University of Edinburgh
School of Informatics

Optimising Robotic Kicking Motions Through Reinforcement Learning

5th Year Project Report
Master of Informatics

Seán Wilson, s0831408

April 4, 2013

Abstract: Learning actions, to complete a task successfully and optimally, is a time consuming and potentially computationally expensive task. Reinforcement learning (RL) has previously been applied to humanoid robotics for optimising actions, revealing its potential to take robotics a step closer to being truly autonomous and versatile. The following research presents the first application of RL to develop a fully optimised kicking motion for the Alderbaran Nao robot. Optimising for a multi-objective reward-function produced stable, accurate and fast kicking motions, able to kick further than all current RoboCup teams. Through the implementation of a dynamic kicking module, we provide evidence that the optimised kicking motions can be adapted to the different ball locations and produce angled kicks. The research and approach in this paper, lends itself to future research in different kicking motions and more complex search-spaces.

Acknowledgements

I would like the thank Prof. Subramanian Ramamoorthy for his constant guidance and inspiration. I would also like to thank Prof. Robert Fisher, Prof. Barbara Webb, Dr. Michael Rovatsos, and Dr. Taku Komura for their excellent teaching and infectious enthusiasm. Finally, I would like to thank Black for being a resilient and tolerant test subject.

Contents

1. Introduction

1.1 Advances in Robotics

Motors are decreasing in size and power consumption. Power sources are becoming lighter, smaller, and producing more power for longer periods of time. As these forms of technology improve, so does our ability to design more flexible robots for more complex tasks [1]. Robots have transformed from the early sensor-driven machines such as W. Walter's 'Tortoise' [2], reacting to the environment through basic voltage changes from sensors, to the latest iteration of Honda's Asimo with advanced image recognition and environmental analysis [3].

In 2012, Asimo demonstrated being able to run, hop on one leg, unscrew a bottle, and pour the content into a glass. These complex actions require a large number of motors in each limb to provide the required dexterity and movement. The number of directions of movement available to a joint is known as degrees of freedom (DoF). By increasing the number of motors and DoF in the robot, it becomes increasingly complex to calculate the required joint movements to perform these actions. Manually coding and testing the voltage and torque levels required to perform such actions would be a time consuming process. Inverse kinematics has become a universal solution to manipulating joints to move end-effectors, such as robotic hands and feet. But the calculation of where and how to move the end-effectors to complete a task, must still be answered. This can be answered by turning to the field of artificial intelligence, specifically learning.

1.2 Learning in Robotics

Learning methodologies have been widely used in the field of robotics research to suitably refine a solution, given a set of actions and objectives in a testing environment. Supervised learning is the predominant form of learning used in robotics. Supervised learning is a method in which alternative, and potentially superior, solutions are sought by utilising training data which defines example

solutions to the problem. But this requires a sample solution in the first place.

An alternative learning method is that of reinforcement learning [4, 5]. Species are often regarded as 'intelligent' when they achieve a goal by learning from experience. An interactive experience often produces a measurable conclusion that can be described as positive or negative. Whether this be through enjoyment level, or nervous system responses of pain and pleasure. This is the fundamental idea of reinforcement learning. By imitating how organic species learn, we can remove the need for training data and optimise actions by giving a simple numerical reward for successful interactions.

1.3 Aims and Objectives

Development over the past decade, in all disciplines that comprise mechatronics [6], have lead to a decrease in the price of robotic equipment and software, allowing companies such as Alderbaran to produce mass-manufactured 'personal' robots [7]. As such, many institutions are now able to purchase these sophisticated robots for academic research. In this paper we use one such robot to perform real world testing, and understand what it takes to manoeuvre a real humanoid robot. Specifically, we wish to learn optimal motions for kicking a ball through trials and feedback reward alone.

This paper is an example of how a complex robotic motion task can be analysed and broken down into the key properties required to obtain optimal solutions through reinforcement learning. We explore the application of reinforcement learning in the field of robotics, and its effectiveness to find balanced, optimal solutions to a multi-objective problem. We discuss why temporal-difference reinforcement learning methods best suit robotic motion, and why Q-learning specifically is chosen as the method used in this paper. We discuss the problems faced when balancing the priorities of multiple objectives in reinforcement learning. We show that providing solutions based on humanoid anatomy and our own experience, may not always be the best solution in a robotics context, but serve well as the basis for defining the motions.

The current kicking engine in use by the Edinferno team [8] is a single distance

straight kick, which requires the robot to get into position behind the ball facing the desired direction. We wish to develop, through reinforcement learning, basic straight kicking motions for multiple distances, optimised for stability, speed, and power. Following this, we aim to build a module which dynamically modifies the motions to produce angled kicks. This module will use three variables: the position of the ball relative to the foot, the desired distance, and the desired angle. These could be supplied by the behavioural module if utilised by the Edinferno team.

This research aims to produce a dynamic kicking engine, with optimised kicking motions for set distances. We wish to accomplish this aim through two clear objectives:

- Using reinforcement learning, produce policies that define kicking motions that are optimised for distance, speed, stability, and accuracy.
- Develop a module capable of modifying the kicking motions, to adapt to the ball and goal locations.

This research is the first attempt at finding a fully optimised kicking motion using reinforcement learning. Previous research [9] has studied only the movement necessary to angle a kick using reinforcement learning. The results show conclusive evidence that reinforcement learning is a viable tool for learning optimised kicking motions through experience on an actual robot.

1.4 Hypothesis

In this paper we explore how reinforcement learning can be used to produce an optimal kicking motion, given limited information about the environment and possible actions. The reward function will combine rewards for multiple objectives: the distance the ball travels, robot stability, execution time, and angular accuracy. With this multi-objective reward function, it is expected that the final policies will satisfy all of the constraints of an optimal kick.

Q-learning reinforcement learning has been widely tested and proven to converge faster than other temporal-difference or Monte Carlo methods [5], and as such

will be the RL method used. It is expected to find an optimal solution within 100 episodes for a small search-space, and within 200 episodes for a more detailed environment.

Using reinforcement learning, it is expected that the resulting kick will not resemble a human kick, with a large backswing, but instead have a small backswing only to give the motors the potential to reach full speed. The accuracy of the kick, in terms of deviation from the intended trajectory of the ball, is calculated outside of the reinforcement learning. It is not expected to be affected by different kicking motions, as each motion will travel through the vector from the ball to the goal position. The accuracy may, however, be affected by the momentum of the kicking motion. It may also be affected by the pitch environment, such as uneven patches. It is also expected that using the top of the foot will produce a more accurate kicking angle, but at the expense of distance. This is implemented and discussed further in section 6.

1.5 Outline

In the following sections, we discuss related work, the breakdown of the problem, and evaluate the effectiveness of reinforcement learning for the problem at hand. In section 2, we define the problem and key concepts that are used in this paper. We move on, in section 3, to evaluate and discuss the kicking motion implementations employed by other RoboCup teams, and related implementations of reinforcement learning to similar humanoid motions. Before testing, we must first evaluate the limitations of the Alderbaran Nao robot and discover how simulators can be used to safely test robotic motion. Using the key findings from the related work, we begin to refine the problem into the components common to all reinforcement learning problems, and carry out preliminary testing of reinforcement learning's application (section 4). In section 5, we discuss the implementation of a reinforcement learning algorithm and reward-function that finds the key-frames that define an optimal kicking motion. In sections 6 we evaluate the application of reinforcement learning, and how the problem can be defined. Future work (8) could include increasing the complexity of the environment, and using physics simulators to further improve the kicking motion.

2. Background

2.1 Context

The RoboCup standard platform league (SPL) [10, 11] is the domain in which reinforcement learning will be applied, specifically to learning an optimal kicking motion for the Alderbaran Nao humanoid robot. RoboCup is an annual robotic football competition, with the goal of developing an autonomous humanoid robot team capable of beating the human world champion football team by 2050. Robocup is an applicable international research initiative, allowing global collaboration to advance the fields of robotics and artificial intelligence. SPL matches are played between two teams of 4 Aldebaran Nao robots (as seen in Figure 2.1), on a 6 by 4 metre pitch, with similar rules to human football [12] (see Appendix A).

RoboCup objectives are the same as in human football: the robot must score points by kicking the ball into the opponent's goal. As such, the robot needs a kicking module capable of applying the necessary force to the ball to place it at a targeted location. Throughout this paper the 'goal' of a kick does not necessarily refer to the physical goal, between the upright posts. The kicking engine is intended to be used to kick the ball to a specific point on the field (location determined by the behaviour module). This will allow for potential passing to team members during a game as well as taking shots at the football goal. This is something that has been tested [13, Ch. 4.a], but not utilised successfully by a team in RoboCup at the time of writing. The primary behaviour, when kicking, has been to advance the ball down the pitch. Teamwork is the greatest advantage that human football teams can obtain to defeat an equally able opponent, and this principle should aim to be applied in RoboCup.

The Nao, specific to RoboCup, has 21 degrees of freedom, 11 of which are located in the legs [7], as depicted in Figure (2.2). The Nao robot is therefore able to reproduce a similar kicking motion to that of a human, although this may not be the most effective action for a humanoid robot. Human muscle is elastic, and as such produces a large force by being stretched and released quickly, as

Figure 2.1: RoboCup specific Nao (left), and Acedemic H25 Nao (right)

demonstrated by a McKibben Pneumatic Artificial Muscle [14]. Nao use brushed direct current (DC) motors [7], a type of servo motor, which does not produce elastic energy. It is expected that this will affect the optimal motion implemented through reinforcement learning. Servo motors have a distinguishing advantage over muscle - consistent accuracy. A servo motor is able to accurately replicate the same actions with each execution, due to the electromagnetic stepper design [15]. This enables the user to accurately specify joint angles. Accuracy is crucial in robotics, which is why servo motors are widely used actuators [7, 15].

2.2 Moving End-effectors

To move an end-effector, such as a robotic hand or foot, we must first move all of the joints that join the end-effector to the root node. In humanoid robotics, the root node is the hip joint for the legs, and the shoulder joint for the hands. There are two main methods for moving the end-effectors: forward and reverse kinematics. Forward kinematics is finding the position of the end-effector from the joint angles in the chain of joints, starting at the base. Forward kinematics has only a single solution, namely the joint angle readings.

