Directed Exploration in Reinforcement Learning

José Pablo Dávalos Viveros

S1470934

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

2015

Abstract

Reinforcement learning in a high-dimensional task, such as skill learning in robots, can be extremely time consuming if done through the classical methods. This is mainly due to the large amount of actions available and the basically infinite size of the search space presented by th configuration of the robot. Generally, for these kinds of tasks, techniques for continuous reinforcement learning are applied. For this project we propose the use of a self-organizing controller to direct the exploration of the behavioural space in order to learn a useful skill, such as walking, in a reasonable amount of time.

The use of a homeokinetic controller can drive a robot to explore its own abilities based only on the motor configuration and sensor values. The robot can use this controller to find stable behaviours that can serve as a base for the robot to reach a goal with a reinforcement learning algorithm.

The behaviours found by the homeokinetic controller have to be stored for later use. As motor signals developed can be modelled as a time series, Recursive Neural Networks appear to be an appropriate candidate for the task of learning them. In this work, a new method to manipulate these networks to store and regenerate multiple patterns with the same network, called conceptors, is explored in the context of robotics. Through experiments with a spherical robot and a hexapod robot the use of these networks as a robot controller is presented.

Acknowledgements

Many thanks to my supervisor, Dr. Michael Herrmann for all his help in guiding me through the project with his advice and ideas, all the weekly meetings and email communication were key to throughout this process.

I would also like to thank PhD candidate Simon Smith for his technical assistance with the simulator and the implementation of the homeokinetic algorithm.

And finally to my parents and friends for their support and encouragement.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*José Pablo Dávalos Viveros*
S1470934)

Contents

Chapter 1

Introduction

The introduction of robots in society appears to be imminent. The advantages of their use in multiple tasks as opposed to human work are numerous, being able to perform dangerous activities without putting human lives at risk, eliminating human error from sensitive operations, even performing tasks that would be impossible to a human due to strength or size limitations is also a possibility for robots. However full integration of robots still is a distant goal. A truly autonomous robot is not possible with only pre coded skills, this would require infeasible time investment from the developers and one could never guarantee appropriate behaviours when a robot encounters a new situation. For a robot to be useful in multiple environments it should be able to adapt to unforeseen circumstances and to perform new tasks not thought by the developers. To solve this a robot should be able to learn new behaviours, but this is a whole new area of research in the field of artificial intelligence.

One possible way to tackle this issue is to learn based on experience. Just like in nature, when a child or an animal is born they start moving their bodies exploring their locomotive abilities, this way eventually a horse can learn to stand on four legs or a baby can learn to crawl without an external teacher. A robot having the ability to learn these kinds of new skills without an external input or teacher would be an important tool in the way towards autonomy.

The approach proposed in this work is a way towards a robot to learn new behaviours without external interaction. This is an important skill found in nature that any system needs to be truly autonomous, one of the desired characteristic of a robot. By giving the robot a drive to explore based on its sensor values can guide the robot to learn completely new behaviours that allow it to perform new tasks. Chapter 2 reviews the previous literature on which the idea behind the project is based. Chapter 3

presents the hypothesis and the main algorithm to be used. Finally in chapters 4 and 5 the results obtained by some experiments performed are presented and discussed to demonstrate the validity of the hypothesis and a conclusion is drawn. Analysing these results shows that this is a promising approach for a robot to learn new skills that could be further improved upon.

Chapter 2

Background

2.1 Introduction

Introducing robots to society requires overcoming multiple challenges that prevent them from interacting with humans and their surroundings correctly. Being capable of adapting to diverse environments is one of the major obstacles to be faced by the field. The classic approach to robotics kinematics, dynamics, motion planning, computer vision, and control, considered to be the fundamentals of robotics (Spong et al., 2006), need to be improved upon to allow robots to better integrate them into diverse real life situations. This work explores the idea of a robot that can adapt and generate new behaviours without the need of human intervention. So far robots have been able to integrate seamlessly in the manufacturing industry due to the capability of classical robot control to perform repetitive tasks in a static environment. Instead of the robot adapting to the task it is the task that is designed with the robot's performance in mind. This is not possible for most tasks in a real world setting where a robot might encounter unforeseen circumstances not planned before. For these kinds of tasks a robot needs to adapt and learn. Most techniques that try to implement this line of thinking revolve around the concept of *dynamic programming* with methods such as *evolutionary algorithms* (Moriarty et al., 1999) or, in the field of *Machine Learning* (ML) (Bishop, 2006), *Reinforcement Learning* (Sutton and Barto, 1998).

Reinforcement Learning appears to be a promising approach for an agent to adapt to different situations in the environment as it implements exploratory algorithms that try to optimize actions to achieve a goal. In this work a way to apply this technique to a robot is explored. This can turn complicated rather quickly due to the size of the search space that the algorithm must explore to learn anything meaningful. The concept of

self-organization, specifically the homeokinetic controller (Der and Martius, 2010), provides knowledge of the internal system that can be applied to direct the exploration of the reinforcement learning algorithm and significantly reduce the time in which a goal can be reached. This method has been explored before (Martius, 2010) (Smith and Herrmann, 2012) (Smith and Herrmann, 2013), the goal of this project is an attempt to improve on those previous works through the use of *conceptors*, a recently discovered computational mechanism (Jaeger, 2014).

This chapter is a review of some previous literature related this work. In section 2.2 the topic of self-organization in robotics is explored, its origins and the idea behind the homeokinetic controller. In section 2.3 the basis for the conceptor networks are tackled, with a look on Recurrent Neural Networks (RNN), specifically the Echo-state Network (ESN) approach. A brief review of reinforcement learning is presented in section 2.4 with special emphasis on the main differences between discrete and continuous space reinforcement learning, required for its application in robotics.

2.2 Self-organization in Robots

$Self-organization$ is a concept based on the spontaneous creation of patterns in complex systems (Martius et al., 2007). As the name implies, a system can be driven to create organized patterns based only on internal criteria available to the system. These patterns can be interpreted in many ways in dynamical systems where multiple elements are at play. From chemical reaction to economic systems and, of course, robotics a system can spontaneously generate patterns given certain paradigms such as homeostasis or homeokinesis. Homeostasis will drive a system towards a static stable pattern while homeokinesis offers a dynamic counterpart that seems more suitable for robot behaviours. These strategies however, only offer a controller that will probe aimlessly the action space of the robot, to apply them to complex models we need to add a goal-dependant condition, usually through minimizing the error in a closed-loop system or through a reward function (Martius and Herrmann, 2012).

Self-organization is particularly interesting for robotics due to it's ability to generate movement in a robot without external interference, this could be a promising approach towards a truly autonomous robot.The ultimate goal is to generate a robot capable of learning and adapting to it's environment so that it is able to function in all kinds of different situations without human interaction. Being driven by internal goals seems like an appropriate way to achieve this. Self-organization as a field of study in

robotics was first introduced by (Der and Martius, 2010) with that goal in mind. The rest of this section will focus on it's implementation.

2.2.1 Homeostasis

The philosophy behind homeostasis has been used since 1932 for biological systems (Martius, 2010). In this context the system acts to maintain biological variables in an equilibrium, this can be seen in many aspects of natural systems such as blood pressure in animals and the control of various levels of variables within a living body e. g. sugar, sodium, temperature, etc. For robotics, however, it drives the agent towards immobility generating a so-called *lazy* robot that tries to spend as little energy as possible to maintain a resting position (Der and Martius, 2010), where the system is at its most stable.

Self-organization in (Der and Martius, 2010) was introduced to robotics within the context of homeostasis, even though different stable patterns emerged a lack of innovation could be observed, this prevented its use as a good exploratory system. The main idea behind homeostasis, and self-organization for robotics in general, is providing the robot with an adaptive internal model capable of predicting the effects of its actions, in the simplest case the model will predict the sensor values of the next time step and try to reduce the error between prediction to the actual sensor readout. Even though this kind of controller may produce stable non-static behaviours (Der and Martius, 2010) it lacks an element for innovation that will lead the robot to explore a larger part of the state space, homeokinesis was then introduced to deal with this problem.

2.2.2 Homeokinesis

Homeokinesis can be defined as a technique that drives agents to activity and innovation (Der et al., 2012). Homeokinesis basically works in with an opposite philosophy to homeostasis and focuses on a coherent exploratory behaviour (Smith and Herrmann, 2013). The main goal of the controller is to destabilize the dynamics to encourage exploration of the action space, i.e., hoemostasis will drive the system to a minimal prediction error state, generally leaving the system to do nothing, while homeokinesis drives the system away from this state making it unstable. This is achieved through the interplay between minimal prediction error, also called the *Time Loop Error* (TLE), and increased sensitivity in the sensors. This controller has been shown to be able to

drive a system to interesting behaviours, such as hexapod walking (Smith and Herrmann, 2012), (Smith and Herrmann, 2013).

