.

Task

The robot is given a range of target coordinates in a 2D space. The task is finding a path towards the targets and apply a successful movement in an unknown environment. In addition to record Sonar sensor data during the navigation, the robot must detect and handle the blocking objects on the path. At the time of obstacle occurrence, the robot must handle the obstacle and recalculate a new path. A similar environment shown at Figure 4.2 is prepared to determine the task success or failure.

4.2.1 Navigation algorithm

A global coordinate system illustrated at Figure 4.3a is defined for performing movements in a *xy*-space. This global coordinate system ensures that the robot moves according to the global origin. By defining the origin as the watching point of view the problem of calculating relative coordinates for the navigation is solved. The navigation algorithm simply calculates the angle to the next target position and the robot rotates accordingly. Then, it calculates the distance to the next position and moves in x-direction.

4.2.2 Obstacle detection and avoidance

An obstacle is detected by Sonar sensors located in front or back side of the robot. The Sonar sensors calculate the distance to the closest object in the sensor range. In order to prevent colliding with external objects, all moving actions terminates immediately when

Figure 4.2: The area prepared for Pepper to perform a navigation task with obstacle avoidance.

the obstacle is located too close to the robot body. At this point, the robot rotates θ degrees and moves dx meters forward. After obstacle handling action, it recalculates a new path to reach the final destination.

4.2.3 Robot application

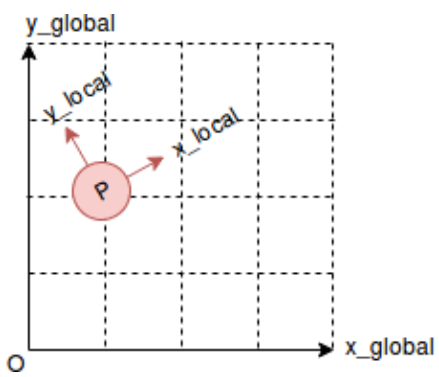
A navigation program is implemented to move Pepper towards target coordinates in a 2D space. The application is subscribed to memory events to collect sensory information from the unknown environment. It executes an asynchronous move function and registers a distance measurement from Sonar sensor every dt milliseconds. Each distance value is assigned with the action taken by the robot at that time step. This sensory information is used to ensure safe navigation in the room and prevent Pepper from colliding to obstacles in the navigation path.

4.2.4 Data Pre-processing

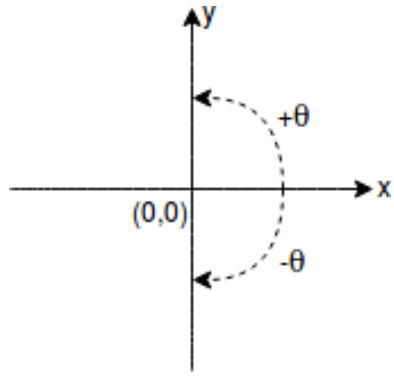
The data recorded by the Sonar sensor is a variety of decimal values. These values cannot be transferred as input to the RC system unless it is converted to readable values. Therefore, a mapping from decimal values into bits of 1 or 0 values is necessary. To fine-tune the data and keep the significance of the values, the data is processed through four different steps illustrated at Figure 4.5; 1. Noise reduction, 2. Normalizing, 3. Scaling, 4. Rounding and 5. Encoding. The preprocessing operations utilizes Python libraries `scikit-learn` - a machine learning library and `Numpy` - a scientific computing package.

Noise reduction

Since the raw data might contain noise such at action assignment to certain distance values, an attempt is done to remove the probable noise. So at the first step the raw data is searched



(a) Coordinate system, 2D space



(b) Rotation direction

Figure 4.3: (a) Global coordinate system defined for navigation. The points in this coordinate system have coordinates from a watching point of view located at origin $(0, 0)$. (b) The rotation around $z-$axis of the robot. Rotating in $+\theta$ is defined counterclockwise and $-\theta$ in opposite direction.

for faulty label assignments See subsection 5.2.2 for labeling of dataset. At each searching sequence, if any noise is detected the value is assigned with a correct label.

Normalizing

The second pre-processing step normalizing the distance values. The intention is to adjust the values measured in different navigation areas to a common scale. The normalized dataset is in a range of $[0, 1]$.

Scaling

To differentiate the values easier and make the differences more significant, all the values are scaled up by a value which is calculated based on the RBN size.

Rounding

At this pre-processing step a simple rounding operation is applied to all values to determine their nearest integer values.

Encoding

At final step, the integer values are mapped into arrays of binary bits. Note that it is not a conversion from decimal to binary, but an encoding from integer values into ones and zeros. The length of the bits array is l and is decided by N - the size of RBN where $l \geq 0.8 * N$. The purpose is to perturb the RBN by at least 80% of its nodes. The mapping between the integer values and the binary bits is exemplified below to give a better understanding of the idea.

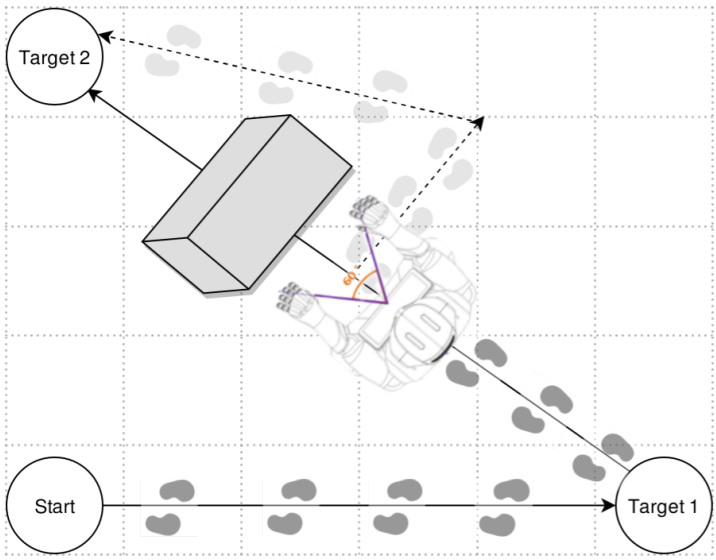


Figure 4.4: An example of obstacle handling in the navigation path by Pepper.

4.3 Reservoir Computing System

The Reservoir Computing (RC) component is where the data from robot is interpreted and passed back to the robot. The RC system forms a 3-layered constitution illustrated at the sub-figure 4.6. It has an input layer, a reservoir and a readout layer. The input layer streams data from the Pepper robot and feeds it into the reservoir. The reservoir being a RBN - a randomly generated Boolean RNN (2.4) returns accumulation of the state changes in the reservoir at each perturbation session. As the final step, the output from the reservoir is interpreted at readout layer. The output from this layer is used to stimulate the robot movements.

4.3.1 RC setup

The RC model utilizes a heterogeneous RBN as its reservoir where the K connectivity and the activation functions are chosen randomly with uniform probability for each node. For every node a TLG or Threshold Logic Gate (2.4.1) is selected being the node's activation function. The activation function decides the nodes' output in the RBN reservoir. The TLGs are assumed to constitute a better approach to biological neuron behaviours compared to the classical Boolean logic gates. The reason behind this assumption is that firing potentials of synapses in neurons can be determined by thresholds [27]. The nodes have connection of K size where $K \in \{1, 2, 3\}$ to other nodes in the reservoir. In addition to internal connectivity between the nodes, some of the nodes are selected by uniform probability to be perturbing nodes and have inwards connections from the input layer. In addition, all of the nodes have outwards connections to the output layer for monitoring