Inverse kinematics is a higher-level problem solving approach. It transforms the

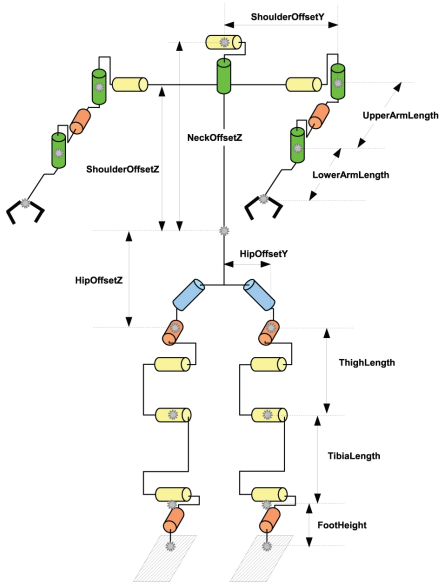

Figure 2.2: Model of the Nao joint motors, demonstrating degrees of freedom [16]

motion plan (the movement required to position the end-effector to the desired location) into joint trajectories for each affected actuator. Starting with the joint nearest the end-point, the joint is moved closer to the target location. This is followed by the movement of the subsequent joints in the chain, until either the target location is reached or all joints make a parallel trajectory through the location, in which case the location is unreachable [17]. The inverse kinematics problem is defined as calculating the value of each joint in a chain of joints, in order to move the specified end-effector or limb to a given location in space. Mechanical characteristics of the joints in the chain, such as available degrees of freedom (DoF), can make the calculation complex and often raise difficulties in obtaining a solution [18]. Inverse kinematics can have many solutions, depending on how much each joint is moved per iteration of the algorithm. Inverse kinematics is used more often in daily robotics, as we normally wish to describe a desired location of the end-effector. In general, the chain of joints from a central node can be represented as a kinematic chain, with rigid links and joints between links termed as kinematic pairs. The kinematic pairs model hinge joints in the robot. For humanoid robots, most hinge joints have 3 DoF (tilt, swivel, and pivot).

The solution to the inverse kinematics problem used by Alderbaran [19] is calculated from 2.1, where: $\dot{X}$ is the derivative of Nao's end-effector position $X = [x, y, z, \theta x, \theta y, \theta z]$ with respect to time, q is the position of all the joints, $q = [q_1, ..., q_n]$, and J^{-1} is the inverse Jacobian matrix of q, $J(q)$.

$$\dot{q} = J^{-1}\dot{X} \tag{2.1}$$

Another important aspect used frequently in robotic motion is interpolation. Given a set of key-frames, we must interpolate the trajectory that an end-effector should take between them, by constructing new points using some function. The simplest interpolation method is linear interpolation. Using equation 2.2 we are able to interpolate the position of y for x, between two given points (x_0, y_0) and (x_1, y_1). Linear interpolation is useful for direct movement between two points, but when the points are part of a longer chain of movements, it results in jerky movements.

$$y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0} \tag{2.2}$$

Spline interpolation is a solution to ensuring smooth transitions between a chain of positions. In robotic motion, the spline curve most commonly used is the Bézier curve [20]. A Bézier curve in 3D space is defined by four control points as in equation 2.3. P_0 and P_3 are the Euclidean vectors from the first key-frame to the next. P_1 and P_2 describe the directions of the curve and t describes the distance between P_0 and P_3, (for example, $t = 0.75$ would be the point three quarters of the way to P_3). Two Bézier splines can be joined by point P_3 of curve one and P_0 of curve two. Let us call the control points of curve two (Q_0, Q_1, Q_2, Q_3). If we ensure the distance P_2 and P_3 is equal to distance Q_3 and Q_1, and $P_3 = Q_0$, we can ensure a smooth transition between the curves [21].

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)^2 P_2 + t^3 P_3,\ t\,\epsilon\,[0,1] \tag{2.3}$$

The static key-frame implementation of joint angles, described in [22], is the most common method of describing joint motion, and is implemented by most Robocup teams [23–25]. This method gives no flexibility for the correction of

angles or trajectory, but as discussed in [26], dynamic solutions do not always produce the best results in the given time frame. It is sometimes beneficial to simply get the ball moving in the correct direction rather than being intercepted by the opponent. A scripted approach requires the robot's body to be perfectly positioned for the calculated line of shot, which in most cases is not achievable in the time frame available. Once a scripted key-frame kick is initiated, the robot is unable to react to external disturbances to itself or the ball, which could affect the accuracy of the kick.

2.3 Alderbaran Software

Since the introduction of the Nao robot, Alderbaran have been improving their NAOqi control framework, which allows cross-platform and language communication with any Nao robot [27]. The framework allows for communication between the different modules such as vision and motion, and allows for homogeneous programming between the supported languages and platforms. Both C++ and Python are supported and share an identical API, allowing users to create modules that can be used in either programming language across different operating systems. As can be seen in Figure 2.3, sourced from [27], NAOqi is a broker which provides access to the directory service, and provides network access for the robot. The directory service contains all required modules to use the robot's sensors and actuators. We take advantage of the NAOqi framework for a number of reasons. Firstly, all methods in the directory service are tested and implemented by the manufacturer, specifically for the robot. As such, we can be guaranteed that the modules will work correctly and efficiently. Secondly, the NAOqi framework establishes network connection to any wireless, or wired, network and provides a user-friendly interface for monitoring the robot's hardware. This will be useful for preventing damage to the robot while testing. Thirdly, the NAOqi framework connects to other Alderbaran software, such as their NaoSim physics simulator which we will use to test motions before applying them to the robot. Fourth, and finally, NAOqi is currently used by the Edinferno team to communicate with the robot.

The specific modules that we will be using from the NAOqi directory are the

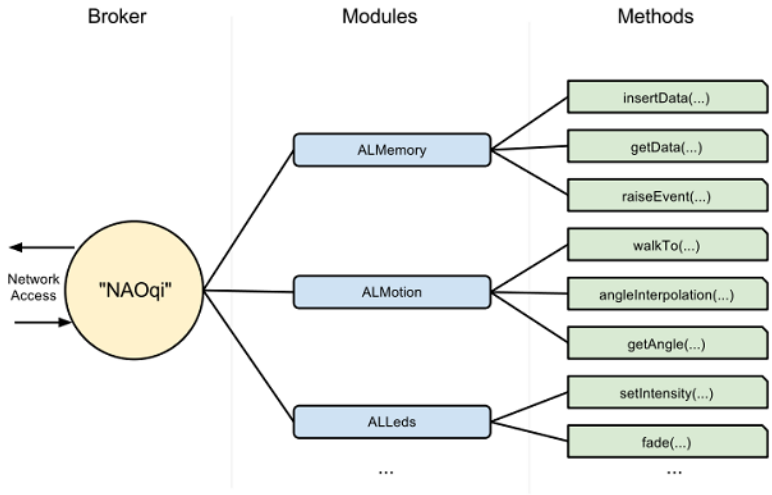

Figure 2.3: A network diagram describing the Naoqi communication framework

sensor feedback modules, and the motion modules to interpolate movements. We create proxy objects, such as ALMotion in Figure 2.3, which behave as the modules containing the methods we require. The NAOqi ALMotion library has two motion modules that allow for end-effector specific movement. Given the desired Cartesian location of an end-effector, the joint angles and motor speeds are calculated automatically using inverse kinematics and either linear or Bézier interpolation. Alderbaran uses Bézier splines for interpolation, which automatically calculate control points P_1 and P_2, given two Cartesian locations (used as control points P_0 and P_1) and the interpolation time between them.

2.4 Reinforcement Learning

Reinforcement learning (RL) is the process of learning how to map situations to actions, to maximise a numerical reward [5]. It is defined by characterizing a learning problem instead of a learning method, as in supervised learning. RL problems can be split into two parts: the learning agent which makes decisions to maximise its reward, and the environment which is comprised of everything outside of the agent. We describe the environment at a moment in time as s, a state. In the current state the learning agent can perform a set of actions which will change the environment, and effectively move the agent into another state,

describing the changed environment. In the problem described in this paper, the robot, the ball, the pitch, and the rewarding process are the objects that make up the environment. The learning and decision process make up the parts of the agent.

A learning agent is told "what needs to be done" (a problem), as opposed to "how to do it" (a method). Thus, given a problem such as moving a ball to a certain location, the agent must find a solution to moving the ball through a set of actions. The advantage of RL is being able to proceed with optimising a problem without any prior solutions. We call the mapping of probabilities to the selection of actions a policy (π). Each policy describes the probability of executing a set of actions in the environment in an attempt to gain a reward. RL is a method of trial-and-error with the actions and states of each episode receiving a numerical reward depending on the outcome; the agent tries to maximise this reward. The agent must decide in each state whether to exploit the best known action (the one that yields the largest reward), or to explore other actions in the hope of receiving a larger reward through lesser explored states. As an agent cannot exploit and explore simultaneously, a careful balance must be struck between the two (discussed further in sections 5.5). We formally define taking an action a in a state s at time t under a policy π as (2.4).

$$\pi(s_t, a_t), \quad \text{for all } s_t \epsilon S,\ a_t \epsilon A \tag{2.4}$$

Some problems have a natural terminating state. We call these problems episodic tasks, as the learning process can be split into episodes or trials of experience. Each state acts as a signal of the environment's state, and as such, must relate the relevant information of the state's desirability back to the agent. The state should provide direct readings, such as sensory measurements, but should also provide enough information to meet the Markov property. The Markov property implies that all relevant information to realise a state must be retained in the description. This is defined formally for a reinforcement learning problem in 2.5.

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\}, \tag{2.5}$$

where t is the time at which an action was taken, leading to state s' and immediate reward r. As each state holds the Markov property, $s_t = s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0$, meaning that s_t retains all information describing past events. We require the Markov property from all states to be able to predict the next state, and expected reward, given the current state and action. The values and decisions made by the reinforcement learning agent can only be assumed as functions of the current state, and so the current state must be representative and informative [5].

As states in a reinforcement learning task satisfy the Markov property, the task itself can be seen as a Markov Decision Process (MDP). Episodeic tasks, such as the one in this paper, are called finite MDPs. We now formally define the transition probability of going to state s', given current state and action, s and a, as 2.6. We can also define the expected reward from the transition to s' from s via action a as 2.7.

$$P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s,\ a_t = a\} \tag{2.6}$$

$$R^a_{ss'} = E\{r_{t+1} | s_t = s,\ a_t = a,\ s_{t+1} = s'\} \tag{2.7}$$

The reward function, as previously mentioned, must be part of the environment as feedback given to the agent. The reward function computes a real-value numeric reward that must represent the desirability of the outcome from the agent's actions. How the reward is given to the states, and to what states, depends on the reinforcement learning method used.With the states and rewards defined, we now move on to the main components of the decision making agent: the value function.

2.4.1 Value Functions

Reinforcement learning is based on estimating the value function; functions that define how good it is for the agent to be in a state, or how good it is to preform a given action in a given state. The value of a state represents the accumulated reward that can be expected from being in the state and following the current policy π. We can define the state-value (V^π) function and action-value function (Q^π) while following policy π formally as 2.8 and 2.9 respectively.