The main learning algorithm for the controller is outlined in (Smith and Herrmann, 2013). The output of the controller u is given by:

$$u_t = s(A(x_t; w^A) + \sigma n_t), \tag{2.1}$$

where s is the sigmoid function, n is a probing signal and $A(x_t; w^A)$ represents the approximator function. w^A is the weight vector for continuous reinforcement learning. Homeokinesis requires an exploration signal defined as:

$$y_t = K(x_t) = g(Cx_t + c), \tag{2.2}$$

where $C \in \mathbb{R}^{m \times n}$ and $c \in \mathbb{R}^m$ are multidimensional adaptive parameters and g is a nonlinear function. The parameters adapt according to the error between the sensory input x_t and the prediction $\hat{x}_t$. This prediction being calculated based on a world model

$$\hat{x}_{t+1} = M(y_t) = Dy_t + d, \tag{2.3}$$

where $D \in \mathbb{R}^{n \times m}$ and $d \in \mathbb{R}^n$. The prediction error ξ_{t+1} is defined

$$\xi_{t+1} = x_{t+1} - \hat{x}_{t+1}. \tag{2.4}$$

So the error in motor space is defined as

$$M(y_t) + \xi_{t+1} = M(y_t + \eta_t), \tag{2.5}$$

where $M(y_t + \eta_t) = x_{t+1}$. Finally η_t is a retrospective error that helps to calculate the error in motor space E_t with the pseudoinverse M'^+ of D, the derivative of M.

$$\eta_t = M'^+ \xi_{t+1}, \tag{2.6}$$

$$E_t = \eta_t^T (J_t J_t^T)^{-1} \eta_t, \tag{2.7}$$

where J is the Jacobian of the sensorimotor loop.

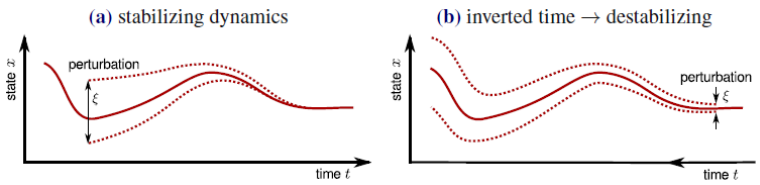

Figure 2.1: Homeostasis drives the system towards stable dynamics while homeokinesis *inverts* the process and gets the system to explore the state space (Der and Martius, 2010)

2.3 Recurrent Neural Networks

Recurrent Neural Networks is one of the two major types of neural networks used in machine learning research (Jaeger, 2002). These differentiate themselves from Feed-forward Neural Network (FNN) by having connections between the neurons in the hidden layer that form cyclic paths. FNN are used for static mapping of the output to the input, being ideal to map non linear functions, through the backpropagation algorithm (Bishop, 2006). By having cyclic paths between neurons RNN resemble the workings of a biological brain more closely, this allows them to represent dynamical systems and temporal knowledge (Borges et al., 2011), but makes training them more complicated.

There are important architectural differences between FNNs and RNNs. As seen in Figure 2.2 FNNs typically consist of multiple layers (input, hidden layers, and output), connecting each neuron on each layer to the neurons on the next layer, always going from input to output. In RNNs the hidden layers are all mixed together in what is called a *dynamical reservoir* (De Raedt et al., 2008). Another important characteristic of RNNs is that the output can also have connections back to the reservoir, allowing to teacher force the output with a sample signal (Lukoševičius, 2012). The dynamical reservoir works as a buffer that is able to store and generate the driving patterns depending on the size and the strength (or weights) of the connections between neurons. This allows the network to reproduce complex behaviours such as chaotic time series like the Mackey-Galss Equation, as demonstrated in section 4.2.1.

There are many ways to train RNNs such as Backpropagation Through Time (BPTT), Real-time Recurrent Learning (RTRL), Higher-order Gradient Descent Techniques, and the Extended Kalman Filter, (Jaeger, 2002) deals with some of these methods, but these methods can run into some problems such as slow convergence and local minima (Strauss et al., 2012). A method that avoids these problems was developed by (Jaeger,

2001) called the echo state approach, explained in detail in the next section.

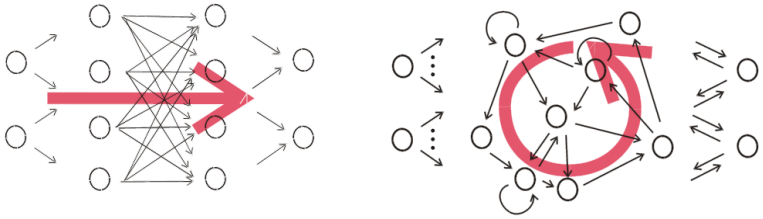

Figure 2.2: Simple illustration of both main types of neural networks. FNN ($left$) and RNN ($right$) (Jaeger, 2002)

2.3.1 Echo-state approach

The echo-state approach for neural networks, also called Echo-State Networks (ESN), was developed in (Jaeger, 2001). The name is derived from the activation state of the output $x(n)$, being a function of previous states can work in the same sense as an *echo* of the previous inputs. Here the recurrent connections are not trained and a random sparse connectivity weight matrix for the reservoir is used instead (Mayer and Browne, 2004). These kinds of network depend on the input history rather than the initial state of the network (Mayer and Browne, 2004). This approach has proven suitable for uses in reinforcement learning (Szita et al., 2006), through the use of function approximation because of the ability to use memory and to avoid convergence problems with other methods of training RNN. Next a method to train a RNN with the echo-state approach is presented, more in depth tutorials can be found on (Lukoševičius, 2012) and (Jaeger, 2002).

The ESNs are one example of supervised Machine Learning, where the desired target output signal $y(n) \in \mathbb{R}^{N_y}$ is known to the algorithm and is used for training. The goal is for the network to generate the output given input $u(n) \in \mathbb{R}^{N_u}$ as closely as possible without overfitting, in Machine Learning terms overfitting is used when a method learns the training set but is not able to perform well compared to other sets not in the training set. To measure the performance of the network an error function is used. in this work we used the Normalized Root-Mean-Squared Error (NRMSE) $E(y, y^{target})$

$$E(y, y^{target}) = \frac{1}{var(y)} \sqrt{\frac{1}{T} \sum_{n=1}^{T} (y_i(n) - y_i^{target}(n))^2}, \tag{2.8}$$

where T is the number of test samples and $var(y)$ is a weighted sum of the variance of the target signal and the generated output. This has an objective interpretation where

$E(y, y^{target}) = 1$ is a constant function of value equal to the mean of the target signal. Generally a value of $E(y, y^{target}) < 1$ should be achieved to use as a reasonable model of the learnt function. To completely define an ESN multiple parameters must be defined, mainly the *leaking rate* of the update equation and the *spectral radius* of the reservoir, the design choices for the ESN will be explored in the rest of this section.

ESNs are a type of RNN with leaky integration, meaning that the previous value has a direct impact in the update equation. The most common update rule is given by the equations

$$\hat{x}(n) = \tanh(W^{in}u(n) + Wx(n-1)W^{bias}), \tag{2.9}$$

$$x(n) = (1-\alpha)x(n-1) + \alpha\hat{x}(n), \tag{2.10}$$

where $x(n) \in \mathbb{R}^{N_x}$ is the vector of neuron activations of the reservoir, $\hat{x}(n)$ is the update term, $W^{in} \in \mathbb{R}^{N_x \times N_u}$, $W \in \mathbb{R}^{N_x \times N_x}$, and $W^{bias} \in \mathbb{R}^{N_x}$ are the weight matrices of the connections between input neurons to the reservoir, recurrent connection in the reservoir, and weights of bias unit respectively (these weights are chosen at random with certain restrictions discussed later), and $\alpha \in (0, 1]$ is the leaking rate that determines the influence of previous inputs in the update equation, there is a special case where $\alpha = 1$ where the model is said to be without leaky integration. The tanh sigmoid wrapper is the most commonly used one, but others can work too. The output is then determined by the linear mapping

$$y(n) = W^{out}x(n), \tag{2.11}$$

where $y(n) \in \mathbb{R}^{N_y}$ cn be a vector or a scalar depending on the number of neurons at the output of the network and $W^{out} \in \mathbb{R}^{N_y \times N_x}$ is the only weight matrix to be trained by the algorithm. There can be variations to this scheme such as adding a direct connection from input to output and a bias term as well as a $W^{back} \in \mathbb{R}^{N_y \times N_x}$ weight matrix where there is a feedback connection from the output to the reservoir for teacher forcing the training update, some of these are explored in section 4.2.1 as well as justification of the parameters used in the experiments. After training, the final weight matrix W^{out} is used to generate the testing data or predict the next element of the driving pattern.

One of the most important characteristics of an ESN is the design of the reservoir. The reservoir serves two main purposes, it works as a non-linear mapping of input to output as well as a memory of the input to provide temporal context, this will generate the desired behaviour of the functions to be learned. Most of the design parameters for the ESN are relevant to the reservoir, the main two being the spectral radius and the leaking rate, although the size and sparsity of the reservoir as well as the scaling

should be taken into account.

The size of the reservoir determines the ability of the network to generate non linearity. A general rule is that the more complicated the task the bigger the reservoir should be, however one should take into account the memory available and the speed of the processor, as larger reservoirs tend to slow down the training step and overfit the model without proper regularization (required for training the output weights). The minimum number of neurons should be at least the number of inputs times the number of time steps the value of an input has to be remembered to solve the problem. A small reservoir can be used to select the rest of the parameters and then scale it according to the task, in the case of robotics a reservoir size in the low hundreds is reasonable to keep the processing as close to real time as possible. Sparsity in the connections of the reservoir is recommended to allow faster updates to the learning algorithm, although this is not as crucial in smaller tasks a small number of connections should be chosen and distributed normally among the reservoir.

The next parameter to be defined is one of the most important for ESNs, the maximal absolute eigenvalue of the reservoir, also known as spectral radius $\rho(W)$, ensures the echo state property of the network. The echo state property ensures that the state of the network is independent of the initial conditions given a long enough input. The spectral radius is the main parameter that determines the stability of the reservoir activations and the amount of time a previous input is going to influence the output. A general rule for choosing an appropriate spectral radius is that $\rho(W) < 1$ guarantees the echo state in most cases. However $\rho(W) \geq 1$ can also be used in tasks that require more memory of the input, testing is required to find the optimum $\rho(W)$.