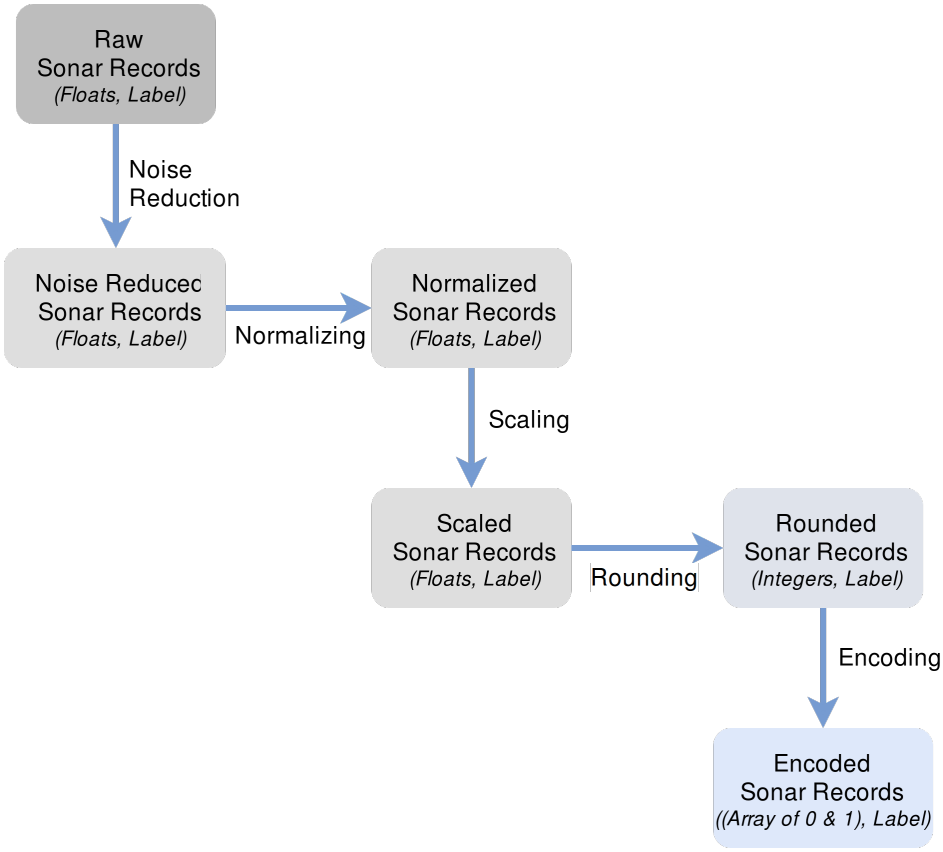


Figure 4.5: Steps of the pre-processing of Sonar data produced by Pepper robot.

Integer		Binary bits
0	$\longrightarrow$	$\{ 0, 0, 0, 0, 0, 0, 0, 0 \}$
1	$\longrightarrow$	$\{ 1, 0, 0, 0, 0, 0, 0, 0 \}$
2	$\longrightarrow$	$\{ 1, 1, 0, 0, 0, 0, 0, 0 \}$
3	$\longrightarrow$	$\{ 1, 1, 1, 0, 0, 0, 0, 0 \}$
4	$\longrightarrow$	$\{ 1, 1, 1, 1, 0, 0, 0, 0 \}$
5	$\longrightarrow$	$\{ 1, 1, 1, 1, 1, 0, 0, 0 \}$

Table 4.1: An encoding example showing the mapping between integers and binary bits arrays where $N = 10$ and $l = 8$. In the left side, it is given 6 integers from 0 to 5, each of which are encoded into a binary array of zeros and ones. The amount of 1-bits depends on the size of integer value. No 1-bits for encoding of the 0 integer, but five 1-bits for encoding of the 5 integer.

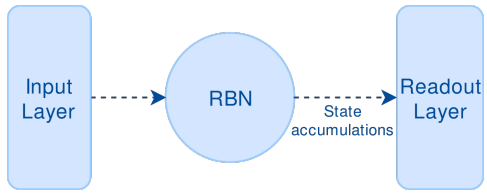


Figure 4.6: Reservoir Computing system components

reservoir's state changes. These connection are illustrated at Figure 2.3

The input layer receives the processed data from the robot and injects every record to the reservoir. Perturbation process is visualized at Figure 4.7 where every data record unit contains a range of Boolean bits and each bit is injected into a pre-selected node in the reservoir. Input bits having value 1 changes the state of the node. Table 4.2 shows the output of each node for all possible conditions. At each iteration of the RC execution, the state change accumulations for each node is increased by 1 if the current state of a node differs from its previous state. At the end of perturbation period the state change accumulations are delivered to the output layer. Figure 4.8 illustrates how data is extracted from readout layer. These accumulations are registered as training data for Ridge classifier. In the training data set, each feature is the change of state per node in the network and the label for each feature set is the robot action at time t. Each move action was labeled with a L_i where L is a set of labels $L = \{$*"forward", "obstacle", "stop", "turn_left", "turn_right"*$\}$.

Table 4.2: Node state transition accumulation when perturbed by input bits.

Input	Current State	Output	State Changes
0	0	0	unchanged
0	1	1	unchanged
1	0	1	increase by 1
1	1	0	increase by 1

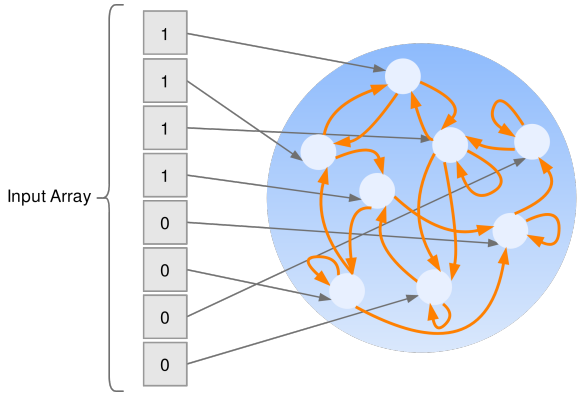


Figure 4.7: Encoded Sonar values are fed to the RBN reservoir as external perturbation.

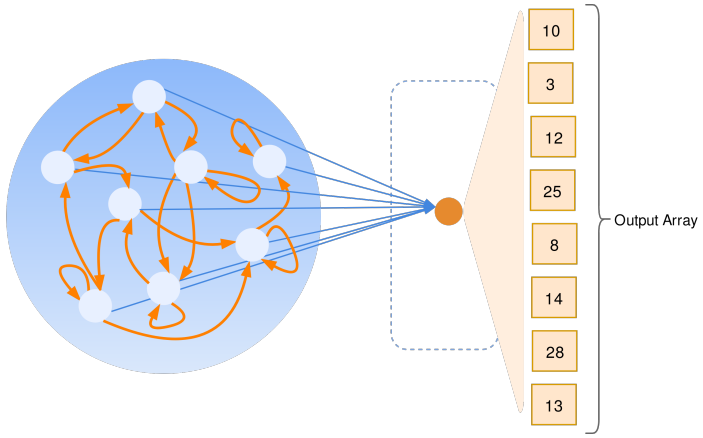


Figure 4.8: Data extraction from readout layer. Each element in the output array belongs to a node state accumulation in the RBN.

4.4 Robot Control

Robot control completes the final step of the simulation process. It enables data transfer from the RC system to the robot. The robot controller receives predictions made at the readout layer level of the RC system and passes these predictions in form of movement directions to the robot. After executing the control application on the Pepper, it applies a move action based on the predicted directions at time t. The next move action happens at time $t + dt$ when the time step dt is over.

Chapter 5

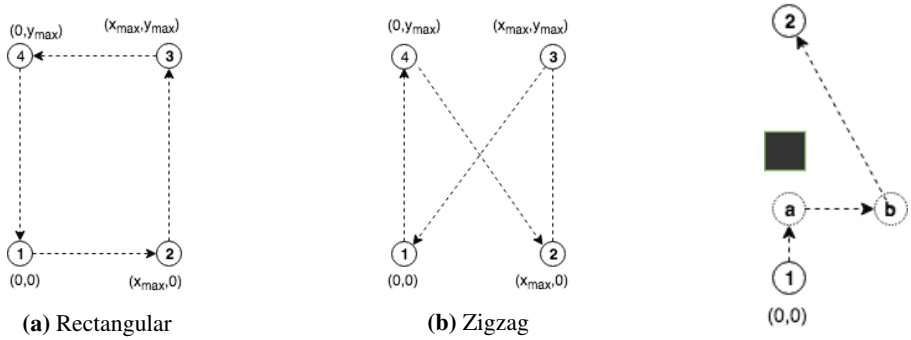
Experiments and Results

During the development process of the simulator platform following experiments are done towards constructing a system with the desired architecture that was introduced at chapter 4. The experiments are divided in two main parts. The first is performing tests on the Pepper robot both for generating data and controlling the robot with predicted actions. The second part is testing the Reservoir Computing system's computational capabilities and performance. In this chapter, the experiment setup is explained followed by an analysis and discussion upon the outcome of the experiment.