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \tag{2.8}$$

$$Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s,\ a_t = a\} = E\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s,\ a_t = a\right\}, \tag{2.9}$$

where E is the expected reward that is to be accumulated over the episode following π. γ is the discount rate between $0 \leq \gamma \leq 1$, which can be changed to make future rewards have a stronger effect the closer it gets to 1. The state-value retains the average of all the visits to the state from any action. The action-value holds the specific value of taking action a in state s. The value functions are estimated through experience and are updated through policy-iteration. A fundamental property of the value function is that they satisfy a recursive relationship between each state and successor. That is, that the value function must solve some form of the Bellman equation 2.10, given the reinforcement learning method used.

$$V^{\pi}(s) = \sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{2.10}$$

2.4.2 Exploration and Exploitation

One of the main concepts of reinforcement learning is striking the correct balance between exploration and exploitation, as they cannot happen simultaneously. If an agent explores continuously, there is little that separates the learning task from an exhaustive state-space search, which is computationally complex, and in most cases infeasible. As a result, it is beneficial for the agent to sometimes exploit known 'good' states. A good state is a state with a known value that is better than others. As the value of a state is the expected future reward, exploiting states with a higher value leads to a known solution with some reward. Exploiting and exploring in an episode leads to learning the returns of different policies other than the greedy policy (π^*) - the policy that exploits the state with the largest expected reward at every time-step. In doing so, the agent may find a policy with a higher return. In situations when a policy π is found to have a

higher expected return than the current greedy policy π^*, it is fair to say that this becomes the new greedy policy.

An optimal policy is said to be the policy in which all states under the policy have a value greater than, or equal to, all other states in the state-value function 2.11, or action-value function 2.12.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \text{ for all } s \epsilon S \tag{2.11}$$

$$Q^*(s,a) = \max_{\pi} V^{\pi}(s,a) \text{ for all } s \epsilon S \text{ and } a \epsilon A(s) \tag{2.12}$$

It is these optimal policies that we wish to obtain through reinforcement learning. There are many techniques to balance the trade-off between exploration and exploitation. These are discussed further in section 5.5.

2.4.3 General Policy Iteration

General Policy iteration (GPI) contains two processes that work simultaneously to improve the current policy of a problem. The first is policy evaluation, which computes the value function for the current policy. Depending on the reinforcement learning method used, this could be a search of the whole state-table (dynamic programming), or could be through an experience (Monte Carlo). The second process is policy improvement, which changes the greedy policy in respect to the value function. This could also be a sweep through the entire state-space (dynamic programming), or from following each greedy state-action pair in the state-space and returning the resulting change of state-action pairs (Monte Carlo). We discuss which tasks each policy iteration method is suited to in the following section. By interacting the policy and the value function in such a way, moving between one and the other in an iterative manner, we move both towards their optimal values.

2.4.4 Reinforcement Learning Method Classes

Policy-search reinforcement learning learns a direct mapping from states to actions. These techniques have been successfully studied while optimising parame-

ters for the Sony Aibo [28, 29], the previous RoboCup SPL robot. On the other hand, value-function based reinforcement learning [5] holds an intermediate data structure that maps states (or state-action pairs) to the long term reward, that can be expected for taking a specific action in the current state. Value-function learning methods provide a semantic representation of an optimal policy. It is for this reason that we focus on value-function reinforcement learning methods.

RL methods can be split into three classes [30], each with different strengths and weaknesses in tackling different learning problems: dynamic programming, Monte Carlo, and temporal-difference. It is important when using RL to understand which class of methods would best suit the problem at hand by evaluating the complexity of the learning problem's environment, and how well the problem is understood.

The classes can be seen on a scale representing knowledge of the environment. Dynamic programming methods can be used when the environment is thoroughly understood and can be perfectly modelled as MDP [5]. Dynamic programming uses the model of the environment to bootstrap; estimating the value of the current state from the value of others. Modelling the dynamics of the environment and iteratively updating all states in the state-value update process can be computationally expensive to calculate, depending on the complexity of the problem. Dynamic programming suffers from the curse of dimensionality; states grow exponentially with the number of state variables [5, 31]. It is also not always possible to calculate the outcome of all actions and their resulting states with enough accuracy for a model. For example, in situations where actions affect the physical world with some degree of uncertainty, and the resulting state is some set of stochastic possibilities.

Monte Carlo (MC) methods, on the other hand, do not need a model; instead they learn from experience. These methods are suitable when the complete dynamics of the environment are unknown and therefore a complete model is not possible. MC does not bootstrap, reducing the computational requirements of policy-iteration. MC updates all states in the episode, which allows an episode that leads to a bad state to be impacted immediately; in doing so, this disregards valuable information about neighbouring states which could have already been gathered. Monte Carlo is a method for solving the prediction problem; using GPI to drive

the value function to accurately predict the return for the current policy, while improving the policy with respect to the current value-function. Monte Carlo methods have been shown to converge to an optimal policy in quadratic time. However, due to their value update approach, they converge more slowly than the other prediction problem solving methods such as temporal-difference [5, Ch.6.3 pp. 141].

Temporal-difference methods are the most commonly used today. They use the advantages of bootstrapping from dynamic programming, but without the need for a model of the environment. Like Monte Carlo, temporal-difference methods learn from experience to solve the prediction problem of GPI. Unlike Monte Carlo, temporal-difference does not wait until the final return before updating the value of states. Instead, temporal-difference methods update their value function by partially moving a state's value towards that of their successor state as they are visited in an episode; this is know as the one-step prediction. This update process can be done either on-line or off-line with regards to the policy being followed. On-line policy improvement iterates over the current policy being followed, updating the action-values towards the successor as defined by the current policy 2.13. Whereas off-line policy improvement updates the action-values towards the next best state, as defined by the greedy policy 2.14.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} = \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.13}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} = \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.14}$$

A model of the environment can be considered as any information that the agent can use to predict how the environment will respond to its actions [5, Ch. 9, pp. 227]. Model-free methods, while computationally cheaper, ignore the dynamics of the environment in the agent's decision making process, requiring more experience before an optimal policy can be found. In model-based reinforcement learning, an agent uses its experience to construct a representation of the control dynamics of its environment. It can then predict the outcome of its actions and make decisions that maximise its learning and task performance. Model-based reinforcement learning methods are also known as planning methods, whereas model-free methods are known as learning methods. We see later, in section

3.4.1, how models can be used to find an optimal policy in a limited number of iterations. In this paper, we are not concerned with the time taken to approach an optimal policy, only that we reach one. Value-function based methods are theoretically guaranteed to converge to an optimal policy eventually [32].

3. Related Work

3.1 Overview

This section will explore the implementations of previous kicking engines for the Alderbaran Nao robot. We also explore the application of reinforcement learning in penalty kicks and walking engines, to gain a better understanding of which methodologies can be utilised to optimise a kicking motion. In particular, we focus on the most effective implementation elements of various RoboCup SPL teams. These elements will be used to form the required parameters and features of the reinforcement learning problem in section 5.

3.2 Team Kicking Engines

3.2.1 B-Human

B-Human [25], the winning team of 2011, describe a trajectory based method to dynamically update the trajectory of the kick [23], to compensate for changes in the environment. This is similar to the method used for trajectory-based walking motions in [25]. A motion engine which describes the joint angles of the kick receives a list of dynamic points as inputs, describing the trajectory of the foot. All other joint angles are calculated by inverse kinematics [18]. This approach allows for online updating of the kick trajectory through cubic Bezier curves to ensure a smooth motion (3.1).

$$b(t) = \sum_{i=0}^{3} {}^{3}_{i} t^{i}(1-t)^{t} - iP_{i} \tag{3.1}$$

The kicking motion is divided into sub-motions that are common components to all kicks; Lifting the kicking leg, moving it back, moving towards to ball, follow through, returning to original position, and placing the foot down. These six movements, using Bezier curves [20], describe the trajectory of the foot in the

complete kicking motion. These sub-curves are joined together by the second and first control points to ensuring the connection points are continuously differentiable, making smooth transitions. [23, Ch. 3.1, pp. 112] proves how each sub-curve, except the first, can be described by a single control point Q_1; each other control point, is inherited from the previous curve in the complete motion using equation (3.2), where Q_i denotes control points from the first curve, and P_i from the second.

$$\frac{P_3 - P_2}{\delta t_1} = \frac{Q_1 - P_3}{\delta t_2} \tag{3.2}$$

The curves are generated from control points calculated from parameters, dynamically provided by the robot's behaviour module. The parameters describe the limb to be moved, the target position of the limb, the motion direction, and the sub-curve identifier. With the sub-curve identifier and limb identifier, the kicking module is able to update the corresponding curve's fourth control point, to comply with the new target location. The motion direction rotates the third control point around the fourth control point to change the direction of the kick. These updates to sub-curves can be applied during the motion of a kick, before the sub-curve is executed, allowing truly dynamic motion.

A balance module, based on centre of mass (COM) stabilisation [33] and a sensory feedback closed-loop PID controller, is used to keep the robot from falling during a kicking motion. The COM stabilisation compensates for any unbalance that may occur during the kicking motion itself, but is unable to account for external disturbances; this is compensated for by the closed-loop PID using Nao's gyroscope feedback.

The COM is projected onto the ground, which the stability module attempts to keep in the centre of the balancing foot polygon. The required angle update for the pitch of the balancing leg is calculated from the mean expected shift of the motion. Using a PID controller, Pythagorean theorem (to relate to the true COM), and inverse kinematics, this is realised through the equations in [23, Ch. 4.1, pp. 114]. After the desired angular velocity is calculated for the COM shift, the gyroscope feedback is used to calculate external disturbances. This is also compensated for using the PID controller.

In practice, only the two sub-curves that move the foot forward for the kick are updated online. It was found that the robot was able to walk up to a randomly placed ball and kick with an average deviation of 3.32°, and an angular deviation with a range of $-0.9°$ and 7.54° with 95%. An average distance of $5436mm$ was achieved with standard deviation of $251mm$. This distance improves upon Edinferno's current kicking distance by $250cm$. There is no mention of execution time, but it is clear from this paper that accuracy is the optimised property of the kicking motion. Accuracy is arguably the most important aspect of a kick, but in a dynamic environment such as RoboCup, slowly executed actions can see the ball taken out of the robot's reach by an opposing robot.

In practice the balancing module, tested while the robot balanced on one leg, was able to compensate for an external disturbance in a time of 2.5 seconds. This was enough time for the robot to recover from a force that otherwise was proven to place the robot off balance. The results are inconclusive as to whether the balancing module was able to keep the kicking motion accurate during external disturbances, or whether the stability module increased the accuracy of the dynamic kicking motion.

Splitting a kicking motion into sub-movements, as implemented in this paper, could be used in a reinforcement learning implementation; using each movement as a time-step. As such, if using the same Bézier curve implementation, the states would be the final control point, and the actions the curved interpolation between control points.

Before a conclusion can be made whether the implementation of an online balancing module is effective at correcting a kicking motion, we must first find out whether it is possible to complete a kick in 2.5 seconds, with a distance similar to that of the B-Human kicking module. If this is possible, it suggests that the balancing module is in fact redundant (see section 4.2).

3.2.2 Austin Villa

The winning team of RoboCup 2012, Austin Villa, take an approach to their kicking engine that takes into account the opponent's position before selecting the type of kick. This, they believe, was a contributing factor to their win in

2012 [34]. The kicking engine comprises of two types of kicks: static standing kicks and kicks executed while walking.