Even though the weight matrices W^{in} and W are chosen at random, the scaling of these weights is important to determine the overall nonlinearity of the system. As a general rule for more linear tasks the input weights should be closer to zero, the reservoir weights can be taken from a distribution of [-1,1] as $\rho(W)$ is responsible for the scaling. A scaling of 1 can also be used for W^{in} if the inputs are normalized.

The previously mentioned leaking rate can be seen as the speed of the update of the reservoir. This parameter has an exponential effect on the dynamics of the reservoir, meaning that the speed-up will not be linear as α approaches zero. This parameter is also difficult to determine in an objective way without trial an error, testing in intervals of 0.01 is recommended.

The final parameter to be determined is the regularization term to determine the final output matrix W^{out}. The linear regression method for this term can either be

performed by ridge regression

$$W^{out} = Y^{target} X^T (XX^T + \beta I)^{-1}, \tag{2.12}$$

or the direct pseudo inverse solution

$$W^{out} = Y^{target} X^{+}, \tag{2.13}$$

where X^+ is the $Moore - penrose\ pseudoinverse$ (Bishop, 2006). As the pseudo inverse tends to be more expensive memory-wise ridge regression is preferred and the parameter β has to be optimized. A simple way to optimize this parameter is using 1 and decreasing it in 1 order of magnitude until an acceptable result is achieved, a larger β will achieve smaller output weights. As a final note, it is really important to seed the random number generator when testing for optimal parameters to make sure the changes in the error are caused by the changed parameter and not by the variations in the initial state of the reservoir.

2.3.2 Conceptors

Conceptors are a computational principle discovered in (Jaeger, 2014) that emulates the interaction between higher neural activities with lower level ones. Conceptors are a neuro-computational mechanism that integrate multiple cognitive systems allowing a single neural network to learn multiple dynamical patterns and regenerate them for later use. Whenever a RNN is excited by a dynamical pattern, a characteristic region from its neural state space is populated. A conceptor represents a pattern that can make the RNN repopulate that same region, essentially acting as a neural filter that only lets the excited regions through, regenerating the pattern without the need of external stimulation.

The conceptors intend to act as a bridge between the top-down (”higher” concepts such as reasoning or concept representation) and bottom-up (”low-level” concepts such as activation of neurons with sensor data and motor commands) oriented research of cognitive functions. This process is realized in three main steps: learning a dynamic, storing it, and regenerating it.

The first step deals with the bottom-up approach to neural dynamics, a RNN is trained using the ESN method shown in the last section. This time the reservoir weight matrix is referred to as W^*. After being driven by an input, the resulting state can be represented as a point cloud in the state space of the reservoir. The shape of this

point cloud can be captured by an N-dimensional ellipsoid whose main axes are the principal components of the states of the reservoir. This ellipsoid, called conceptor C, is a geometrical representation of the regularized correlation matrix R of the state cloud and can be expressed as $C = R(R + \alpha^{-2}I)^{-1}$ where I is the identity matrix and $\alpha \geq 0$ is a design parameter called *aperture*. This aperture can be interpreted as a scaling of the reservoir data resembling the proportional scaling of light energy that reaches the film based on the aperture of a camera lens.

The second step deals with storing or *loading* the conceptor in the weight matrix of the reservoir. This step is required so that the reservoir can later recall the patterns it has learn. Here a new weight matrix W is calculated such that $Wx^j(n) \approx W^* x^j(n) + W^{in} p^j(n)$ for all times n and all patterns j. Computing W can be seen as a simple regularized linear regression, however this representation of the dynamics in the reservoir is inherently messy and unstable, another step is required to have a complete conceptor.

The final step deals with the top-down approach where a conceptor is used to drive the pattern from the loaded reservoir. After a conceptor $C^j = C(p^j, \alpha)$ has been derived from a sample pattern and loaded into the reservoir by calculating W, the loaded reservoir is able to regenerate the target pattern with no input by $x(n+1) = tanh(Wx(n))$, this is also called *self prediction* (Mayer and Browne, 2004). However, if multiple patterns have been loaded to the reservoir, the dynamics become unpredictable and the output is unable to regenerate any of them. The conceptor is used in this step to act as a neural filter that will derive the driving pattern from the messy output of the update equation. The new update rule becomes $x(n+1) = C^j tanh(Wx(n))$. By observing the reservoir with the previously trained output weights W^{out} the pattern will be closely regenerated.

These conceptor appear to be useful for robot control of repetitive movements, such as walking, considering that motor commands in these kinds of cyclic motion will generate a signal that closely resembles a time series. The matrix representation of conceptors makes them very flexible and applicable to many different situations, they can theoretically be applied whenever a RNN is present. This work demonstrates one of the possible uses for conceptors in robotics in a limited fashion, the experiments show one possible research branch that could be explored in the pursuit of an autonomous robot capable of adaptation and learning new movement options based on its own configuration and its surrounding environment. More specific details about the implementation of conceptors on this work can be found on section 3.4.

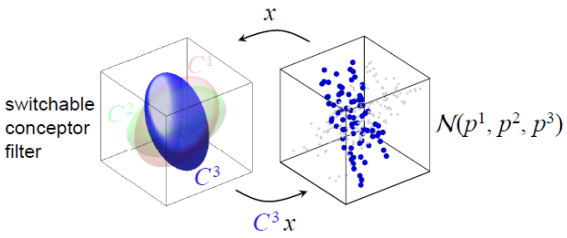

Figure 2.3: Representation of the use of conceptors, as the reservoir is driven by different patterns p^j the activation state of the neurons is represented by a point cloud. Applying a conceptor C will filter the point cloud and regenerate the pattern without the need of a driving input (Jaeger, 2014)

2.4 Reinforcement Learning

Reinforcement learning (Sutton and Barto, 1998) is based on the idea behind an agent's interaction with its environment. It is through this interaction that the agent will learn to take actions depending on the situation, in order to maximize an arbitrary numerical reward signal. The agent is not explicitly told what actions should be taken by an explicit teacher, rather the agent learns through a process of trial and error and measuring the reward obtained. This learning method tries to emulate nature in a sense where organisms learn through the discovery of their own reward functions, if a child gets hurt by touching fire it will try to avoid it the next time, if an animal finds a food source it will keep coming back whenever it is hungry. This way an agent can learn the basics of cause and effect in a sense, eventually generating optimal behaviour that maximizes said reward over time.

To define reinforcement learning from the context of a computational approach one must characterize the problem to be tackled. This problem has to be one of a goal-oriented kind, through its action the agent must change the state of the environment in order to reach one or multiple goal states. In the reinforcement learning problem an agent must have a way of incorporating these three aspects of state, action, and goal. Any method that has way to have a notion of the state of the environment and being able to take actions that affect this state of the environment that will eventually lead you to a goal characterized by the reward signal can be classified as reinforcement learning.

As the objective of the agent is to learn an optimal policy that maps states to actions in order to maximize the long time reward, one of the main challenges of reinforcement learning is the balance between exploring the state space and taking the optimal actions

based on the agent's current knowledge. This exploration versus exploitation dilemma is one of the main issues in developing reinforcement learning algorithms and is still a major topic of research. This can prevent the application of reinforcement learning to larger tasks but can be addressed if optimality is not necessary to complete the task. The size of the search space tends to be a big problem for reinforcement learning algorithms, the use of function approximation Doya (2000) was developed for such cases.

2.5 Conclusion

This chapter presented the main concepts that were used throughout the project. The focus of the work is in the use of self-organization together with conceptors to direct the exploration of a reinforcement learning algorithm in a robot, this will allow the robot to learn a goal oriented behaviour in a reasonable amount of time. This kind of reinforcement learning has been demonstrated to work in (Martius, 2010) through the use of the concept of *behavioural primitives* (see section 3.5), the use of conceptors allows a different approach to the process of acquiring and storing them.

One of the most important tools used in this work is the use of conceptors to streamline the process of acquiring these behavioural primitives. As shown in (Jaeger, 2014) these can be used to store and recall complex signals such as motion capture data, then used to drive the movement of a wire frame model, which brings to mind the movement of a robot. The use of a homeokinetic controller to develop complex behaviours has been proven before, however these are not oriented towards any particular goal (other than reducing the TLE), but with the help of conceptors and reinforcement learning a robot can effectively use this controller to direct those behaviours towards a more specific goal.

The next chapter explores in a more detailed view the problem to be tackled by this project, the use of self-organization to drive a robot so that it can learn goal oriented behaviour together with conceptors. The hypothesis and the approach taken to explore a possible solution are also presented based on the concepts introduced in this chapter.

Chapter 3

Hypothesis and Approach

3.1 Introduction

In a practical context it is impossible to prepare an mobile robot to deal with every issue it might encounter in a changing environment. The robot has to be able to adapt to changes and unforeseen circumstances for it to truly be autonomous. Due to the capacity to deal with the interaction between an agent and its environment, the use of reinforcement learning comes to mind to make a robot broaden its skills to fit a specific goal. However, in a high-dimensional space such as a robot these learning algorithms don't scale well due to the rapid growth of the search space. Take for example the hexapod robot presented in section 3.3.1, with three joints in each leg and six legs even if there were as few as three different choices per Degree of Freedom the search space would be $3^{18} > 10^8$ actions. A more compact representation such as behavioural primitives is needed to use these kinds of algorithms.