5.1 Pepper - Navigation in Unknown Environment

A coordinate system is defined to deal with relative changes in robot local position to the global positions in a room (see Figure 4.3a). Using this coordinate system, you can give global coordinates to the robot independent of its local position. Pepper's local positioning is based on its location at start-up time. When the robot is turned on, a World Frame position is initiated to a value close to zero in each axis (x, y, z). Movement in positive x-axis defines the forward direction, movements in the positive y-axis defines the left-side direction. Movements in z-axis are from robot torso and its head upwards defined in the vertical direction. In these experiments no movements in z-axis take place.

Executing any navigation tests on the Pepper requires communicating with robot's operative system and its sonar sensors. So as the first step, a TCP connection to the robot OS is established to enable the communication. Next, the application for the navigation is executed given $C = \{c_0, ..., c_k\}$ a sequence of 2D coordinates assuming c_0 to be the start point and c_k the final point. The robot standing at coordinate c_i calculates a path toward the target c_{i+1} and then it executes an asynchronous move to its destination. Sonar measurements are recorded every $50ms$ during the asynchronous moves. At the time of arriving to target destination, the robot calculates a new path for the next target coordinate and so on. Otherwise, it continues up to 15 tries to reach its target. If any obstacle appeared in a distance closer than $0.6\ m$ to the robot, the robot stops immediately and handles the



(a) Rectangular **(b)** Zigzag **(c)** Obstacle

Figure 5.1: Paths sketches on the coordinate system. Pepper starts at point 1 and navigates to its checkpoints. Navigation ends at start point in case (a) and case (b).

obstacle based on the strategy described at 4.2.2. As the next action it recalculate a new path to reach its considered destination.

5.1.1 Validating Navigation Algorithm

Several paths were designed to approve the correctness of the navigation algorithm in the pre-thesis project [2]. Both the rectangular path[1] at Figure 5.1a and the zigzag path[2] at Figure 5.1b were used to validate the path calculation of the algorithm within an area without any obstacles. Figure 5.1c shows the obstacle handling[3] when an obstacle is detected in the considered path. Video records from these experiments are available at YouTube (See footnotes).

5.2 Generating Sonar Dataset

5.2.1 Recording Raw Sonar Measurements

In order to generate a training dataset for the reservoir, a navigation path of 15 coordinates is defined including 5 obstacle objects located between coordinates. Figure 5.2 illustrates the coordinate system of the testing area showing the target positions and the location of obstacles. The results from this test is tabulated at Table 5.1. The robot registers Sonar values every $50\ ms$. The total time for completing this task is around $5.2\ min$ resulting in 6237 distance values in a range of $[0.3, 5.0]$ meters. A video[4] showing one of the test navigation with obstacle avoidance is recorded.

[1]Rectangular path: `https://www.youtube.com/watch?v=Js16GBAkjMQ`
[2]Zigzag path: `https://www.youtube.com/watch?v=8IB9cZ-wwsE`
[3]Obstacle avoidance: `https://www.youtube.com/watch?v=_Doic-AN3l0`
[4]Navigation test with obstacle avoidance: `https://youtu.be/bw9sHoqyxb8`

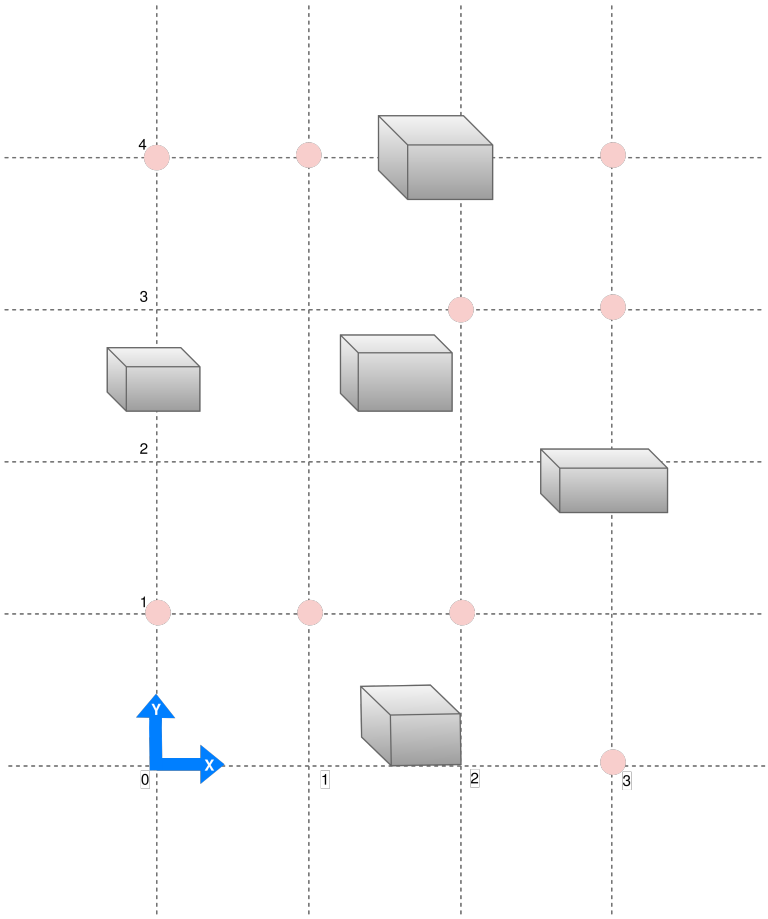


Figure 5.2: Illustration of the navigation area for the generated training dataset. It contains 5 obstacles shaped as boxes. The blue 2-sided arrow represents the Pepper robot and its orientation, gray boxes represents obstacle objects and the red circles represent the target coordinates. The navigation path is defined as a sequence of 2D coordinates.

Table 5.1: Results for navigation test consisting of a path with 14 target coordinates in a room with 5 obstacles located between several target points.

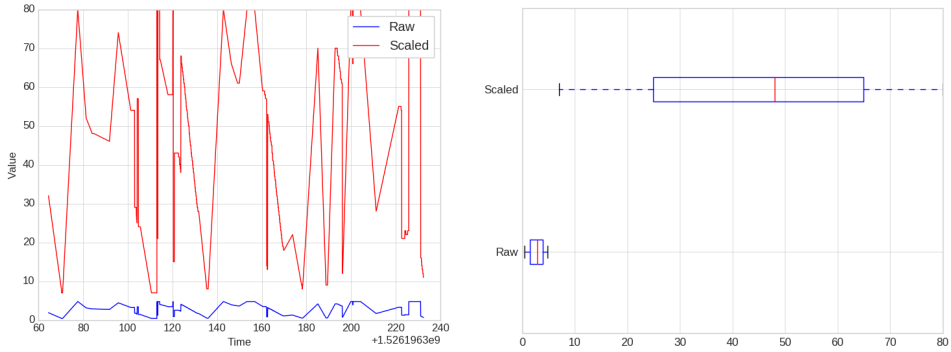
Configurations		**Results**	
Nr. Obstacles	5	Path Duration (min)	5.2
Nr. Targets	14	Dataset Length	6237
dt (ms)	50		

5.2.2 Labeling Data Records

The data values are labeled during the asynchronous move action in the navigation path. Each distance is assigned with a label $L_i \in L$ where $L = \{$*"forward", "obstacle", "stop", "turn_left", "turn_right"*$\}$ is a set of labels.