The static kick used in 2012 is a faster implementation of the one used in 2011 [35, Ch. 6.1, pp. 12-16]. This is able to kick a distance ranging between 1.5 to 3.5 metres. The distance achieved by the kicking motion is varied by changing the execution time of the kicking motion. The variance amount needed was determined by through running tests at different time interpolations and fitting a linear model to the logarithm of the distance. As with B-Human, the kicking motion is computed by moving the foot through an interpolation spline and using inverse kinematics to move the appropriate joints. The kicking motion is also split into phases, similar to B-Human, but with only a single forward movement. Only the align state and kick state as seen in Figure 3.1 are changed to vary the kick, depending on the ball's location relative to the foot. It is unclear, however, if the team has continued to use directional kicks as implemented for 2011 in [35].

Using B-Human's walking engine [25] as a basis, Austin Villa were able to tune the walking motion to include a kick while still moving. Due to Nao's flat feet and no foot joints, B-Human's walk is inherently stable between steps. This allowed Austin Villa's team to modify the Cartesian movement of the foot mid-step to kick in the desired direction, before placing the foot and continuing. The kicking leg does not adapt to the ball's relative location to the kicking foot, but these kicks were only intended to keep the ball rolling in the desired direction.

There is no mention of execution time, but as the static kick is only executed when the surroundings are clear of opponents, it would seem that execution time was not a priority. Austin Villa do not use a balancing module as B-Human do, and there is no mention of angular accuracy. Therefore we are unable to compare this method in these respects to B-Human's kicking engine design. However, B-Human's kicking motion is able to produce 55.3% more distance than Austin Villa's furthest stationary kick. This indicates that there is still room for improvement for distance in Austin Villa's approach.

B-Human, and section (4.4) of this paper, show that the resulting distance of a kick can deviate as much as $25cm$ with the same kicking motion. This indicates that there maybe little benefit to constant varying distances as opposed to set interval distances.

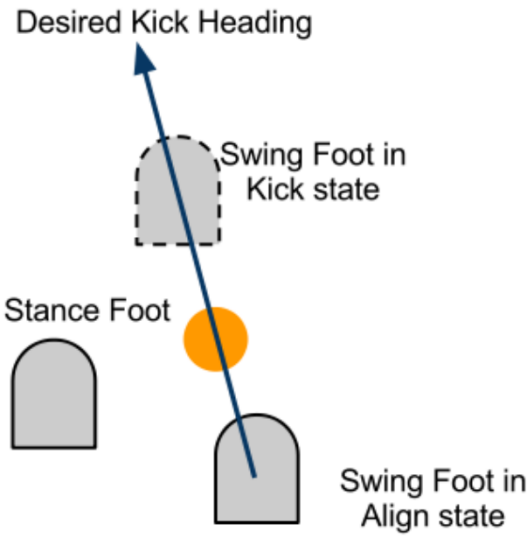

Figure 3.1: From [35], a diagram of the kicking trajectory

Kicking motions are linear forces applied to a ball, perpendicular to the desired direction. Therefore, the movement Austin Villa use in the back and forward position to determine the angle of the shot, could be applied to the respective time-steps in the reinforcement learning method. In this paper, we only look at producing static kicks, but a kicking motion while moving could be implemented using reinforcement learning and is discussed further in section 8.2.

In 2010, a more in depth report was produced on the Austin Villa kicking engine used in RoboCup that year [13]. This paper describes the same implementation as above, but with more detail on when a specific kick is chosen; this is beyond the scope of this paper. The same accuracy and distance measures discussed above are mentioned in [13].

3.2.3 rUNSWift

From their 2010 report [36], it is apparent that Team rUNSWift implemented a similar omni-directional stationary kick as described by [35]; splitting a kick into states and dynamically updating the start and end state of the propulsive phase. Their kicking motion can be dynamically updated to kick a ball that sits between $50mm$ in front of the foot and $120mm$ either side of the foot. Through empirical testing, it was found that moving the foot $30mm$ either side of the

ball's centre would give an angled kick of 45°. This implementation was not used in the RoboCup competition, due to slow execution times and only producing a kicking distance of $400cm$.

Instead, the team used a slow walking kick. The slow walk allowed the robot to pause mid-stride, while maintaining balance, and execute a kicking motion before continuing with the walk. The kicking motion, unlike other teams, is implemented through angle interpolation of the joints to move the foot. The joints of the robot, during the walk, are controlled using sinusoidal interpolation, to give a smooth motion between the kicking and the stepping commands. The walking engine initialises a kick when a power level parameter, supplied by higher level behaviour tasks, greater than 0.01 is provided. This power parameter sets the strength of the kick, thus varying the distance achieved. The strength of the kick was determined empirically, resulting in equation 3.3, where $kickPeriod$ is the execution time in seconds from the start to finish of the propulsive phase. The execution time of moving the foot to the start of the propulsive phase, and returning the foot to centre takes $0.14seconds$ each. Balance is maintained by moving the arms in the opposite direction of the kicking leg, providing counter momentum to the rapidly moving leg.

$$kickPeriod = 0.9 - power * 0.5, \qquad power \; \epsilon \; (0.01, 1.0] \tag{3.3}$$

Figure 3.2 shows the resulting kicks from 13 shots while varying the power level. No quantitative results are supplied, but from Figure 3.2 it is clear that the distribution increases, the further the ball is kicked. The results provided are biased by the balls colliding with each other, making the results even less conclusive. It appears, from knowledge of pitch dimensions (see Appendix A), that the furthest kick in the image is roughly $250cm$.

From rUNSWift's team report, we begin to see a pattern in the way teams represent and divide kicks into phases, or states. Their kicking motion, depending on the power required can take between $1.13seconds$ and $0.68seconds$. These times do not include weight shifting, as these times are not mentioned. Although the kicking motions do not produce large distances, the quick execution times could see the ball stay in the team's possession. If passing is implemented effectively,

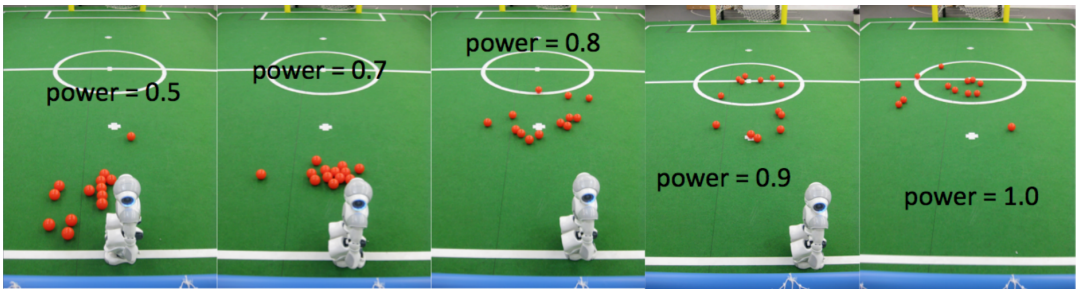

Figure 3.2: Results of different power values of the rUNSWift slow-walk kicking motion

this could result in a more desirable kicking motion. But as a standard kick, on a 6 metre long pitch, it is beneficial to be able to kick long distances. Counter-balancing the robot's momentum shift with the arms is an intuitive and simple solution which should be investigated further.

3.3 Simulator Kicking Motions

An area of RoboCup with much similarity and crossover with the standard platform league, is that of simulated RoboCup. This has been used mainly for multi-agent cooperation research in the past, due to simple high level abstraction of the simulators. But in 2003 SimSpark was proposed [37] as a more realistic, model-based, physics simulator for RoboCup. This proved to provide a more realistic football simulator, able to narrow the gap between simulation and real world actions. However, the simulation of robotic motion is complex to model, and does not always transfer well between simulation and real robot as discussed in [38].

As a result of the advances to model-based simulator competitions, we are able to relate to simulator team implementations of kicking motions. FC Portugal developed an omni-directional kicking engine for their simulated team [39] in Simspark, using similar dynamic phases and Bezier curves as [23]. Inverse kinematics is used to manipulate the angle of the joints in the leg, in order to move the foot to the specified position in the curved trajectory. A balance module is designed to measure the COM of the robot, projected into the polygon area of the balancing leg, and move the arms and hip joint to keep the COM in the centre of the polygon.

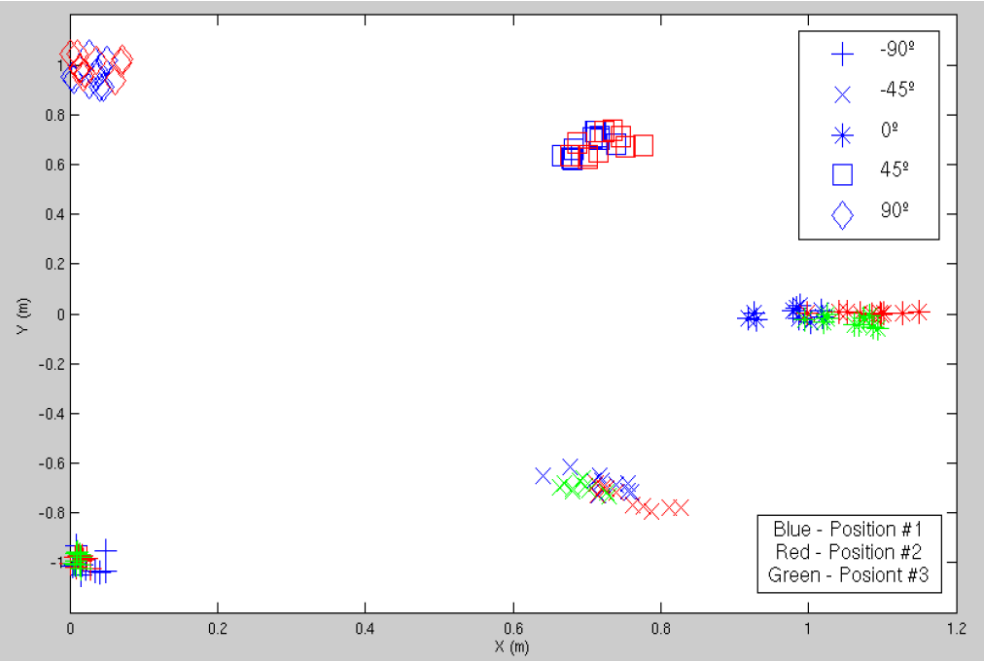

Figure 3.3: 10 kicks at 90°, 45°, and 0°. The robot is positioned at 0 on the y-axis

The simulator test results are shown in Figure 3.3 after a ball was placed at three different positions relative to the kicking foot; position 1 on the inside of the kicking foot, position 2 directly in front of the kicking foot, and position 3 on the outside of the kicking foot (there is no mention of the distance from the foot to the ball). After testing at 90°, 45°, and 0° degrees, the kicking engine kicked the ball successfully with an average of deviation of 1.12°. This proves, at least in simulation, that dynamically updating the kicking trajectory in relation to the ball and kicking leg as proposed by [13, 23, 39] increases the accuracy of a kick, without needing to reposition the robot to directly in front of the ball. But as discussed in [38], simulated testing can only produce estimations of real robot performance.