The use of self-organizing algorithms in a pre-training phase during a reinforcement learning paradigm has been shown to have a correlation towards a higher reward for high-dimensional tasks and a faster learning time in low-dimensional tasks (Smith and Herrmann, 2013). In this work this approach is used together with conceptors to direct the exploration on the behavioural space of a robot in order for it to learn a new skill autonomously. Learning new skills for a self-organizing robot can be incredibly time consuming and ultimately impossible due to the size of the search space, generally due to the curse of dimensionality. It is unreasonable to expect the robot to find a useful behaviour by proving at the search space randomly. The need for directed exploration to reduce the search space is present if the robot is to be more autonomous. A similar technique was introduced in (Martius, 2010) where the use of the home-

okinetic controller allowed the acquisition of multiple behaviours where feed-forward neural networks were used as expert controllers for low dimensional robots such as a four wheeled robot and a spherical robot with three degrees of freedom. In this work this approach is tested with the use of RNNs and connceptors instead, also applied to a higher dimensional hexapod robot to test its limitations.

In this chapter the methodology used in this work is presented. Section 3.2 addresses the hypothesis of this research in a formal manner. Section 3.3 presents the main software and hardware used for the simulations and experiments in the project. Finally, section 3.4 presents the design of the algorithm to be used in those experiments, mainly the acquisition of behavioural primitives through the use of conceptors.

3.2 Hypothesis

The main goal of this dissertation is to prove the use of conceptors as a way to capture robot behaviours. In (Martius, 2010) feed-forward neural networks were used in a competing experts schema, where the design had to be fine tuned carefully to ensure a proper behaviour of the controller. In this work the use of a conceptor network is proposed to simplify the process of acquiring behaviours.

By the use of a homeokinetic controller a robot is able to generate coherent behaviours that can be stored in a RNN through the use of conceptors. This conceptor network can later be used as a stand alone controller to be implemented towards a more specific goal by a reinforcement learning method. The RNN will be able to drive the robot faster in a more coherent behaviour than the homeokinetic controller by itself.

3.3 Experimental Environment

The LpzRobots simulator (Der and Martius, 2010) was used to test the homeokinetic controller on various types of robots in a realistic setting. A stand alone Matlab program (ran with version R2009a) was also developed to test the performance of the ESNs and conceptors as time series generators. The Matlab code provided a nice environment to test the capabilities of ESNs and conceptors. This was also used to determine the importance of the parameters to the successful implementation of the networks by analysing the NRMSE in various configurations.

Even though the transition from simulator to physical robot is rarely a smooth process, the simulator allows testing the validity of the methods used as it is a powerful

tool for low cost experimentation and provides a lot more flexibility than testing directly in hardware. Another important feature is the inclusion of a matrix library for C++ that facilitates the use of important linear algebra operations and the implementation of conceptors.

3.3.1 LpzRobots Simulator

The LpzRobots simulator provides the necessary framework to create simulations using either pre-existing robots and controllers, or to create new ones. A tutorial is provided in the installation package where a robot has to be generated from scratch, see Fig 3.1. This, together with the different robot and controller templates provided can aid in the development of complex simulations even if one is not familiar with the C++ interface.

This simulator was developed with the concept of self-organization in mind. The controllers created with the simulator are dynamic and adaptable to different robot configurations. The algorithms of the self-organizing controllers can work around the physical limitations of a real hardware environment, even though the behaviours found might not be exactly the same, the theory behind the methods should still work.

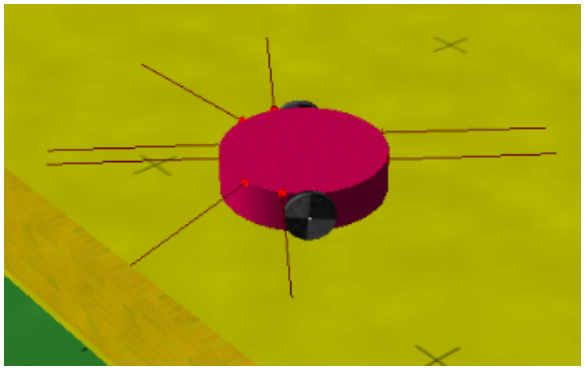

Figure 3.1: The simulator provides a flexible environment for reasearch of robot behaviours. Multiple robots can be created like this differential drive robot from the tutorial provided within the simulation files, multiple distance sensors where added to provide information to a simple obstacle avoidance controller

3.3.1.1 Spherical Robot

The spherical robot has a special configuration that mixes simple controllers with complimentary body dynamics to produce complex movement. The robot contains only

three linear actuators, each actuator is an axis orthogonal to the other two and has a mass that can be moved linearly. Due to the shift in its centre of mass and the shape of its body, the robot will roll until it stabilizes. The direction of the roll depends on the orientation of the axis that moved its mass. By oscillating the mass in synchrony with the roll the robot can be driven at different speeds. Due to the interaction between the three actuators, the robot can follow complex trajectories like driving in circles or in a zig-zag pattern.

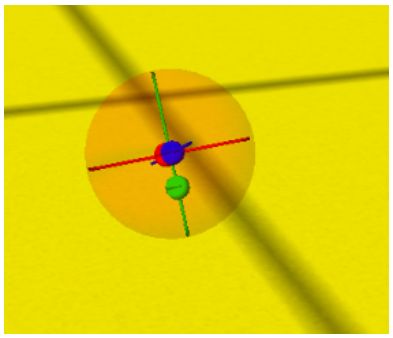

Figure 3.2: Spherical Robot with three masses that can be controlled linearly through their respective axis

3.3.1.2 Hexapod Robot

The hexapod robot is a model provided that is more closely related to a robot that could be used in real life situations. The robot consists of a main body and six legs, each leg is made of three links connected by three rotational joints, one that connects the leg to the body and the other two join the links between them, the two antennae at the front are not used in this work but can act as sensors. The robot can develop complex movement such as walking because the links that connect the robot can move both up to down and back to front. To make sure the robot does not fall over or accidentally turns upside down the links are capped at a 45 degree movement.

3.3.2 Computer Hardware

All the experiments for this work were realized with a laptop equipped with an Intel Core i7-4700MQ @ 2.4 GHz processor and a Nvidia GeForce GT 755M graphics card. This should be considered if tests with physical robots are to be performed. Generally on-board processing power in robots tends to be limited, with this in mind the methods used within this work are kept to a low computational cost when possible. It is also

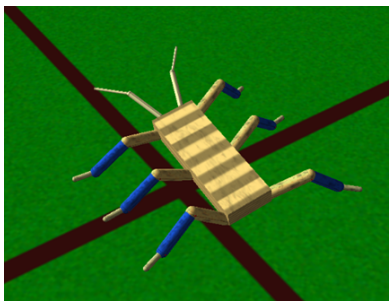

Figure 3.3: Hexapod Robot with three joints for each leg

important to note that for the simulator to run smoothly a dedicated graphics unit is recommended, but the simulations can be run in a low end computer at lower speed with the only issue being that the real time taken for the simulations would be longer, as the internal clock of the simulator can be sped up or slowed down.

3.4 Definition of Experiments

The first experiments to be realized for this work are time series generation with ESNs and conceptors. The Echo-state approach described in section 2.3.1 is used to train a RNN with multiple sample signals. The method to generate conceptors for this RNN is presented in (Jaeger, 2014). After driving the reservoir with each of the training data the correlation matrix of the reservoir states R^j is estimated as $M^j(M^j)^T/L$ where L is the length of the training vector and $M^j \in \mathbb{R}^{N_x \times L}$ is the collection matrix of state activations of the reservoir. This correlation matrix will later be used to calculate the main conceptor matrix C^j called projector. The output weights W^{out} can then be calculated through ridge regression. After training, the new reservoir weight W is adapted by linear regression such that $\tanh(Wx^j(n)) \approx \tanh(W^* x^j(n) + W^{in} p^j(n))$. The final step is to calculate the projector matrix C^j, for this the eigenvectors of the correlation matrix U^j are needed so the singular value decomposition of R^j is calculated, $SVD(R^j) = U^j \Sigma^j (U^j)^T$. The projector is then calculated as $C^j = U^j S (U^j)^T$ where $S = \Sigma(\Sigma + \alpha^{-2} I)^{-1}$, α is the aperture of the conceptor, and $I \in \mathbb{R}^{N_x}$ is the identity matrix.

After the projector matrix is generated tests can be performed with the driven reservoir in a generative mode where $x(n+1) = C^j \tanh(x(n) + W^{bias})$. The output then can be observed by the previously trained output weight matrix $y = W^{out} x$. This will allow the network to regenerate the patterns from the reservoir with a single set of train-

ing weights and a new matrix for the reservoir weights that is no longer generated at random.

3.4.1 Acquisition of Behavioural Primitives with Conceptors

The key in understanding the behaviours generated by the homeokinetic controller is in the value of the TLE. A framework similar to the one used in (Martius, 2010) is proposed, see figure 3.4. In this framework, the homeokinetic controller is run in parallel to an expert network consisting of multiple feed-forward neural networks that learn based on a winner take all scenario. Instead of the expert network, the use of a single conceptor network is proposed. One of the main advantages to this approach is that the conceptors can easily merge behaviours together and regenerate them later, it is still important to pre-define the number of conceptors to be used.