5.2.3 Scaling Dataset

Raw data from Sonar sensor is not applicable as training data for the reservoir. Therefore it is processed through several steps explained at subsection 4.2.4. It is mainly scaled up to signify the difference between small distance values. Figure 5.3 shows the scale of the Sonar values before and after the pre-processing.



(a) **(b)**

Figure 5.3: Comparison of the raw data with scaled data for the first 1000 values in the dataset. (a) Line plot . (b) Box plot

5.3 RC Execution

Experiments to analyze the RC system performance and RC dynamical properties is done using the parameter configurations that are showed in Table 5.2. Except the RBN parameters, $P(XOR)$ and perturbation period p are examined. $P(XOR)$ refers to the probability of amount of XOR node in a combination with TLG nodes. p refers to the number of RBN execution at each perturbation. The process of executing the RC system is illustrated by a

state machine diagram at Figure 5.4. Suppose that the RBN is in state S_i and is perturbed by an input value. After each perturbation, the RBN is executed in a time period of p. So the RBN will be in state S_{i+p} after the end of $i-$th perturbation.

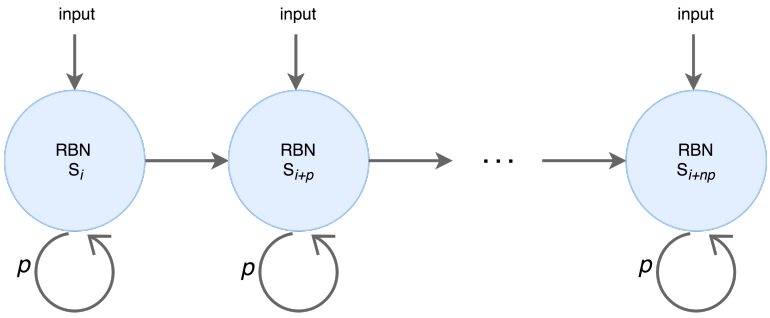


Figure 5.4: State machine diagram that illustrates the execution process of the RC system during the experiments. Each blue circle represents a reservoir and the circle arrow represents a periodic execution of it. S_i is the RBN state at time i. When the RBN is perturbed by an input, it is executed for a period of p. n shows the numbers of input values used to perturb the reservoir.

Because of large overlap in input values labeled as *"turn_left"* and *"turn_right"* and because of law accuracy of the system we decided to use a dataset containing data values labeled with only *"turn"*. A dataset containing two turn labels contains very similar value range. Variation of each movement category in the dataset is illustrated at Figure 5.12 that compares the similarity of those two *turn* categories.

Table 5.2: Tested configuration parameters for RBN reservoir.

RBN Parameter	**Configuration**
RBN size (N)	$\{50, 100, 125\}$
Input connectivity	80%
$P(XOR)$	$\{0.25, 0.5\}$
Connectivity (K)	$\{(1, 2, 3), (2, 3), (2)\}$
Perturbation period (p)	$\{10, ..., 100\}$

5.4 RC System Performance Evaluation

Accuracy is selected to measure the performance of the RC system. Accuracy is a ratio that shows in which degree prediction results conform to the correct values ($Accuracy = \frac{nr.correct\ labels}{nr.incorrect\ labels}$). Accuracy values lie in range $[0, 1]$ where 1 represents 100% correctness and 0 represents total failure of the predictions. Different configurations for RBN are tested to analyze the effect of each parameter on the performance of the RC system to predict the correct directions of the robot.

(a) N = 50, $P(XOR) = 0.25$

(b) N = 50, $P(XOR) = 0.50$

(c) N = 100, $P(XOR) = 0.25$

(d) N = 100, $P(XOR) = 0.50$

(e) N = 125, $P(XOR) = 0.25$

(f) N = 125 , $P(XOR) = 0.50$

Figure 5.5: Comparison of accuracy for different K connectivity. (Left) $P(XOR) = 0.25$. (Right) $P(XOR) = 0.50$

In addition to the accuracy as a measure of total performance, other metrics are selected to analyze the generalization ability of the system on each category separately. Precision, Recall and F1-score are metrics that are used to evaluate the machine learning classification models. *Precision* is the ability of the classifier not to label negative samples as

positive where *Recall* is the ability of the classifier to select all positive samples. *F1-score* is a combination of these two metrics showing the relation between positive samples of test set and the samples labeled positive by the classifier [45].

Table 5.2 shows the selected parameters to generate different RBNs. The input connectivity of the RBN reservoirs is a dynamic value which differs based on the RBN size. This value is chosen to be 80% of the total number of the nodes in the reservoir to make the input perturbation as significant as possible. The experiments in the following sections show RC executions with different perturbation periods compared with the calculated accuracy.

5.4.1 Analysis of Accuracy and RBN Parameters

Figure 5.5 shows performance comparison of the simulator with different RBN size N across perturbation periods of 10 to 100 time unites. All of the sub-figures compares the different $K-$connectivities and probabilities of $XOR-$nodes in the RBN network. Here, K and $P(XOR)$ is defined as following; $K \in \{\ k_1 = \{1, 2, 3\},\ k_2 = \{2, 3\},\ k_3 = \{2\}\}$ where $P(XOR) \in \{0.25, 0.50\}$. Here we want to see how the RC-system performs and how the various configurations influence the results.

Table 5.3: Accuracy results for RBN with size *N=50*

RBN Configuration			**Accuracy**		
N	$P(XOR)$	K	Min	Max	Avg
50	0.25	1, 2, 3	0.60	0.67	0.64
		2, 3	0.58	0.66	0.63
		2	0.56	0.67	0.61
50	0.50				
		1, 2, 3	0.52	0.66	0.62
		2, 3	0.52	0.68	0.61
		2	0.52	0.65	0.60

Table 5.4: Accuracy results for RBN with size *N=100*

RBN Configuration			**Accuracy**		
N	$P(XOR)$	K	Min	Max	Avg
100	0.25	1, 2, 3	0.58	0.68	0.64
		2, 3	0.55	0.69	0.63
		2	0.61	0.67	0.65
100	0.50				
		1, 2, 3	0.57	0.67	0.62
		2, 3	0.53	0.68	0.60
		2	0.53	0.68	0.61

Table 5.5: Accuracy results for RBN with size *N=125*

RBN Configuration			Accuracy		
N	$P(XOR)$	K	Min	Max	Avg
125	0.25	1, 2, 3	0.65	0.71	0.67
		2, 3	0.57	0.68	0.63
		2	0.64	0.69	0.67
125	0.50				
		1, 2, 3	0.56	0.67	0.63
		2, 3	0.54	0.63	0.59
		2	0.53	0.67	0.61

Quantity of XOR nodes among the thresholds nodes

Threshold logic gates were used as node activation functions in the RBN reservoir to generate a digital reservoir that carry closest properties of a biological neuron culture reservoirs. To investigate if usage of threshold logic gates can produce dynamics in the RBN reservoir similar to logic gates with Boolean functions, we include only XOR functions to create variation in node outputs. The probability of 25% and 50% is tested and the results are gathered in Table 5.3, Table 5.4 and Table 5.5. Comparison of accuracy results shows some deterioration when the probability of XOR nodes is increased to 0.5. So this might suggest that a reservoir containing threshold logic nodes has the ability of interpreting data and producing dynamics in the system.