3.4 Applications of Reinforcement Learning

3.4.1 Penalty Kicks

Reinforcement learning (RL) has been applied to a kicking engine for the Alderbaran Nao robot recently in [9], and has successfully proven the usefulness of RL in quickly optimising a kicking action to score a penalty. The paper compares three different value-function based learning algorithms: model-free Q-Learning, model-based R-MAX and RL with decision trees (RL-DT). In this paper, the problem is defined as a standard Markov Decision Process (see 2.4).

In this paper, Q-Learning [40] was run using ϵ-greedy exploration with a learning rate of $\alpha = 0.3, \epsilon = 0.1$, where α is the amount the value is moved towards its true value (the return). R-MAX explores all unknown states M times to quickly build a model of the environment. The agent is forced to explore all states with less than M visits, as it assumes these states have the maximum reward. Once all states have been visited M times, the optimal policy is find through value iteration. Model-based RL-DT generalises aggressively, during model-learning to reduce the number of required episodes to visit all states. RL-DT generalises rewards across states to learn the model in as few samples as possible, using supervised learning techniques. The algorithm described uses a heuristic to decide when to explore. If the model predicts that only a reward less than 40% of the maximum immediate reward is possible, the agent goes into exploration, choosing the least visited state-action. If the model predicts a reward with greater value, then the agent exploits.

The algorithms were tested by kicking a ball, 1.8m away from a stationary goalkeeper. The robot's state was described as the x-coordinate of the ball and the distance the foot was shifted from the hip. From this state, the agent had three possible actions: Move-in, move-out, and kick. Each movement, in or out, moved the foot $4mm$ in either direction. The robot received a reward of -1 for every move, -20 if the robot fell, +20 if it scored a goal, and -2 if it did not. The experiment was run in Webots simulator and on a physical Nao robot. Each algorithm was tested for 30 trials, 100 episodes per trial in the simulator, but only once with the physical robot and only for RL-DT.

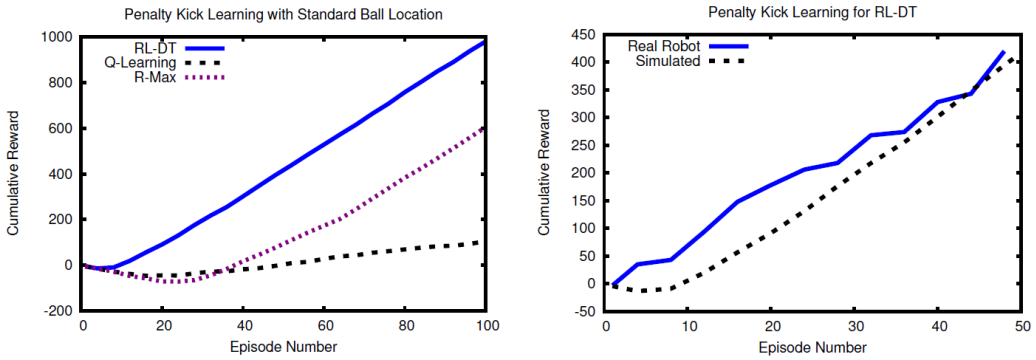

Figure 3.4: (Left)Resulting cumulated reward during penalty kick trials, (right) RL-DT simulator and physical robot comparison

From this paper as seen in Figure 3.4, it is clear that model-based RL learns sufficiently faster than a model-free algorithm in the kicking domain. It was found that out of the 30 trials, R-Max found one of the two optimal policies every time, RL-DT 28 out of 30, and Q-learning 18 out of 30. Q-learning was found to learn quicker than the other two algorithms, but once the models were completed, the two model-based algorithms found better final policies.

The problem proposed in this paper has a larger search space than the penalty kick domain, making the generation of a model more computationally expensive. As seen in the results, Q-learning began accumulating more reward while the model-based algorithms were exploring and building a model of the environment's dynamics. On a physical robot, with a larger search space, building a model would take many more episodes. However, a model-based approach could be used if testing in a simulator, as episodes can be executed quickly. The resulting model could then be used to find an optimal policy through value-iteration. Figure 3.4 shows that simulator and real world test results are very similar, and arrive at the same optimal policy. This optimal policy, when applied to a physical robot, would therefore be superior to the optimal policy from a model-free approach, derived from physical testing. This is discussed further in section 8.1.

A reinforcement learning approach to a kicking motion will see many similar states in a state-action value function. The supervised learning approach in [9] could be used to generalise similar states in the learning problem in this paper.

[9] used reinforcement learning to optimise a single parameter, the kicking angle,

as this was the criteria for a successful penalty kick. This paper will extend the application of RL into a multi-objective optimisation problem, to satisfy the main elements of a kicking motion: balance, accuracy, distance, and execution speed. Multi-optimisation can be achieved through the reward process, which is discussed in detail in section 5.5.2.

3.4.2 Walking Engines

An area of humanoid robotics in which reinforcement learning has been significantly studied is that of optimising a walking gait [17, 41, 42]. These papers provide useful insight to the application of reinforcement learning in a humanoid context. Walking is considered a continuous task with many states which can cause the curse of dimensionality. As such most applications of reinforcement learning use value-function approximation methods instead of a lookup value table. value-function approximation uses experiences to generalise over a larger subset of the state space. The kicking motion can be seen as an episodic task in which there is a finite number of states and actions. The approach in this paper suggests a method to reduce the search-space to a manageable subset, which can be described easily in a value-function table. Function approximation implementation are discussed further in future work section (8.1).

3.5 Summary of Related Work

From the team reports, it is clear that all kicking motions can easily be divided into universal phases. These phases lend themselves well to being represented as the time-steps of a reinforcement learning problem. [35] implements a simple and effective method to alter trajectory angles. If the kicking motion is split into time-steps, which correspond to these phases, this adjustment method could be employed. All kicks use the front curved bumper of the foot, which could be the leading contribution to inaccurate kicks.

It is unclear whether a separate, on-line balancing module increases the accuracy of a kicking motion when undisturbed, but some form of balance control should be implemented to minimise disturbances that could affect the accuracy of a

kick. This could simply be in the form of a negative reward for some measure of movement from the gyroscope feedback. Balancing for kicking motions is tested and discussed in section 4.2.

A value-based reinforcement learning method provides a logical representation of how good a state is, but requires a problem to be described as a Markov Decision Process. Value-based methods however, are theoretically proven to find an optimal policy from a set of stationary policies [32]. Reinforcement learning problems, trying to learn humanoid motions, can suffer from the curse of dimensionality because the number of policies increases exponentially as the variables in a state increase. The search-space must be reduced as much as possible, without reducing the detail required to gain an optimal policy that accurately describes the optimal kicking motion.

[17, Ch.2.5] explains how temporal difference methods such as Q-learning retain detail of previous episodes by only updating state-values by a fraction of the reward (a problem suffered by Monte Carlo methods). However, this causes TD methods to suffer delayed impact from rewards that occur far into an episode. This can be solved with eligibility traces. The problem in this paper can be split into episodes containing only 4 or 5 time-steps, if split into the sub-states of a kicking motion. As such, this problem will not require eligibility traces.

4. Preliminary Work

Before a methodology can be written for the implementation of the reinforcement learning algorithm, preliminary testing must be conducted to verify the findings from the related work. All code written for the robot should first be tested in a simulator. We therefore discuss the search for a suitable simulator.

4.1 Simulation Testing

Testing with robots can be a costly process in many respects. Each test must have the environment set up to the exact specifications in order to gain reproducible results, without being influenced by unforeseen environmental impacts. This, in itself, is time consuming. As reinforcement learning testing can, depending on the complexity of the problem, take upwards of hundreds or thousands of iterations, the time consumption only increases. It can also be costly financially, as modern robots can be expensive and susceptible to damage from falling or stressing mechanical parts. The motors used to move the joints of the robot also have limited periods of use. Like many mechanical parts, when under constant strain, the motors of a robot can overheat. The repetitive nature of reinforcement learning, and with the requirement to place the weight of the robot on one leg, puts a lot of strain on the ankle and hip motors of the balancing leg. Due to these issues, testing is ideally carried out primarily in a simulator.

Simulators offer a cost-effective alternative to testing with physical robots. Simulators are always available and environments can be quickly be set up and reset. Simulators use physics engines to recreate the physical properties of the real world. With a physics engine, simulators can quickly calculate the outcome from a test scenario such as that of a kicking motion without having to visually produce the actual test. This gives the added benefit of reducing testing time, meaning that multiple trials can be conducted in the time it would take to carry out one real world trial. Although simulation testing carries many benefits, it also comes with its share of problems [43].

For a simulator to produce accurate results, it needs two main components: accurate physics, and an accurate model of objects in the environment. The physics engine of a simulator must be as realistic as possible if the results from any simulation are to be used as predictions of the outcomes on the real robot. Gravity, impulse from collisions and mass are a few examples that must be accurately calculated within the simulator. The accurate physics engine is useless without an accurate model of the environment itself. In RoboCup, the ball, pitch surface, and robot's weight, surface friction and material properties must be accurately represented. To get an accurate representation of the ball and pitch surface properties is straightforward, but to produce an accurate model of the robot's physical behaviour is more complex. The accuracy, torque, and compliance of the motors are required to be able to represent the joint movements accurately, when moving or making contact with other objects in the environment. The effects of simulation testing inaccuracies, and how to improve robot models, have been explained in [38].

Despite the modelling complexity, when accurate models are used, simulated testing can be used to prevent physical damage to expensive machinery while producing similar results to physical testing. Possible problems that could occur in the case of kicking a ball with a Nao robot are:

- Damage to limbs and motors when the robot falls. As we want to produce the largest force possible when contacting the ball, the motors are set to full stiffness, meaning very low compliance. If the robot were to fall for any reason, the motors could sustain damage.

- Damage to the motors and sensors through over heating. As the robot would be balancing all the weight on one leg, with motors running at full stiffness, the motors would heat up quickly and would eventually cause wear.

- Testing could be affected by elements outside of the controllable environment, such as motor slippage, varying surface conditions, and performance changes due to battery level and motor temperature.

There are several simulators that have physics models of the Alderbaran Nao robot and all were tested for their use for this research. Unfortunately, at the

time of writing, all simulator were rendered unfit for purpose.

The first to be tested was SimSpark [37], a widely known simulator used since 2004 for the official RoboCup simulated football series. A model of the Alderbaran Nao robot was introduced in 2008 but the simulated RoboCup league has primarily been used to propose multi-agent research questions. The official model of the Nao robot has been shown to be an inaccurate model but has been improved with the modifications explained in [38]. SimSpark does not use the NAOqi framework (see section 2.1) for communication with the virtual robots and instead uses HTTP protocols. Nao Team Humboldt have produced a unified interface to allow their team to play in both a SimSpark simulation and with a real robot team [38]. As we wish to use the already existing NAOqi framework, SimSpark was unsuitable (see section 2.1 for the advantages of NAOqi).