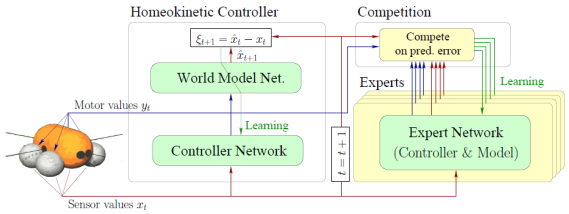

Figure 3.4: Architecture proposed by (Martius, 2010)

3.5 Conclusion

In this chapter an approach towards the use of conceptors as a robot controller is presented. The behaviours developed by the hoemokinetic controller can be stored in a RNN, this RNN can then be used as a separate controller applicable to a more specific goal oriented behaviour. Conceptors can capture behaviours that can then be used in reinforcement learning schema, even if the behaviours are not optimal, if the robot is able to develop an interesting skill through self-organization it can be used a pseudo states to gain positive reward in few trials.

In the next chapter the results of the experiments realized throughout the project are presented. There are two main phases to these experiments, first the use of RNNs to generate time series and second the application of the homeokinetic controller.

Chapter 4

Evaluation and Discussion of Results

4.1 Introduction

In Chapter 3 we presented the basis of the experiments to be used in this work. Most of the work relies on the use of conceptors to capture time series and the use of the homeokinetic controller to drive the robots.

This chapter presents the results obtained by the experiments. Section 4.3 examines Echo State Networks and conceptors. Section 4.3 analyses the behaviours generated by the homeokinetic controller on the spherical robot and the hexapod. in section 4.4 the ESNs were used as a controller for the spherical robot. Finally section 4.5 summarizes the findings.

4.2 Conceptor Network Applications

4.2.1 Echo State Networks for Time Series Learning

To determine the capabilities of the RNN and find suitable parameters for the conceptors, the first experiments were training an ESN with different driving patterns and testing the results with NRMSE. Multiple sine waves, one with high frequency (HF) and another one with low frequency (LF,) and a chaotic time series (specifically the Mackey Glass (MG) Equation, see equation 4.1, with standard parameters, time constant $\tau = 17$, $\beta = 0.2$, $\gamma = 0.1$, and $n = 10$) as well as a weighted sum of the HF sine wave and the MG equation were used.

$$\frac{dx}{dt} = \beta \frac{x_{t-\tau}}{1 + x_{t-\tau}^{n}} - \gamma x. \tag{4.1}$$

The first test of the ESN was performed with a low frequency sine wave, see figure 4.1. The parameters for this test were chosen arbitrarily, based on (Lukoševičius, 2012) (this tutorial provided optimization for a specific test set of the Mackey Glass Equation), with a reservoir of 1000 neurons, a relatively small regularization parameter $\beta = 10^{-8}$ for the ridge regression, a spectral radius $\rho = 1.25$ (typically not recommended because the echo state property is not guaranteed), and a leaking rate $\alpha = 0.3$. As can be seen, the performance was not comparable to the results presented in (Jaeger, 2001) due to the poorly chosen parameters for this kind of driving pattern. One of the most important parameters for these kinds of simple time series turned out to be the regularization parameter for the ridge regression (*NRMSE* dropped from 0.18681 to 0.0025353 by increasing the parameter from 10^{-8} to 10^{-4}), the larger this parameter is, the smaller the trained output weights will become. This means that the states of the reservoir will not saturate as easily working more like a smooth transition rather than an on-off switch. The performance of the MG equation however, did not change much (*NRMSE* increased from 1.1921 to 1.2193).

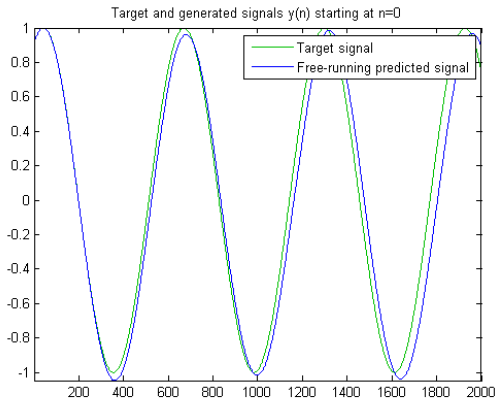

Figure 4.1: Echo State Network trained with a low frequency sine wave, parameters were not optimized and the performance was relatively poor for such a simple pattern ($NRMSE = 0.18681$).

As the MG equation has proven to be the more problematic for the ESN to reproduce, the training parameters were optimized with the equation in mind. Data was separated in three blocks, 4000 samples for training data, 2000 samples for test data, and 800 samples for the initial run off, this is needed to allow the reservoir to forget the state of the arbitrary initialization.

To test different configurations of the network a simple experiment was performed with different connections to the output. The results are presented in figure 4.5.

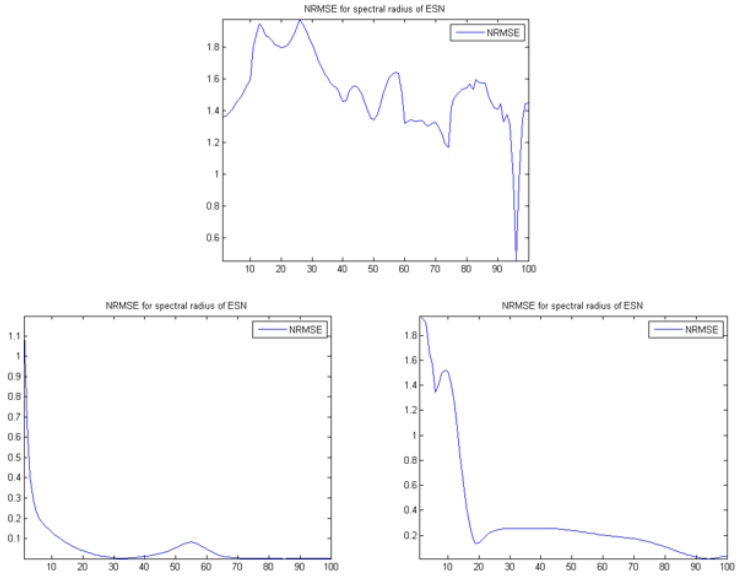

Figure 4.2: The spectral radius was tested on the MG equation (top), HF sine wave ($left$), and LF sine wave ($right$). As can be seen a larger spectral radius clearly performs better for the sine waves. The MG equation has a small working window as it approaches 1, the data seems to indicate that a spectral radius larger than 1 might be the optimal, however for the scope of this work $\rho(W) = 0.96$ was chosen as it performs well enough as a spectral radius larger than 1 does not ensure the echo property of the network.

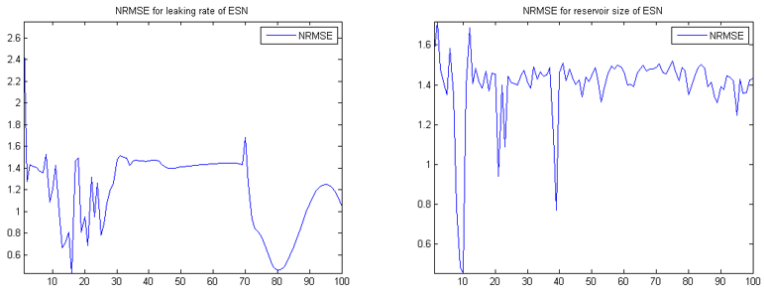

Figure 4.3: For the leaking rate two areas where performance for the MG equation is optimal can be seen, one around $\alpha \approx 0.3$ and the other one around $\alpha \approx 0.8$, 0.8 was chosen as a higher leaking rate provides better memory of previous inputs, which can be useful in a controller setting. Finally the size of the reservoir was tested. For the MG equation this parameter does not seem to have an important effect as the smaller error was obtained with the training reservoir size of 100.

After all the parameters were defined (leaking rate $\alpha = 0.8$, spectral radius $\rho(W) = 0.96$, reservoir size 100, and regularization parameter $\beta = 10^{-4}$) the network was tested

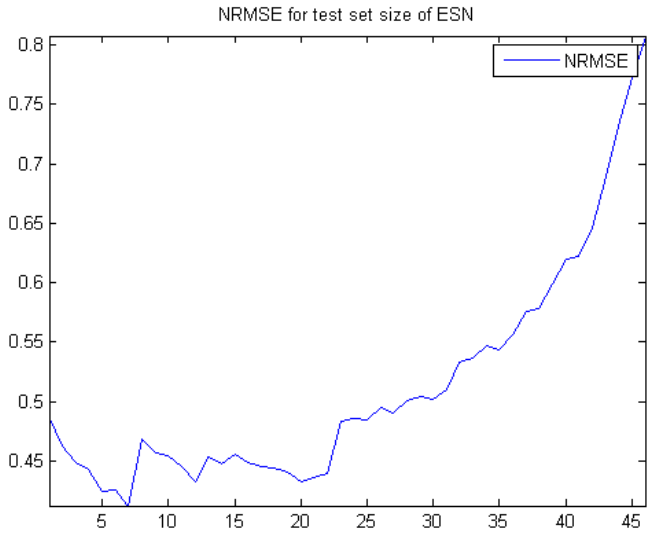

Figure 4.4: To test the performance of the final parameters selected the error of the MG equation was calculated against test sets of various sizes. Even after 5000 steps in the time series, the network performance is still acceptable with an $NRMSE < 1$.