$K-$connectivity

The aim of constructing this simulator is to imitate the biological neural cultures properties. One of their properties is their heterogeneity. The relation between neurons is variate and to compensate for this we want to simulate such relations and achieve a heterogeneous neural network model that matches the structure of the actual neural cultures. $k_1 = \{1, 2, 3\}$ was meant to be a heterogeneous representation and $k_2 = \{2, 3\}$ was tested to see if better dynamics or a less chaotic reservoir could be achieved. $k_3 = \{2\}$ was tested to see if the chaos in the reservoir could be prevented because Kaufmann [46] stated that homogeneous reservoirs with $k = 2$ can achieve the edge of chaos. In this research, the heterogeneous networks performed as well as the homogeneous network. To emphasize the capabilities of heterogeneous RBNs, the maximum accuracy of 0.71 in Table 5.5 was reached by a heterogeneous RBN with k_1-connectivity and size 125. This peak is also illustrated by the blue curve at Figure 5.5e. Looking at minimum, maximum and average accuracy results in Table 5.3, Table 5.4 and Table 5.5, we cannot conclude with significant performance differences between the various $K-$connectivity. Despite the average results, the blue curve for k_1 connections at Figure 5.5e displays a more stable progress and less sensitivity to the various perturbation periods. In addition, the blue curve is kept over the two other curves most of the time and never falls under 0.65, showing that it performs better in average compared to the other curves. But in general, both the blue curve and the orange curve (which represents RBNs with k_3-connectivity) performs very similar where the average results documented in Table 5.5 are equal for $N = 125$ and $P(XOR) = 0.25$.

RBN size

In Figure 5.5, the sub-figures in the left 5.5a, 5.5c and 5.5e shows the results for $P(XOR) = 0.25$ for $N = 50, 100, 125$ accordingly. Since the results for $P(XOR) = 0.25$ seemed to be more satisfying, we want to analyze the effect of RBN size on the performance on the left-sided sub-figures. In Figure 5.5e RBN has the largest size and the RC has achieved the highest average accuracy for all the three curves. Probably, enlarging the size of RBN would decrease the loss to some degree. Given the same dataset and labels for the current experiments, a minor improvement is presumed to be demonstrated by larger RBN reservoirs. Because of the resource limitations, testing the larger networks are omitted in this research.

Conclusion

Adding XOR nodes with probability of 0.25 creates good dynamics in the system combined with TLG nodes in the RBN reservoir. Using heterogeneous networks are also suitable to achieve desired dynamics in the RC system where the results for such heterogeneous systems were satisfying in our experiments. But other RBN configurations must be adjusted and tested in order to increase the correctness ratio in the predictions. We can also assume that increasing the size of the RBN reservoir could help decreasing the loss. But the drawback of using larger networks would imply higher time complexity in the simulation. Varying the perturbation period did not show significant changes in classification accuracy. But choosing low perturbation period might prevent the RBN to reach steady-state that means the RBN will be interrupted and produce chaotic dynamics more than reaching the edge of chaos.

To summarize, the simulator was able to generate a dynamic system which was responsive to the external fluctuations. Achieving optimal results for the navigation task require experimenting with various configurations of RC system and generating the input dataset either using other sensors or label them in another way.

5.4.2 Analysis of Generalization on each Category

Confusion matrix

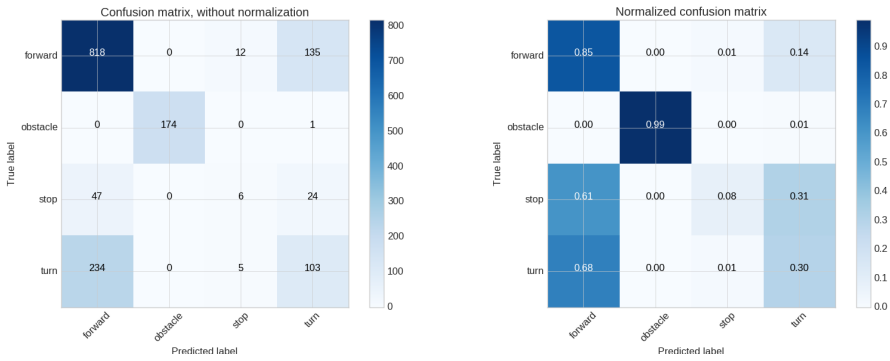
A confusion matrix is a visualization used to evaluate the output quality of a classifier. Figure 5.6 shows a confusion matrix for a RC execution with an accuracy equal to 0.71. The elements along the major diagonal represent the number of correctly labeled samples and the other elements elements represent the numbers of samples labeled wrong by the classifier [47]. Predicted labels are located on $x-$axis and the true labels are on $y-$axis. Higher value on the diagonal indicate better prediction ability.

Figure 5.6 shows the confusion matrix for the prediction results of a RBN of size 125 without normalization 5.6a and with normalization 5.6b. The values in diagonal of shows the correct predictions by the ridge classifier. Figure 5.6b shows that 99% of the samples with *"obstacle"* label are classified correctly but only 8% of the samples with *"stop"* label. The reason behind the low accuracy on samples with *"stop"* and *"turn"* is that the variation of the values for these categories is high and there are some outsider values. The variation of the test values is illustrated in Figure 5.12b. One possible reason behind the

Table 5.6: Showing configurations and results for the most accurate RBN RC execution test.

Configurations		**Dynamics**		**Results**	
RBN size	125	Nr. attractors	6237	Accuracy	0.71
K-connectivity	(1, 2, 3)	Avg length attractors	33.9	Avg precision	0.67
Input connection	100	Transient time	25.1	Avg Recall	0.71
Perturbation period	60			Avg F1-score	0.68
$P(XOR)$	0.25				

wrong predictions for *"stop"* category is that the robot can stop anywhere to change its direction to find a new path or walk towards the next target point.



(a) Confusion matrix without normalization

(b) Confusion matrix with normalization

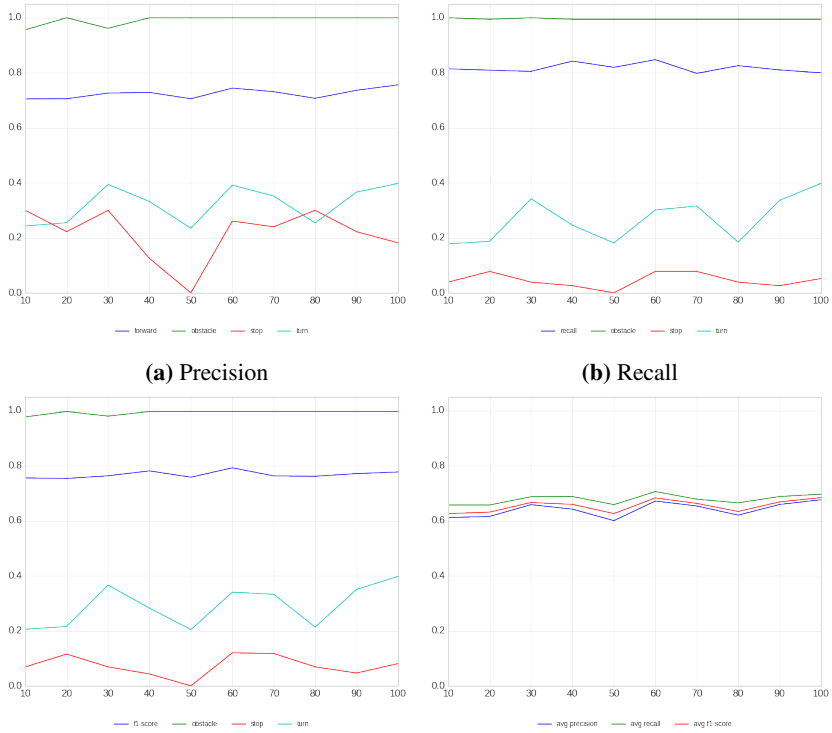
Figure 5.6: Confusion matrix showing 99% correctness for classifying *"obstacle"* category. $N = 125$, perturbation period $= 60$, $K-$connectivity = *(1, 2, 3)*, $P(XOR) = 0.25$ and input connections = 100.

Precision, Recall and F-score

Precision, Recall and F-score ratios are calculated for each category separately. Figure 5.7 shows similar evaluation as in the confusion matrices. Again we can observe that the *"obstacle"* and *"forward"* categories are labeled correctly by the classifier. This is also another sign to the challenges about classifying the *"stop"* and *"turn"* categories.