Alderbaran released their own simulator, NAOsim during the early months of this paper which allowed users to communicate with a virtual Nao through their virtual interface, Chorographe [44]. This simulator uses the NAOqi framework to communicate with the virtual Nao, through the same protocols as the real robot. This makes switching between real robot and simulated robot quick and painless. NAOsim was a perfect solution for testing simple movements, and testing balance and control. Unfortunately, it was unsuitable for the reinforcement learning testing as environments could not be scripted or measured. Environment objects could only be placed manually into the virtual world, preventing the user from collecting location measurements from the environment. The only advantage was that of protecting the robot from damage in preliminary testing. As such, NAOsim will be used for testing preliminary motions, only to ensure the safety of the robot. NAOsim was made redundant in February 2013 by a new, more suitable simulator which is highly recommended for future work (see section 8.1).

Before any motion is executed on a real robot, it should be tested in a simulator, to ensure that all calculations are correct and to prevent hardware damage. All of the experiments in this paper were first executed in the NAOsim simulator as described above. No simulator, at the time of writing, was able to satisfy the requirements of: modelling the robot's action accurately, allowing scripted manipulation of the environment, and ability to use the NAOqi connection proto-

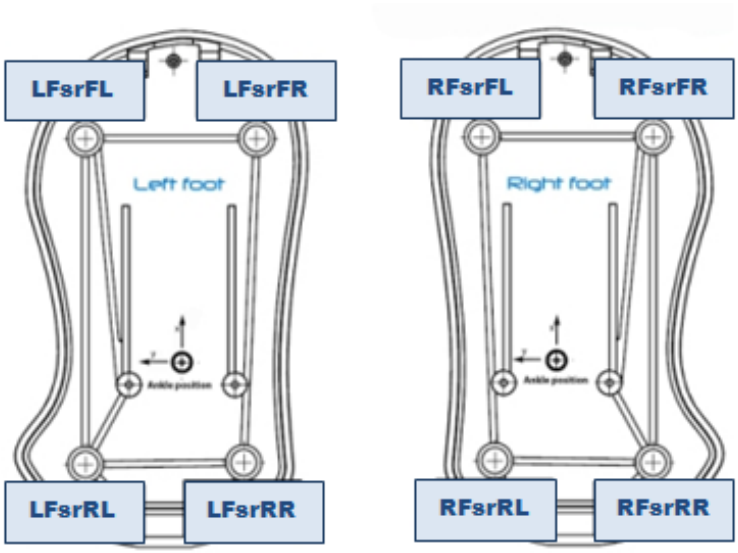

Figure 4.1: The distribution of the pressure sensors on the bottom of Nao's feet

col. Therefore, all reinforcement learning testing will be conducted on the actual robot.

4.2 Robot Balance

The reinforcement learning tests for a kicking motion must be conducted from a well balanced position, to prevent the robot from falling. Past implementations from kicking engines have all seen the robot shifting its centre of mass (COM) over to the balancing leg, by rotating the ankle and hip motors, as seen in Figure 4.2. Using the four corner pressure sensors (force sensitive resistors) on the underside of the robot's feet (see Figure 4.1), we can determine the correct amount of weight shift [45]. The sensors measure a reading between 0N (Newtons) and 25N, which is then converted into kilograms as output readings. When all four pressure sensors read the same value of pressure, we can assume the COM is directly over the centre of the balancing leg.

Through iterative empirical testing, a balance was struck between weight shift, COM height, and kicking potential to ensure that the balance was maintained when one of these three parameters were changed. All measurements of the

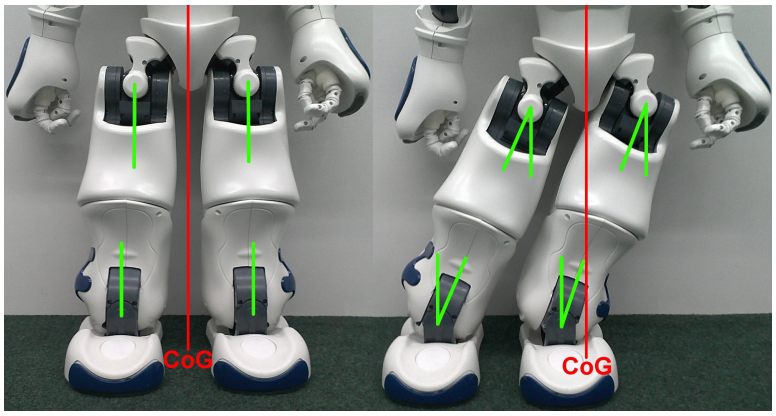

Figure 4.2: The projection on the centre of gravity and the angle change in the ankle and hip joints

robot's movements are measured in metres, relative to the robot's initial pose as seen in Figure 4.2, left. The joint angles for the hip, knee, and ankle are all calculated through inverse kinematics (see section 2.2).

The correct weight shift was found by moving the robot's torso in $1mm$ intervals in either direction of the starting position, and taking the difference of the combined left pressure readings against the combined right readings as seen in equation 4.1.

$$difference = |(leftFront + leftBack) - (rightFront + rightBack)| \quad (4.1)$$

The results in table 4.1 show an almost evenly distributed COM at a shift of $70mm$ from the initial pose. Through the iterative process this was finally set to a $70mm$ weight shift over the balancing leg with a downwards shift of $15mm$ as this gave the best balance, foot reach and average distance for the original Edinferno kick, and the benchmark kick.

We must also strike a balance of height displacement through bending of the knee joint. Too low, and the robot will be unable to gain sufficient leverage at the end of the foot for a good distance; too high, and the robot will become unstable during the momentum shift of the kicking motion. The height of the COM was tested after the benchmark kicking motion was found (see 4.4). We tested 3 different heights of the hip: 10mm, 20mm, and 30mm lower than the original

Hip translation from initial pose (mm)		Pressure Sensor output (kg)		
y-axis	z-axis	Inside Edge	Outside Edge	Difference
60	-10	3.49	1.9	1.59
60	-20	3.84	0.83	3
70	-10	2.83	2.6	0.17
70	-20	2.54	2.78	0.24
80	-10	2.13	3.34	1.21
80	-20	2.53	3.15	0.62

Table 4.1: The weight pressure on the right leg with left leg raised and COM moved over the right leg

	Benchmark Kick			Edinferno Kick		
Height (mm)	Average Dist. (cm)	Standard Deviation (cm)	Stable (%)	Average Dist. (cm)	Standard Deviation (cm)	Stable (%)
-10	523	5.09	70%	345.8	21.9	100%
-20	514.2	11.6	70%	405.6	35.8	100%
-30	515	3.4	20%	369.2	33.5	100%

Table 4.2: The balance and distance averages for COM over 10 kicks

pose. The original height was not tested as it gave no flexibility for moving the foot laterally; any lower than $30mm$ clearly made the kicking motions less effective. Table 4.2 shows the results of using the benchmark, and the static Edinferno kicking motions 10 times at the three different heights. The robot was deemed unstable if any of the foot sensors were no longer under pressure by the robot's weight.

Although the robot did not fall, from these results it was clear that moving the robot too low made the benchmark kick unstable. The benchmark kick was designed to take as little time as possible to kick a far distance. Although the distance to move is increased, the interpolation time remained unchanged. This increased the lateral momentum, causing the robot to 'pop' and become unbalanced as it returned to a stationary position. The results indicate that the robot is more stable and produces better kicking motions between a $10mm$ and $20mm$ drop in height. Taking into account standard deviation, average distance, and the increase in foot range, it was decided that a $15mm$ height drop would be used for the reinforcement learning testing.

In [36], the robot's arms moved to counter the momentum shift of the kicking leg. As the robot's soles are flat and constructed from plastic, the robot is unable to grip the surface of the carpet. It was found that the robot's bearing can be changed by as much as 28∘ from the kick momentum. We tested the benchmark kick with and without arm movements and it was found that the robot's position was less affected by the momentum of the swinging leg with the arms moving in the opposite direction.

The bearing change was calculated by lining up the robot's feet parallel to the sideline of the pitch. After the kick, a piece of paper was placed in front of the robot's feet and the angle between this paper and the sideline was measured. Out of 10 kicks with the arm movement, the robot's bearing deviated a maximum of 12∘ with an average deviation of 7.3°. However, this did not affect the ball's trajectory, as the bearing change occurred from the propulsive phase of the kicking motion (due to having no backswing). Without the arm's counter-momentum, the robot's bearing changed a maximum of 28°, with an average of 25° change. Although this did not affect the trajectory of the ball, it did affect the robot's balance, causing the robot to wobble, but never fall. As such, a counter movement of the arms will be used in the reinforcement learning motion learning. The team's current code also places the arms a little wider when kicking to help distribute the weight. This will also be carried through to the new kicking motion.

We discussed in section 3.2 the implementation of balancing modules to help stabilise a robot while kicking a ball. Although [23] describes a dynamic stability engine which can cope with external influences, there is no discussion on how accurate a kicking motion is, while being disturbed by external forces. However, the advantages of a stability module for coping with dynamic changes to a ball's location are clearly explained in [46]. The objective of this paper is to obtain an optimal kicking motion for a static environment, in which the ball is always in the same place. Therefore, a dynamic balancing module should be referred to future work, when building a kicking engine for a dynamic environment, based on the motions acquired in this research.

Axis	Min	Max
x	-80mm	+100mm
y	-25mm	+30mm
z	-0mm	+70mm

Table 4.3: The limiting reaches of the kicking foot

4.3 Movement Limits

In order to define the search-space for the leg movements, we must find the limits of the kicking leg in all directions. As with finding the balancing point, we use empirical testing to find the limits by moving the leg in the x, y, and z-axis separately, until all limbs in the leg are parallel or touching. $1mm$ was added to the limits to prevent motor damage from joint lock. The z-axis values were also limited to the height of the ball, as movements above the ball's height would not result in impact. Table 4.3 shows the resulting limits for the foot when the torso is moved by $70mm$ over the balancing leg and $15mm$ down.

These measurements have been cropped to take into account extreme reaches of multiple-axis; for example, reaching as far forward and as far out as possible. The leg was able to move further in all directions when not at two extremes due to the spherical reach of the hip ball joint. [47] describes a solution to this problem, but is not implemented in this paper.

Interpolation time limits were found by: (1) taking two key-frames that make a linear movement between time-steps; and (2) finding the smallest interpolate time possible between them. The resulting time was 0.1 seconds. This time will be too fast for some key-frame pairs between time-steps. If it is used as the interpolation time for these cases, the motors will not move; this results in a negative reward which will ensure that the time is not selected again in that state. The maximum time limit chosen was 0.3 seconds as this results in a 0.6 second forward swing which is completely stable, but still produces enough force to move the ball 1 metre.