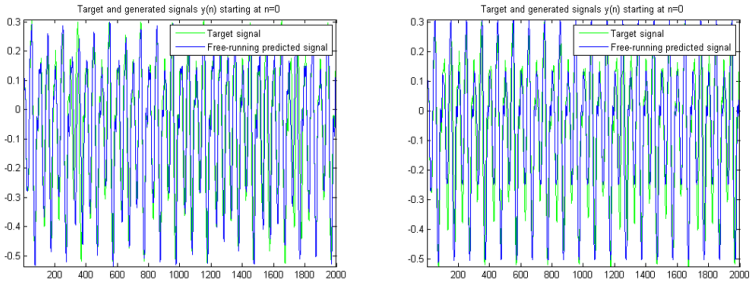

Figure 4.5: Effects of a direct connection between the input neuron and the output ($left$) against no connection ($right$).Even thought the overall performance of the network is similar in both cases ($NRMSE_{connection} = 0.5059$ and $NRMSE_{noConnection} = 0.4557$) but the direct connection creates a more noticeable chaotic behaviour, when there is no connection the network will not follow the small jumps in the target but the phase of the output matches the input more closely, which explains the lower error.

against the four sample signals. No direct connection was used between input or extra bias unit to output as the conceptor network will generate the desired signals without an explicit input neuron. The results of the performance of the network are shown in figure 4.6.

As a final remark of the ESNs, all this tests were performed in what is called the *generative* mode of the network. In the testing phase only the initial state $x(n)$ of the

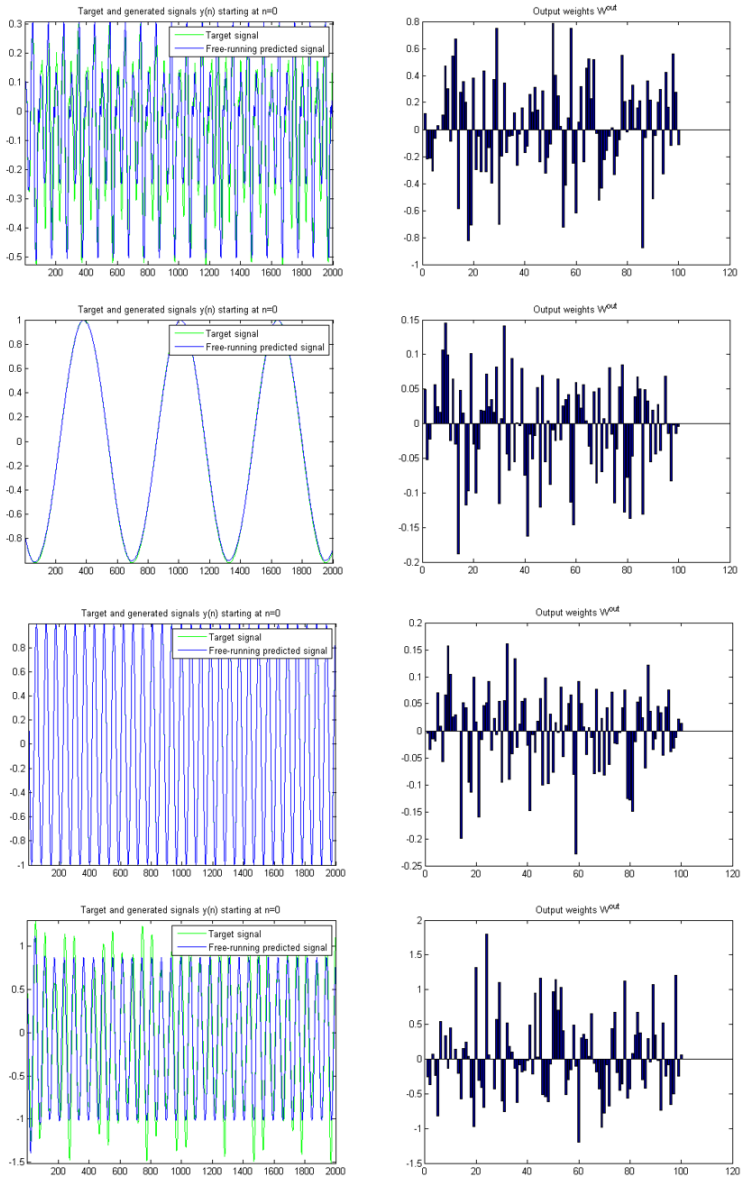

Figure 4.6: The final constants of the ESN where tested with all the sample training signals. From top to bottom the MG equation ($NRMSE = 0.4557$), the LF sine wave ($NRMSE = 0.0157$), the HF sine wave ($NRMSE = 0.0021$), and the weighted sum of HF sine wave and MG equation ($NRMSE = 0.3787$).

reservoir is provided as the input and the ESN uses the prediction $x(n+1)$ as the next input. In the *predictive* mode just predicts the next step in the time series, as expected the performance of this mode is comparatively a lot better ($NRMSE = 0.0026$ for the MG equation) since the prediction error does not accumulate, however this mode is not useful as a robot controller where no test data is available.

4.2.2 Computing Conceptor Matrix of Test Patterns

To test the performance of the conceptors the only other parameter to be optimized is the aperture. First the effects of the aperture are analysed for the LF sine wave in figure 4.7. Frequency appears to be one of the most important characteristics for aperture adaptation, the lower the frequency of the target signal the larger the aperture should be.

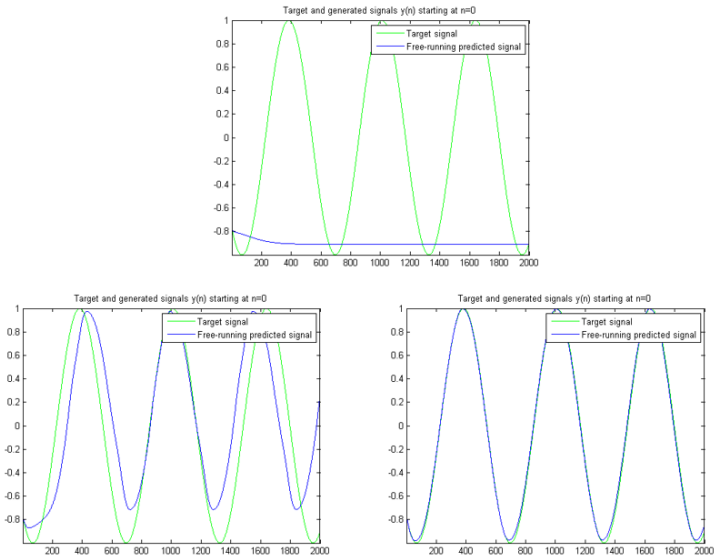

Figure 4.7: The period of the original target signal appears to be a defining factor for the performance of the conceptor with respect to its aperture. In this lower frequency example the aperture was tested with three values small ($\alpha = 250$ *top*), medium ($\alpha = 1000$ *left*), and large ($\alpha = 10000$ *right*) values where the conceptor improved performance significantly for larger values ($NRMSE_{small} = 2.2413$, $NRMSE_{medium} = 0.6093$, and $NRMSE_{large} = 0.0346$).

One of the main issues while adapting the aperture is the large range of values that it can take, the only limitation is $\alpha > 0$. For the purpose of the test signals used in this work, an aperture adaptation procedure was realized by testing the performance of a single conceptor that learned only one signal with and aperture range of $\alpha \in [1, 1000]$, see figure 4.8.

One interesting characteristic about conceptors is the ability to drive the reservoir state from an arbitrary initial state, see figure 4.9. Conceptors work as attractors in the state space of the reservoir, one problem that could arise is if the initial state of the reservoir is in a state space not affected by the attractor, a way to deal with this

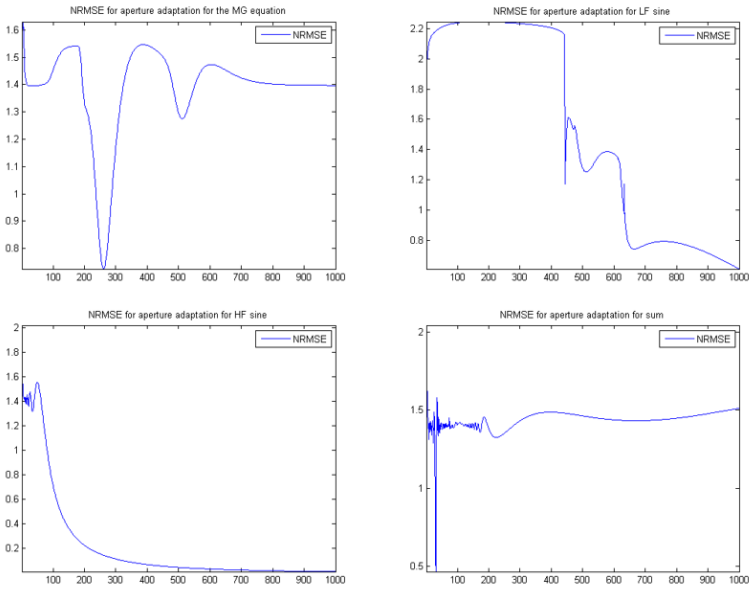

Figure 4.8: Aperture adaptation for four target signals. Top left is for the MG equation ($NRMSE = 0.7254, \alpha = 261$), top right is for the Low Frequency sine wave ($NRMSE = 0.6093, \alpha = 1000$), bottom left is for the High Frequency sine wave ($NRMSE = 0.0059, \alpha = 1000$), and bottom right is the weighted sum of the MG equation and the HF sine wave ($NRMSE = 0.4656, \alpha = 31$). It is interesting to note how the effect of the aperture is different for each time series, in the case of the weighted sum being a really small window of reproduction, with the MG equation the window is a lot larger and the changes in value of the NRMSE are smooth as the aperture changes. For the sine waves a trend to better reproduce the signal is observed as the aperture grows, the error for the LF sine drops significantly after an aperture of 100 and for the HF sine the error drops after 650.

issue is one possible research branch for conceptors, passing through an intermediate transition state might be necessary.