5.5 Validating Correctness of RBN RC System

The RBN RC system perturbed by Sonar data values from Pepper performed poorly. Hence, a couple of validation tests were required to analyze the reasons behind the poor performance. The results from these tests are useful to analyze the provided dataset characteristics and its effect on the performance of the RBN RC system. The correctness of the RBN RC operations and its data interpretation ability is approved by good performance



(a) Precision

(b) Recall

(c) F1-score

(d) Avg precision, recall and F1-score

Figure 5.7: Showing precision, recall and F1-score results for each separate category.

for a different problem or dataset. The following tests are introduced to validate the correctness of the RC system.

Randomly generated values

The firs validation test is to train RBN RC systems with a set of randomly generated *float* values. A number of 5000 float values are generated randomly with uniform probability in range $[0.3, 5.0]$ to represent sonar output values. The values between $[0.6, 0.8]$ are labeled as *obstacle*, values less than 0.6 are labeled as *stop* and the ones greater than 0.8 are labeled as *forward*. Figure 5.8 shows that the RC system works optimally for the random generated dataset. This validates correctness of the implemented RBN RC system is valid. Here, we may conclude that the variation in the data from real sonar datasets are hard to classify. An analysis of the dataset is explained in section 5.7.

Temporal parity

The second validation test is chosen to be *Temporal parity* and represent a problem different from the navigation problem. Also, the RC systems for this task showed good

generalization capabilities in previous researches such as [1] and [30]. For this task the RC system is trained by an input stream of bits. In this case the reservoir is able to keep information for a sliding window of n bits injected into the reservoir in the past and an offset value t. The task is to evaluate the n bits to determine if an odd number of 1 values contains in the sliding window [30]. The RBN RC system generalization performance resulted in an accuracy over 90% for some RC executions with sliding window size of $n = 3$ and $t = 0$. The plot at Figure 5.8 shows the accuracy of the predictions for a heterogeneous RBN with $N = 100$ nodes where the states are initiated randomly for each run. The boxplot in Figure 5.8 does not show a much better average accuracy for this task but it has reached high accuracy over 90% in some executions. We can assume that it is possible to reach higher average accuracy for this task if the RBN configuration is customized for this task.

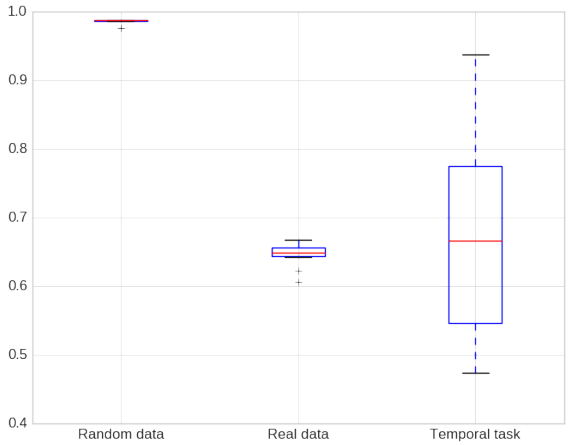
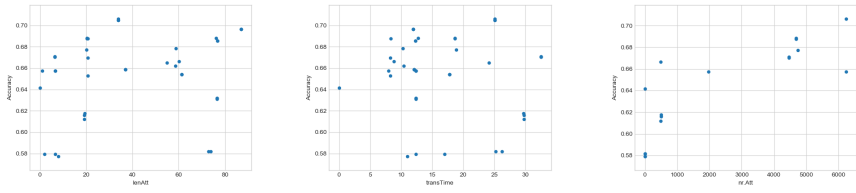


Figure 5.8: Performance comparison of RC execution with randomly generated dataset, real data and temporal parity task. It shows 99% accuracy for randomly generated dataset.

5.6 Analysis of Dynamics

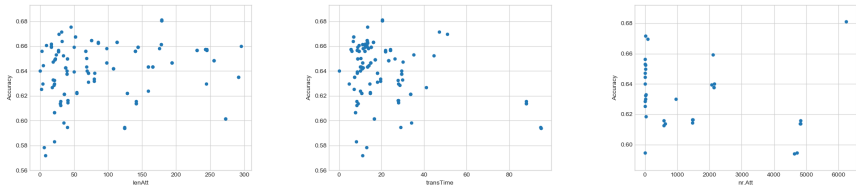
RBN behaviors are recorded during the execution of RC system. Here we want to see if any of the registered behaviors is related to the system dynamics and its performance. Generalization performance of a RC system consisting of a heterogeneous RBN with size 125 and 100 is illustrated in Figure 5.9 and Figure 5.10. Figure 5.9a shows the relation between accuracy on $y-$axis and average attractor length on $x-$axis. It is difficult to see any clear distinction for short and long attractor lengths . The maximum achieved accuracy greater that 0.70 has an average attractor length of 34. But this is not enough to conclude anything about attractor length and the effect of it on correctness of the system decisions. The reason is that both high and low accuracy values are achieved for short and long attractor lengths. The same conclusion can be applied to the RC system with heterogeneous and homogeneous RBN of size 100 which is illustrated at Figure 5.10a and Figure 5.11a. Looking at average transient time at figures 5.9b and 5.10b, most of

the experiments have resulted in an average transient time between 5 and 20. Looking at numbers of attractors for each experiments in all three RC systems shown in 5.10b and 5.11c we can see that most of experiments have a number of attractors between 0 and 10000. In Figure 5.9c we cannot say anything specific because of the low number of performed tests. Figure 5.11 illustrates the relation between the derived properties and



(a) Avg length of attractors **(b)** Avg transient time **(c)** Number of attractors

Figure 5.9: Scatter plot compares the effect of RBN dynamics on accuracy results for a heterogeneous RBN with $N = 125$ and $k = (1, 2, 3)$.



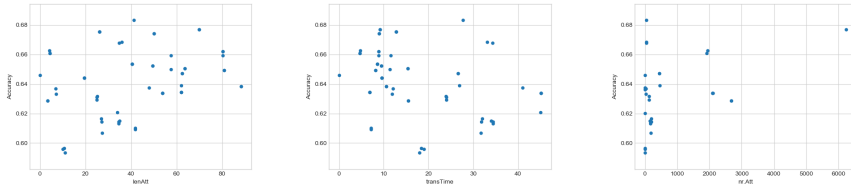
(a) Avg length of attractors **(b)** Avg transient time **(c)** Number of attractors

Figure 5.10: Scatter plot compares the effect of RBN dynamics on accuracy results for a heterogeneous RBN with $N = 100$ and $k = (1, 2, 3)$.

accuracy for a homogeneous RBN. Figures 5.11a, 5.11b and 5.11c show average attractor lengths, average transient time and number of attarctors for the a homogeneous RBN with size 100 and $K-$connectivity of 2. The accuracy results under 0.60 5.11a belong to RBNs with short attractor length at the same time some experiments resulted in accuracy above 0.66 with similar lengths. Looking at the average transient time, Figure 5.11b show spread and various results that makes it difficult to conclude anything about the relation between the transient time and the accuracy.

Conclusion

No clear relation between the different RBN properties and the achieved accuracy results is detected. But results show that the RBN RC systems are not highly sensitive to changes and do not tend to evolve toward chaotic behaviour.

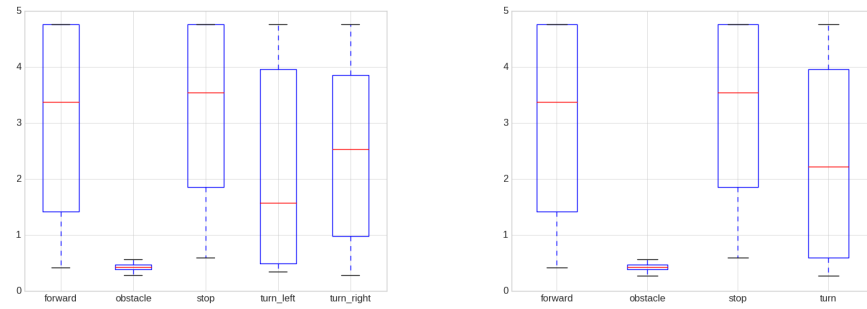


(a) Avg attractor length **(b)** Avg transient time **(c)** Number of attractors

Figure 5.11: Scatter plot compares the effect of RBN dynamics on accuracy results for a homogeneous RBN with $N = 100$ and $k = (2)$.