4.4 Benchmark Kicking Motion

As discussed in 2.1, it is not expected that the optimal kicking motion will require a large backswing as implemented by [13, 23, 36, 39]. It was hypothesised that the execution time of a kicking motion could be significantly reduced, without affecting the kicking distance, by removing the backswing. The backswing with motor control only served the purpose of allowing the motors to build up to full speed before contact with the ball. The kicking engine is called when the ball is within 18mm of the foot and it was hypothesised that this would be enough distance to build the motors up to significant speed before contact.

The kicking motion currently in use by the Edinferno team formed the basis of the benchmark kicking motion. After removing the backswing, some further phases of the kicking motion were merged to remove more excess execution time: the lifting on the kicking leg was merged with the forward swing, and the placement of the kicking leg was merged with the recall of the kicking leg. It was assumed that these phases could be merged as the soles of the robot are smooth and should not snag on the carpet when kicking. This left only four phases to the kicking motion: shift weight over the kicking leg, kick forward, replace foot, and distribute weight evenly. Through empirical testing, the interpolation time between each key-frame was reduced as much as possible, while maintaining stability based on the above criteria. The forward key-frame of the propulsive phase was corrected to make contact with the centre line of the ball, at $3.5cm$ from the ground. The resulting kick, as demonstrated by the results in table 4.3, was a kicking motion that executed in 1.54 seconds able to cover a distance of up to $535cm$, with an average distance of $523cm$, and a standard deviation of $5.09cm$. The kicking foot extends out by $13.5cm$, ensuring firm contact with the ball when the kicking engine is executed. Although the robot does not fall, the robot does sometimes unbalance with a slight wobble. This should not affect the robot's ability to begin walking directly after an executed kick.

This kicking motion falls short by $20cm$ of the kicking engine in [23], but improves upon the distance of Austin Villa's kick in [34] by $170cm$. Although only a static motion, this kick could be adapted to dynamically align the foot with the ball before kicking. This, of course would increase the execution time of the kick. It

is this kicking motion which we shall compare against the kicking motion derived from the reinforcement learning approach.

4.5 Uncontrolled variance

Simulators, although able to accurately simulate real world physics, cannot always model the uncontrolled variables that effect real world testing. Robot control problems suffer from many uncontrollable variables due to the physical properties of moving, mechanical components and the environment they manipulate [48]. The problem of kicking a ball is no exception. Although the servo motors in the Nao robot produce accurate movements, motor-slippage, wear, and heat will always produce some variance in a repeated action. The placement of the ball and changes in surface properties will also affect the trajectory of the kick.

In reinforcement learning, it is important that an agent's reward is not affected by variables outside of their control, or the learning problem is no longer static and a true optimal policy may not be found. We observe 20 kicks, using the benchmark kicking motion, kicking the official RoboCup ball from the same spot, $18mm$ away from the centre of the kicking leg. The average distance recorded was $521.9cm$ and a maximum deviation of $9.9cm$. The standard deviation for distance was $5.46cm$ with variance 29.89. The average accuracy error from the trajectory line was $20cm$, with standard deviation $7.64cm$ and variance 58.4. Figure 4.3 shows a graphical representation of the maximum deviation expected.

From these results, it was determined that the reward for distance would be in $10cm$ intervals. The accuracy of the kick depends on many variables that are outside of the agent's control, including ball placement. As discussed in section 5.7.6, we use a template to place the ball as accurately as possible, but this may not eliminate all variances. As such, a penalty of the same value for not kicking the ball will be issued if the robot's bearing changes. The bearing change of the robot is due to momentum, which is in the agents control, and so the agent can receive feedback for this.

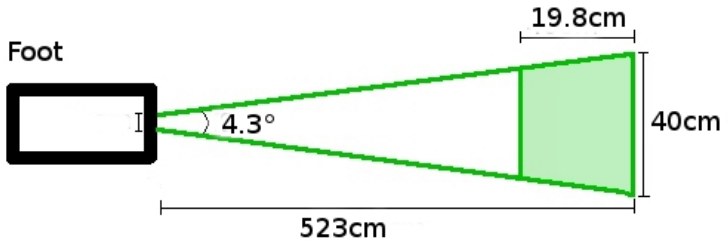

Figure 4.3: A graphical representation of the maximum deviation of the kicking motion

4.6 Preliminary Reinforcement Learning

The related work has presented us with a way of applying reinforcement learning methodology to the optimisation of a kicking motion. We discuss in detail how the problem was approached in section 5, but to summarise for the context at hand: a kicking motion can be divided into phases, such as the back-swing and forward-swing. At each phase the agent could place the robotic foot in any location defined by key-frames; we call these locations the states of the agent. The agent's actions can be defined as the motor commands required to move key-frames. To ensure that the robot is stable when executing the kicking motion, and to reduce the learning problem, the distribution of weight phases are performed outside of the reinforcement learning. The fastest, most stable solution to these phases were empirically evaluated in section 4.4 and adds a total of 1.4 seconds to the execution time. Figure 4.4 depicts a example break down of a kicking motion into 3 time-steps (phases of a kick), 3 states per time-step (different heights of the foot), and 3 action types (slow, medium, or fast interpolation).

As a preliminary test, to prove that reinforcement learning can be suitably applied to the problem at hand, a test was conducted to optimise a linear kicking motion, with a restricted search-space. The following experiment was conducted to evaluate the key elements of a reinforcement learning problem. Firstly, we wish to determine whether convergence to an optimal policy is possible, given current decomposition of the problem to time-steps, states, and actions. Secondly, we wish to acquire a suitable reward-function, given the optimisation criteria. Finally, we must find a suitable learning rate, to reduce the amount of episodes required.

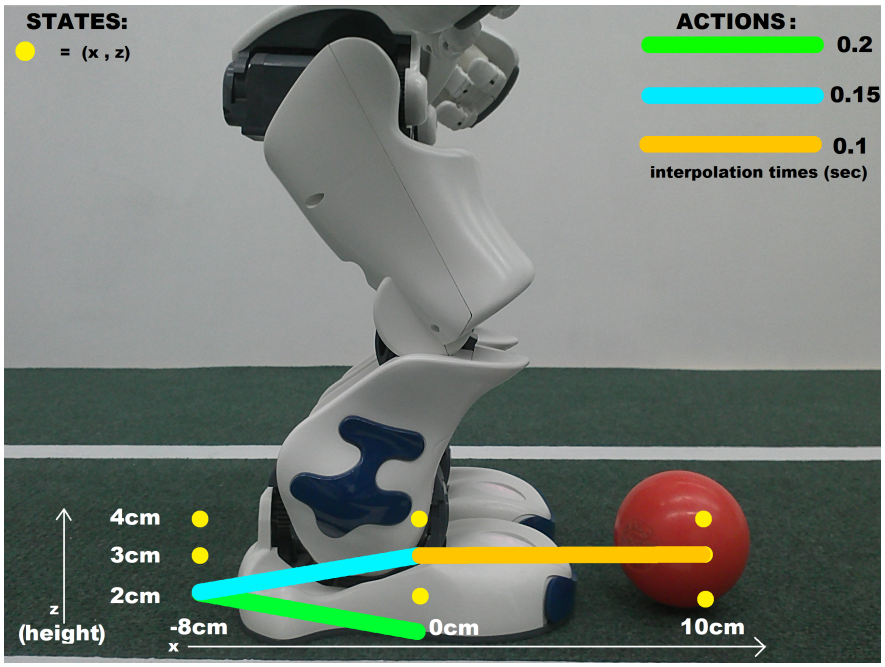

Figure 4.4: A graphical representation the states and actions available to the learning agent

Monte Carlo (MC) First-visit is an on-line model-free RL algorithm that does not require a model of the environment, bootstrapping, or immediate rewards [5, Ch. 5.4, pp. 122]. Instead, Monte Carlo algorithms are rewarded a final return at the conclusion of an episode; in Monte Carlo, we refer to the reward for an episode as the return. Monte Carlo was chosen for the algorithm's simplistic policy iteration and reward functionality; the convergence time is not a priority as the limited search-space ensures quick convergence, regardless of the algorithm. We use First-visit as there is no possibility to revisit the same state in one episode, due to the implementation of the search-space. Algorithm 1 shows the On-Policy First-visit Monte Carlo algorithm used in the simple test.

We use an on-policy implementation to evaluate and improve the policy being used to produce episodes. The policy (π) in this algorithm is an $\epsilon - soft$ policy (4.2); for all $s \, \epsilon \, S$ and $a \, \epsilon \, A(s)$, the probability of choosing action a in state s is greater than 0. Specifically, we use $\epsilon - greedy$, meaning that with probability $1 - \epsilon + \frac{\epsilon}{|A(s)|}$, we exploit the state with the highest value (the greedy state); alternatively, we explore a non-greedy state, each with a probability $\frac{\epsilon}{|A(s)|}$ of being chosen. We use ϵ to control the balance between exploration and exploitation.

This is a widely used method in which to evaluate the current policy; by exploring outside of the policy actions, we are able to evaluate whether the current policy is better or worse than the newly explored policy (see section 2.4.3).

$$\pi(s, a) > 0 \tag{4.2}$$

The search-space will be limited to 2 time-steps, each with 3 states, and each state with 4 possible execution times. Monte Carlo was implemented for this test, as these methods update all states from the episode with the final return. This reduces the amount of experience required before the influence of actions is realised at the earlier states. With only two time-steps, a linear kicking motion will be produced. By limiting the search space, we ensure the whole search-space is quickly explored. As the search-space is very limited, it is expected that an optimal policy will be found in relatively few episodes. As such, ϵ will be calculated using equation $1/0.1 * episodeNum$, where $episodeNum$ is the current episode number. This guarantees exploration for the first 10 episodes, after which the exploration rate drops at a logarithmic rate until episode 100 when the exploration rate would be $\epsilon = 0.1$ (10% exploration).

In the implementation (see Algorithm 1), we begin by preparing the test environment for the agent using algorithm 2 described in section 5.4; we define the states and actions using the defined robot limits, number of states, interpolation times, and time-steps. A full description of the set-up is discussed in section 5.4. We then run an implementation of Monte Carlo reinforcement learning for 100 episodes, using the final greedy policy ($\pi*$) as our optimised kicking motion.

In (a), we calculate whether to exploit or explore in each time-step of the episode, using a random value between 0 and 1, and the ϵ-greedy value in this state. This is our policy evaluation. The policy used for an episode is a list, which holds the index value to the corresponding key-frame in the state-table. With the episode policy determine, we use the index values from the policy list to retrieve the corresponding key-frames for the states of the episode. Using the Nao SDK modules (see section 5.2), motor commands are sent to the robot to move the end-effector (the foot) between the key-frames in a linear motion.