As a final experiment for conceptors, the four sample signals were used to drive the RNN and the new matrix W was calculated, see figure 4.10. As presented before, big differences in frequency of the original driving signals can reduce the performance of the conceptors while regenerating the original patterns from the new reservoir states.

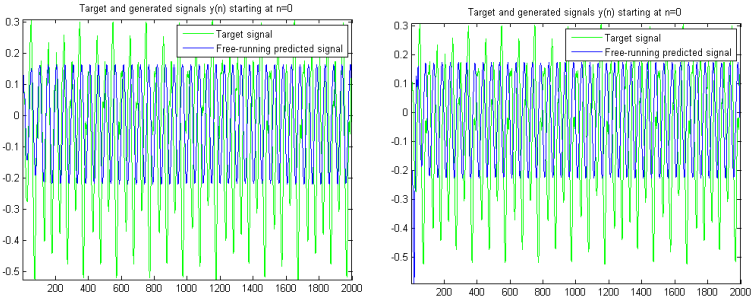

Figure 4.9: Generation of pattern by reservoir driven by the MG conceptor after aperture adaptation. The conceptor has trouble generating the chaotic behaviour of the pattern but it's still good enough to get a NRMSE below one, as seen in the aperture adaptation. Note how the conceptor can drive the reservoir from an arbitrary starting point, in the first case the next value of the reservoir state $x(n)$ after training was used, while in the second graph an arbitrary starting point for the reservoir states of $x(n)$ as the zero vector was used.

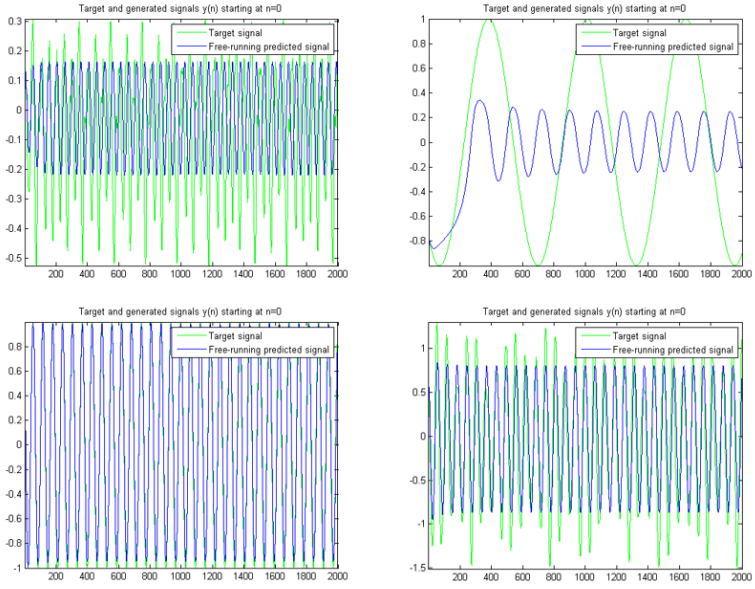

Figure 4.10: The order remains the same as in figure 4.8, the same network was driven with all the four test patterns at the same time. Performance in the generative process of the network decreased significantly with the optimal apertures found by the adaptation ($NRMSE_{MG} = 1.5569$, $NRMSE_{LF} = 1.2578$, $NRMSE_{HF} = 0.0770$, $NRMSE_{sum} = 0.5451$). Not surprisingly the network has trouble regenerating the LF sine wave due to the large difference in frequency. The high error for the MG equation is because of a phase shift caused by the mixed reservoir states, the sine element of the weighted sum mitigated this effect.

4.3 Homeokinesis for Robots

4.3.1 Behaviour of Spherical Robot

As presented in (Martius, 2010), the spherical robot is able to develop a coherent behaviour fairly quickly. Within the first 20 seconds of simulation the robot has already started rolling with a stable signal from the controller. The shape of the Time Loop Error, see figure 4.11, however, is never stable. At the beginning the TLE starts small, but as the robot is driven to exploration it can quickly increase, see figures 4.17 and 4.13.

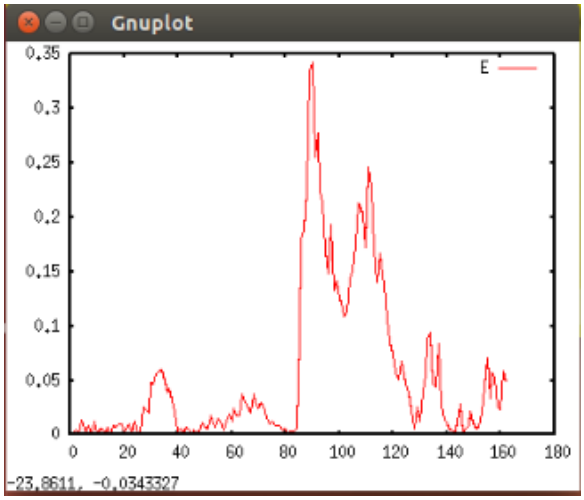

Figure 4.11: Time Loop Error at the beginning of the experiment.

Generally, the robot rotates around an axis, where the other two synchronize to generate the roll and the rotation axis develops small oscillations that stabilize the robot. Whenever the average TLE increases, the robot switches behaviours, see figure 4.17. When the three masses oscillate at the same time, see figure 4.13, the robot follows circular trajectories.

4.3.2 Behaviour of Hexapod Robot

The behaviour of the hexapod is a lot less stable than the spherical robot. Even though individual legs can coordinate their movement, in general the robot as a whole will not move in any particular direction. At the beginning the robot will move its body and try to stand up while the TLE increases. After while the robot is able to stand and move individual legs around, sometimes taking small steps forward or backwards. In the next figures the main two motors of the first leg were measured for better clarity.

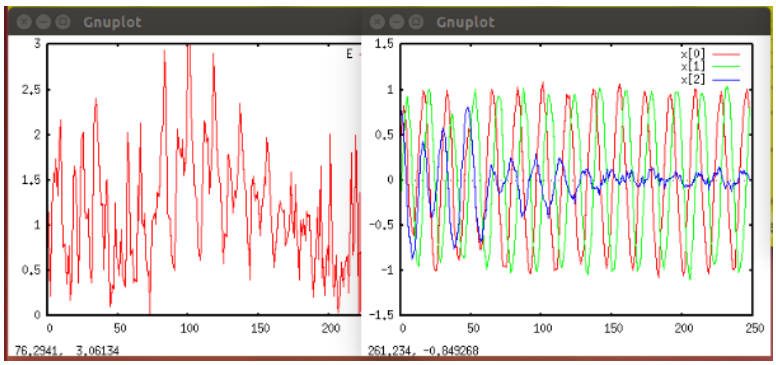

Figure 4.12: A new behaviour is developed. The average shape of the TLE increases and the blue actuator decrease the amplitude of oscillations.

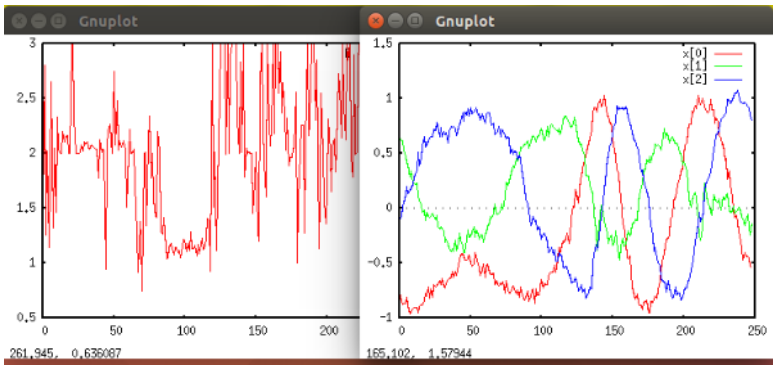

Figure 4.13: After a while, the sphere develops slower oscillations. The TLE by this point has increased from less than 1 to osculate around 2.

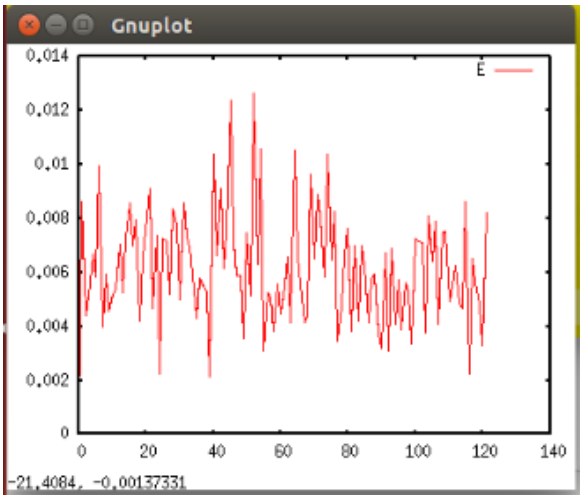

Figure 4.14: Osculations of the TLE for the hexapod robot start closer to 0 than with the spherical robot. Due to lack of body driven dynamics, the sensor values remain stable for longer periods of time and the robot has a harder time increasing its exploration.