5.7 dataset Analysis

The reason behind the law performance of the RC system is assumed to be the dataset and the variation of each data category and the limitations behind the sensor which was used to generate the dataset. In the current dataset, some values have the potential to be classified with multi-labels. This section will describe some drawbacks of the dataset which was utilized for these experiments.



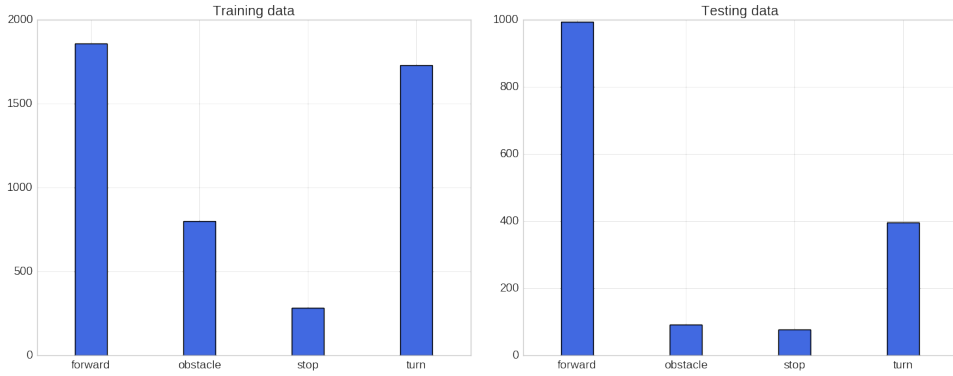
(a) Dataset containing 5 categories including both *turn_left* and *turn_right*. **(b)** Dataset containing 4 categories including only *turn* category instead of *turn_left* and *turn_right*.

Figure 5.12: Comparison of the raw sonar data with different number of categories.

5.7.1 Variation in Dataset and Distribution of each Category

Figure 5.12 shows a comparison of different movement categories in the raw sonar data. Figure 5.12a shows the original labeled data variation where the overlap for *turn_left* and *turn_right* was large. The dataset is therefore relabeled with only four categories to reduce the overlapping data values. The new distribution of sonar values for each category after relabeling is shown by Figure 5.12b.

Figure 5.13 shows the numbers of each category for training and testing sets. Even the overlap for *forward* and *stop* category is still high after the relabeling, the numbers of data values labeled as *stop* are much fewer both in the training set and the test set. Figure 5.14



(a) Training dataset **(b)** Testing dataset

Figure 5.13: Distribution of each category in the dataset.

shows how the samples are labeled compared to the existing numbers of each category in the test sample. We can see that the classifier has labeled more values as *forward* category compared to the existing true labels. Many values are also labeled as *turn* but not all of those labels are true positive. Figure 5.6 shows that only 30% of the samples were correctly labeled.

In general, such sequential data values that do not lie in a specific range are difficult to classify. One suggestion to improve the results could be to think of another way of categorizing the data and apply new experiments. Another solution could be using other sensors that could generate other types of data that might be easier to classify.

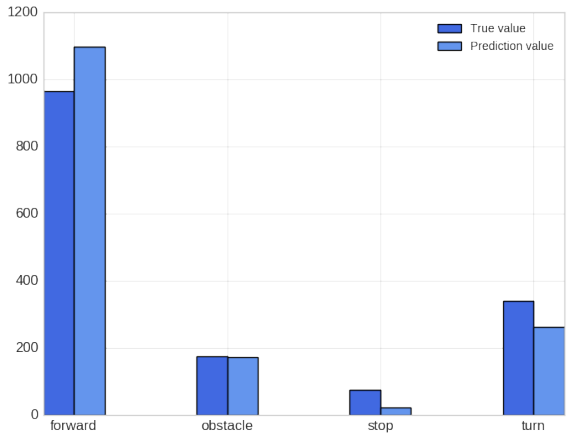
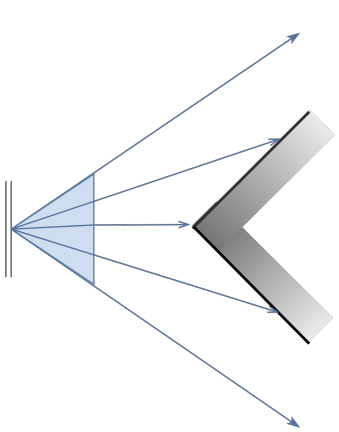


Figure 5.14: Bar plot showing the number of predicted labels for each category compared to the existing number of true labels in the test dataset.

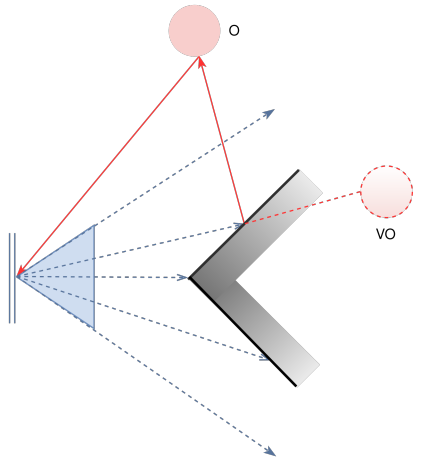
5.8 Sonar Sensor Limitations

Distance measurements in a Sonar system depends on several environmental factors such as distance, shape and characteristics of the reflector surface and viewing angle [48]. Using the Sonar to be the only sensor caused unexpected move actions during obstacle avoidance experiments. The robot collided with objects very often. It turned out that the sensor was not able to detect objects when receiving echoes from convex corners Figure 5.15a. The reflection of sound waves is illustrated at Figure 5.15b where the sensor suffers from false ranging. It basically can measure distance to a virtual object based on the reflection of echoes to objects out of the beam range. Similar situation can cause noise in the dataset that was generated during the navigation.

The sensor used in this experiment is a Sonar sensor with an effective cone of a θ equal to 60 °. More details about the sensor properties is available at Table 3.1. The objects appeared in this cone are likely to send echos and cause noise in the dataset which could be a reason to achieve poor accuracy results. The robot movement controlled by predicted directions are recorded. The recorded videos [5] show unsuccessful movements very early in the test path. The limitation of sonar sensors can be solved using the algorithms to detect corners first and calculate a path based on more accurate sensor values afterwards. Sonar sensor works well for walls with flat surfaces but suffers from false ranging.



(a) Convex Corner



(b) Convex corner and false ranging

Figure 5.15: (a) shows how the sound waves (blue arrows) are sent to a convex corner where (b) shows the receiving echos (red arrows) and the resulted false ranging by the sonar sensor. The dotted red circle is the virtual object detected by the sonar sensor.

[5] Failure (1): `https://www.youtube.com/watch?v=-jbrs8vwjAk`
Failure (2): `https://www.youtube.com/watch?v=5lywlsdtTcw`
Failure (3): `https://www.youtube.com/watch?v=2nlt3FbP1ik`

Chapter 6

Conclusion

6.1 Conclusion

In this project, we have implemented a simulator for a bio-digital machine and outlined an infrastructure that connects the simulated RBN reservoir to the Pepper robot. The simulator is a functioning system that enables communication between the Pepper robot and the RC system. This is achieved by mapping output data from robot sub-system to a readable data form for RC sub-system, and mapping the output data from the RC sub-system to the robot motor control unit.

Experiments confirm that using TLG nodes do not prevent the system from producing desirable dynamical behavior. In addition, the simulator is assumed to be a close imitation of the living neural network because the neural cultures constitutes a heterogeneous network with threshold based spiking behaviors. TLGs are suitable as activation functions for digital reservoirs combined by a distribution of XOR nodes of 0.25 probability. This combination helps preventing the system to reach an ordered structure. The simulator generates heterogeneous RBNs that can create dynamics in the reservoir system. Here the heterogeneous property allows us to select unlike number of links between the nodes in the RBN something that makes the simulation more similar to the biological properties of neuron cultures. The achieved performance results showed to be unsatisfying. The maximum accuracy was 0.71 for a heterogeneous RBN with size 125. The poor accuracy results underlies several reasons. The main reason is assumed to be the skewed distribution of the categories in the training data set. The data set might not be a representative perception of the navigation environment because the Sonar sensor was not able to generate accurate and noise free data records, i.e. real-world challenges.

The implemented simulation system has proved that integrating the living neural networks in a digital platform is viable by exploiting the Reservoir Computing paradigm. It has potential to achieve accurate predictions by new adjustments of the data set and the reservoir configurations. The real-world challenges are present here, as for all robotic work. However, the experimental results show that the proposed RC-neuron-robot approach is a sound way toward achieving the goal of the NTNU Cyborg project.

6.2 Future Work

The next big step is to replace the simulated RBN reservoir with the actual living neural culture. The results show that such an approach to construct a hybrid machine is viable. However, there are several possibilities toward improving the simulated RC-robot framework. A suggestion to improve the current system would be to generate the dataset in another way, either by combining Sonar sensor with other Pepper sensors or experiment with other sensors. This might improve the training dataset in a way that is easier to differentiate. One other suggestion would be to modify the dataset labeling or change the task design totally. An example would be extracting the motor and wheel velocity during the navigation that can increase the probability of reaching accurate predictions. But this requires applying new experiments and few changes to the system.

Bibliography

[1] Aleksander Vognild Burkow. *Evolving Functionally Equivalent Reservoirs for RBN Reservoir Computing Systems*. 2015.

[2] Neshat Naderi. *Interfacing Living Neural Cultures to Control a Robot*. Pre-thesis project for this master thesis., 2017.

[3] Peter Aaser. Investigating in-vitro neuron cultures as computational reservoir, 2017.

[4] Julian F Miller, Simon L Harding, and Gunnar Tufte. Evolution-in-materio: evolving computation in materials. *Evolutionary Intelligence*, 7(1):49–67, 2014.

[5] Charles Darwin. *On the origin of species, 1859*. Routledge, 2004.

[6] Peter Aaser, Martinius Knudsen, Ola Huse Ramstad, Rosanne van de Wijdeven, Stefano Nichele, Ioanna Sandvig, Gunnar Tufte, Ulrich Stefan Bauer, Øyvind Halaas, Sverre Hendseth, et al. Towards making a cyborg: A closed-loop reservoir-neuro system. In *Proceedings of the European Conference on Artificial Life 2017*. MIT Press, 2017.

[7] Koen De Bosschere, Albert Cohen, Jonas Maebe, and Harm Munk. Hipeac vision 2015, 2015.

[8] Maktuba Mohid, Julian F Miller, Simon L Harding, Gunnar Tufte, Odd Rune Lykkebø, Mark K Massey, and Michael C Petty. Evolution-in-materio: Solving function optimization problems using materials. In *Computational Intelligence (UKCI), 2014 14th UK Workshop on*, pages 1–8. IEEE, 2014.

[9] Yaneer Bar-Yam. *Dynamics of complex systems*, volume 213. Addison-Wesley Reading, MA, 1997.

[10] Francis Heylighen et al. The science of self-organization and adaptivity. *The encyclopedia of life support systems*, 5(3):253–280, 2001.

[11] George M Whitesides and Bartosz Grzybowski. Self-assembly at all scales. *Science*, 295(5564):2418–2421, 2002.

[12] Melanie Mitchell. Life and evolution in computers. *History and philosophy of the life sciences*, pages 361–383, 2001.

[13] Steven H Strogatz. *Sync: How order emerges from chaos in the universe, nature, and daily life*. Hachette UK, 2012.

[14] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC Press, 2018.

[15] Bruce Hannon and Matthias Ruth. *Dynamic modeling*. Springer Science & Business Media, 2001.

[16] Harold J Morowitz. *The mind, the brain and complex adaptive systems*. Routledge, 2018.

[17] Robert Legenstein and Wolfgang Maass. What makes a dynamical system computationally powerful. *New directions in statistical signal processing: From systems to brain*, pages 127–154, 2007.

[18] Daniel A Wagenaar, Jerome Pine, and Steve M Potter. Searching for plasticity in dissociated cortical cultures on multi-electrode arrays. *Journal of negative results in biomedicine*, 5(1):16, 2006.

[19] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Johannes Jensen and Gunnar Tufte. Reservoir computing with a chaotic circuit. In *Proceedings of the European Conference on Artificial Life 2017*. MIT Press, 2017.

[22] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007*, pages 471–482, 2007.

[23] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.

[24] Wolfgang Maass. Liquid state machines: motivation, theory, and applications. In *Computability in context: computation and logic in the real world*, pages 275–296. World Scientific, 2011.

[25] Mantas Lukoševičius, Herbert Jaeger, and Benjamin Schrauwen. Reservoir computing trends. *KI-Künstliche Intelligenz*, 26(4):365–371, 2012.

[26] Jochen J Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007.

[27] Carlos Gershenson. Introduction to random boolean networks. *arXiv preprint nlin/0408006*, 2004.

[28] Sarma Vrudhula, Niranjan Kulkami, and Jinghua Yang. Design of threshold logic gates using emerging devices. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 373–376. IEEE, 2015.

[29] Donald W Marquaridt. Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation. *Technometrics*, 12(3):591–612, 1970.

[30] David Snyder, Alireza Goudarzi, and Christof Teuscher. Computational capabilities of random automata networks for reservoir computing. *Physical Review E*, 87(4):042808, 2013.

[31] Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. In *European Conference on Artificial Life*, pages 588–597. Springer, 2003.

[32] Matthew Dale, Julian F Miller, Susan Stepney, and Martin A Trefzer. Evolving carbon nanotube reservoir computers. In *International Conference on Unconventional Computation and Natural Computation*, pages 49–61. Springer, 2016.

[33] Manfred E Clynes and Nathan S Kline. Cyborgs and space. *The cyborg handbook*, pages 29–34, 1995.

[34] Kevin Warwick. Cyborg 1.0. *Wired*, 8(2), 2000.

[35] Kevin Warwick. I, cyborg. `http://www.kevinwarwick.com/i-cyborg/`, 2018. Online; accessed 12.01.2018.

[36] About ntnu cyborg project. `https://www.ntnu.edu/cyborg/about`, 2017. Online; Accessed 21.12.2017.

[37] Gunnar Tufte and Stefano Nichele. Ntnu cyborg: making r.u.r take one. Technical report, 2016.

[38] Softbank Robotics and Aldebaran. Who is Pepper? `https://www.ald.softbankrobotics.com/en/robots/pepper`, 2017. Online; Accessed 05.12.2017.

[39] Aldebaran. Naoqi sdk documentation. python sdk - overview. `http://doc.aldebaran.com/2-5/dev/python/intro_python.html`, 2017. Online; Accessed 10.01.2018.

[40] Softbank Robotics and Aldebaran. Technical features. `https://www.ald.softbankrobotics.com/en/robots/pepper/find-out-more-about-pepper`, 2017. Online; Accessed 05.12.2017.

[41] Aldebaran. What is choregraphe. `http://doc.aldebaran.com/2-5/software/choregraphe/choregraphe_overview.html`, 2017. Online; Accessed 10.01.2018.

[42] Aldebaran. http://doc.aldebaran.com/2-5/family/pepper_technical/motors_pep.html, 2018. Online; Accessed 10.01.2018.

[43] William H Strickland and Robert H King. *Characteristics of ultrasonic ranging sensors in an underground environment*. US Department of the Interior, Bureau of Mines, 1993.

[44] Lindsay Kleeman and Roman Kuc. *Sonar Sensing*, pages 491–519. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[45] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[46] Stuart A Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467, 1969.

[47] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[48] Milagrons Martinez, G Benet, F Blanes, P Pérez, and JE Simo. Using the amplitude of ultrasonic echoes to classify detected objects in a scene. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR03), Coimbra, Portugal*, volume 30, pages 1136–1142, 2003.