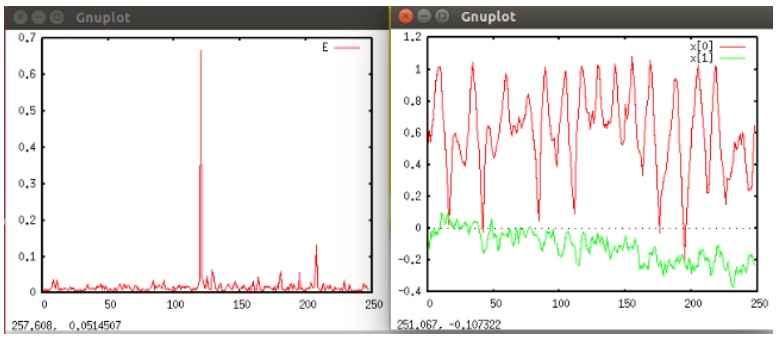

Figure 4.15: Spikes in the TLE do not switch the behaviour on the robot while the overall magnitude of the error remains small.

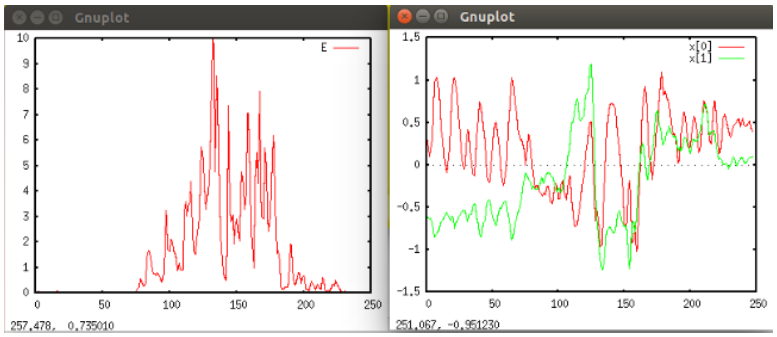

Figure 4.16: The TLE reaches larger numbers than with the spherical robot. As the average TLE increases, both leg motors switch behaviours.

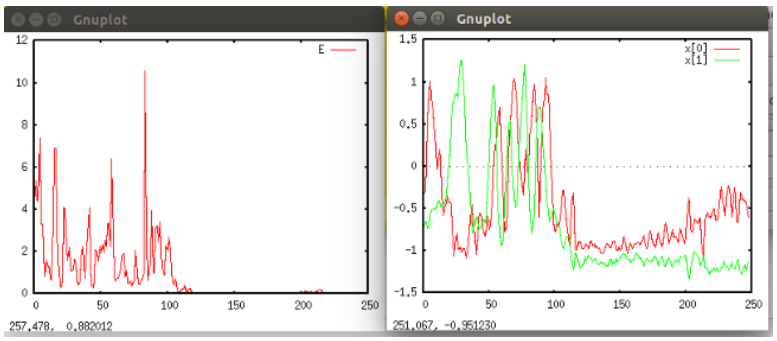

Figure 4.17: When the average TLE decreases significantly in value a shift in the dynamics of the leg is observed

4.4 Performance of RNN Controller

An ESN was applied as a controller to the spherical robot. In this experiment there was no competition between multiple experts, so the network could only learn one behaviour. After some time, the homeokinesis was switched with the ESN as a controller. The robot was able to maintain a higher speed than with the normal homeokinetic con-

troller, this behaviour was developed by the network learning multiple dynamics at the same time, however the network is able to maintain a stable throughout the rest of the simulation.

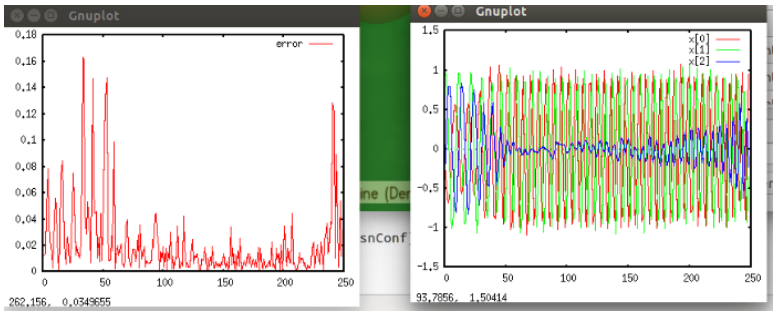

Figure 4.18: At initialization the actuators of the robot quickly develop a faster oscillation than previously observed with the pure homeokinetic controller.

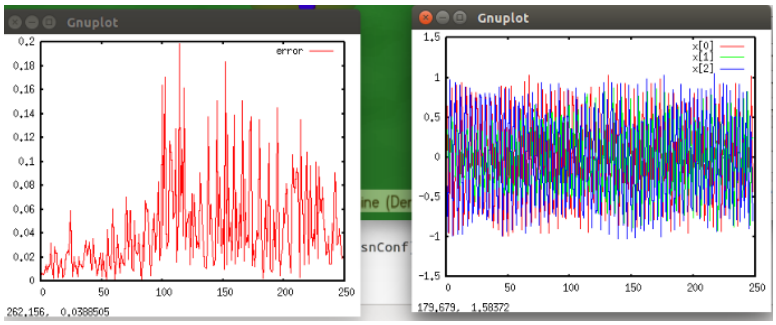

Figure 4.19: Once the behaviour has stabilized, the ESN can maintain the dynamics for long periods of time. The TLE is now oscillating in values closer to 0 than with the homeokinetic controller.

4.5 Conclusion

In this chapter the results of the experiments are presented. The conceptor network was used in time series generation as a stand alone Matlab code to demonstrate its performance. The homeokinetic controller was applied to the spherical robot and to the hexapod robot, and their behaviour observed to define the final scope of the experiment.

The conceptor network, however, was not ready to be used as a controller for the deadline, however some interesting behaviours were observed when a single ESN was used with the spherical robot.

Chapter 5

Conclusion

This work presents a way towards the use of conceptors in a self-organizing framework for robot control. This appears to be a promising approach for robots to learn and develop skills in an adaptive manner, where both the physical aspect of the robot and its interaction with the environment are utilized.

The use of ESNs for time series generation was tested thoroughly. Conceptors were also used and some interesting characteristics were observed. The importance of similar frequencies for the same reservoir was determined as the network could not reproduce a simple sine wave that had been learned previously with small error.

As shown in chapter 3, the use of behavioural primitives through RNNs is able to drive a spherical robot faster than the pure homeokinetic controller, however there are still large question marks left to answer. So far only one ESN has been able to be applied to the controller, switching behaviours with a conceptor schema is yet to be tested. The competition between behaviours has to be tested in order to fully prove the ESN implementation.

5.1 Possible Future Work

Due to time constraints, the scope of the project had to be reduced significantly. The hexapod robot was not able to produce useful behaviours so a method such as the one presented in (Smith and Herrmann, 2013) can be used to improve the behaviour of the homeokinetic controller. The controller used for the ESN has to be improved by mixing multiple behaviours in order to derive the conceptors. When multiple behaviours are stored one can use reinforcement learning to achieve more interesting goals than increasing the speed of a spherical robot.

Another obvious extension of the work is the implementation on physical hardware. Implementing conceptors in a more flexible robot, such as a humanoid, should also be explored.

Bibliography

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Borges, R. V., d'Avila Garcez, A., and Lamb, L. C. (2011). Learning and representing temporal knowledge in recurrent networks. *Neural Networks, IEEE Transactions on*, 22(12):2409–2421.

De Raedt, L., Hammer, B., Hitzler, P., Maass, W., De Raedt, L., Hammer, B., Hitzler, P., and Maass, W. (2008). 08041 abstracts collection–recurrent neural networks-models, capacities, and applications. In *Recurrent Neural Networks*.

Der, R. and Martius, G. (2010). Playful machines.

Der, R., Martius, G., and Pfeifer, R. (2012). *The Playful Machine: Theoretical Foundation and Practical Realization of Self-Organizing Robots*, volume 15. Springer Science & Business Media.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245.

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

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik.

Jaeger, H. (2014). Controlling recurrent neural networks by conceptors. *arXiv preprint arXiv:1403.3369*.

Lukoševičius, M. (2012). A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade*, pages 659–686. Springer.

Martius, G. (2010). *Goal-oriented control of self-organizing behavior in autonomous robots*. PhD thesis, PhD thesis, Georg-August-Universität Göttingen.

Martius, G. and Herrmann, J. M. (2012). Variants of guided self-organization for robot control. *Theory in Biosciences*, 131(3):129–137.

Martius, G., Herrmann, J. M., and Der, R. (2007). Guided self-organisation for autonomous robot development. In *Advances in Artificial Life*, pages 766–775. Springer.

Mayer, N. M. and Browne, M. (2004). Echo state networks and self-prediction. In *Biologically Inspired Approaches to Advanced Information Technology*, pages 40–48. Springer.

Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *J. Artif. Intell. Res. (JAIR)*, 11:241–276.

Smith, S. C. and Herrmann, J. M. (2012). Homeokinetic reinforcement learning. In *Partially Supervised Learning*, pages 82–91. Springer.

Smith, S. C. and Herrmann, J. M. (2013). Self-organisation of generic policies in reinforcement learning. In *Advances in Artificial Life, ECAL*, volume 12, pages 641–648.

Spong, M. W., Hutchinson, S., and Vidyasagar, M. (2006). *Robot modeling and control*, volume 3. Wiley New York.

Strauss, T., Wustlich, W., and Labahn, R. (2012). Design strategies for weight matrices of echo state networks. *Neural computation*, 24(12):3246–3276.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Szita, I., Gyenes, V., and Lőrincz, A. (2006). Reinforcement learning with echo state networks. In *Artificial Neural Networks–ICANN 2006*, pages 830–839. Springer.