After all states and actions in the episode have been executed, the kicking motion

Initialise Environment:
Set: $noTimeSteps$, $noInterpolationTimes$, $noStates$
Calculate all $s \epsilon S$, and $a \epsilon A(s)$ given x, y, z, t step-sizes from alg: 2
π, $\pi^* \leftarrow$ arbitrary policies
$Q(s, a) \leftarrow$ all zero
$Returns(s, a) \leftarrow$ an empty list, for all $s \epsilon S$ and $a \epsilon A(s)$

$episodeNumber \leftarrow 0$

while $episodeNumber < 100$ **do**
$episodeNumber \leftarrow episodeNumber + 1$
$\epsilon \leftarrow \frac{1}{0.1(episodeNumber)}$

(a) For each t in $timeSteps$:
Get s and a for each t in episode, given π and ϵ
(b) Get $keyFrame$ and $interpTime$ from state-table using reference (s_t, a_t)
Execute episode using $motionModule(keyFrame, interpTime)$
(c) Calculate return using alg: 4
(d) For each s, a pair from the episode:
$R \leftarrow$ return from episode
Append R to $Returns(s, a)$
$Q(s, a) \leftarrow$ average$(Returns(s, a))$
(e) For each t in $timeStep$:
$a_t^* \leftarrow \arg\max_a Q(s_t, a_t)$
$\pi^*(s_t) \leftarrow a_t^*$
(f) For all $a \epsilon A(s)$:

$$\pi(s, a) \leftarrow \begin{cases} 1\text{-}\epsilon + \epsilon/|A(s)| & \text{if } a = a^* \\ \epsilon/|A(s)| & \text{if } a \neq a^* \end{cases}$$

end while
return π^*

Algorithm 1: On-line First-visit Monte Carlo

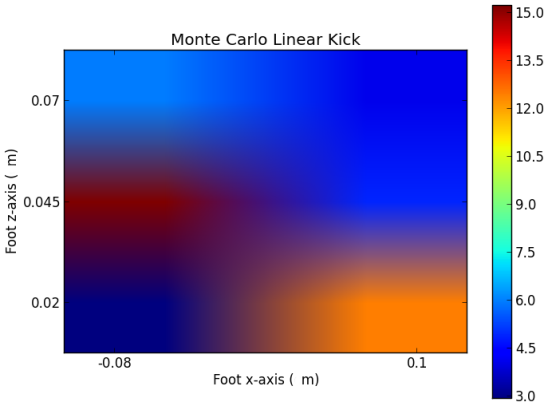

Table 4.4: A heat map presentation of the preferred foot motion

	Foot Sagittal position	
Height	-8cm	+10cm
7cm	5.98	4.1
4.5cm	15.235	4.94
2cm	2.94	12.48

Table 4.5: The state value table

is complete. We then must calculate the return from the episode. The experiment will use the same multi-objective reward-function as the final implementation (see algorithm 4). These objectives are: minimizing execution time, and maximizing stability, accuracy, and distance. A return of 1 will be given to every $10cm$ the ball travels, and for reasons discussed in section 5.5.2, a value of -1 will be rewarded for every hundredth of a second it takes to execute the kicking action. If the robot falls during the kicking motion, the agent will get a return of -500. If the robot does not kick the ball at all, the return is zero plus the execution time return. A return of -500 (for a fall) was selected as this is how long it would take for the robot to return to its feet, according to the above execution time reward function. Zero was selected for no kick, as we do not wish to negatively reward previous states in the episode, that may not have been the contributing factor of the missed kick.

Each state-action pair, $Q(s, a)$, has a list which retains all of the returns from every execution of action a in state s to that state. In (d), the return from the episode is added to this list, and the average over this list gives the value of that state-action pair. The action with the highest value is denoted a^*. Policy improvement occurs in part (e). We iterate over each time-step, t, and give a^* the greedy probability $1 - \epsilon + \frac{\epsilon}{|A(s)|}$ and all other a probability $\frac{\epsilon}{|A(s)|}$.

Figure 4.4 shows a heat map representation of the state values; highest values are red, and low values move towards blue. The $x - axis$ represents the forwards

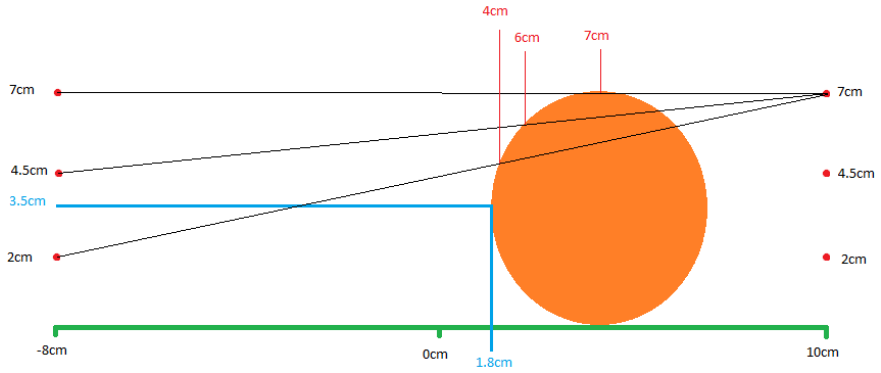

Figure 4.5: This shows the height of contact points with the ball from all back-swing states (red). The optimal contact point is the centre of the ball (blue)

and backwards motions of the foot, with far left the back-swing, and far-right the forward-swing. The $y - axis$ represents the height of the foot from the ground. As our algorithm produces a state-action value table, we combine all of the state-action values for a state and take the average to produce our state-values. These can be seen in table 4.5. The highest valued back-swing was at a height of $4.5cm$ with an interpolation action time of 0.3 seconds from the initial position. The forward swing was favoured at $2cm$ high, with an interpolation time from the back-swing position of 0.233 seconds. This was also the greedy policy for 41 episodes, and produced an average distance of $528.1cm$. We note that the algorithm works correctly, and explores new motions by changing the the key-frames depending on the current policy. After 23 trials, the agent was exploring kicking motions that produced distances no worse than $330cm$. From 20 trial kicks, using the key-frames from the final greedy policy, we recorded a kicking motion that kicked $528.1cm$ with a variance of 57.29, and standard deviation of $7.56cm$.

In the preliminary experiments, it was observed that the robot would get stuck on top of the ball in some episodes if kicking the ball at a height of $7cm$. The ball has a diameter of $7cm$ thus, a kicking motion of $7cm$ for both back-swing and forward-swing would only just kick the ball or place it on top of the ball, see Figure 4.5. For the subsequent experiments, $6cm$ will be set as the height limit for the foot.

It was found that a value of 1 per 10cm, did not weight the value of the resulting

distance high enough. For example, in test number 3, the agent's value function converged to a policy that did not actually kick the ball. From analysing the resulting state value-function the greedy policy was correct, numerically. All policies explored in the first 50 episodes had resulted in an average of $350cm$. Although not a noteworthy distance, this still performs as well as the current static kicking motion in use. These stable kicks had used the longer interpolation times, resulting in a large negative reward; an execution time of 0.55 seconds, with a resulting distance of $353cm$, yielded a reward of -20. The final greedy policy's execution time was 0.2 seconds, with an average distance of $11cm$; this yielded an average reward of -19. As such, this policy prevailed as the optimal kicking motion from this test. It is clear that this occurred because execution time was prioritised over distance.

It can be concluded from these results that a few changes were needed to accurately represent the priority of each objective. Firstly, the overall objective must be realised in the reward-function: to produce a faster, more powerful kicking motion than the current, while maintaining stability and accuracy. By making the reward-function return a neutral value of zero, when similar results to the current motion occur, we will firmly ground the reward-function to the primary objective. Secondly, a delicate balance between execution time and distance must be achieved. Finally, angular accuracy was not realised through reward in this experiment. Some interpolation times caused excessive momentum, altering the bearing of the robot. This greatly affected the accuracy and distance achieved in some episodes. As such, in future experiments, the bearing of the robot should be concidered in the reward-function. Solutions to these problems are discussed further in section 5.5.2.

The kicking motion that resulted from this preliminary test proves that a kicking motion can be optimised using reinforcement learning. With a limited search-space and linear action, the resulting kicking motion kicked, an average, $122.5cm$ further than the current Edinferno kick, and $5cm$ further the benchmark kick. However, the total execution time was 2.05 seconds. This is half a second slower than the benchmark kicking motion, but the robot was stable for 100% of the trial kicks.

5. Methodology

5.1 Overview

Previous work on kicking motions optimised for a single objective: either kicking accurately [9], or staying balanced [23]. Angular accuracy and stability are only two criteria in which to evaluate a kicking motion. The following implementation tries to optimise kicking motions for multiple criteria using off-policy temporal difference Q-learning. Specifically, we evaluate a kicking motion on the distance it makes a ball travel, the angular accuracy of which it travels, the time it takes to execute, and how stable the robot is during the motion. These criteria will be realised through a multi-object reward function, giving a numerical reward relating to the desirability of the outcome.

Through the implementation below, we wish to obtain optimal policies that will produce effective kicking motions for five different distances: 1, 2, 3, and 4 metres, and as far as possible, at a bearing of 0°. We will also experiment with how effective these policies are at being applied to directional kicks at 45°. It is intended that the final kicking motions will be suitable for use by the Edinferno team. As such, the optimal policies calculated through RL will be built into a kicking engine. This will select, modify, and execute the correct kick type, when given: a distance to kick, the ball displacement from the centre of the robot, and an angle.

In this chapter, we discuss the software we require to implement the learning algorithm and control the robot. We explain how the kicking motion is to be implemented and executed, using key-frames, Bézier interpolation, and inverse kinematics. We explain how the environment is deconstructed into states and actions that can be applied to a reinforcement learning problem, and how these states and actions are pruned into a manageable search-space. We describe the implementation of the temporal difference algorithm, used to iterate over the search-space to find an optimal kicking motion in the available number of episodes. Finally, we describe the experiments and the set-up we intend to use to evaluate the effectiveness of the algorithm, and the policies it produces.

5.2 Language and Libraries

As mentioned in section 2.3, Alderbaran have implemented the NAOqi control framework [27]. This will be utilised for communicating with, and controlling the robots actuators. By using software provided by Alderbaran, we ensure that required modules, such as inverse kinematics, are optimised for the robot being used. The reinforcement learning agent will be controlled on a network connected laptop, sending motor commands to the robot through the network via the NAOqi communication protocol.

From the two supported languages, Python will be used for the creation of the reinforcement learning module. There are a few advantages to using Python over C++, namely Python's simple approach to list manipulation which allows for easy creation and manipulation of state and value tables. Python programmes have found to have a slower run-time than C++ programmes on the Nao robot which has had noticeable consequences in computational heavy modules such as vision and balance. However, the run-time of the programming language is of no concern in the following programs as all calculations are done pre-execution. From the NAOqi Python SDK, we will specifically be using the Cartesian control API [49]. This library contains the modules necessary to move the end-effectors through Cartesian space. We will also use the pose module to place the robot in a initial pose for the start and end of an episode. Another third party Python module used is Pylot, a plotting library used to plot the resulting action-value tables as heat maps.

5.3 Kicking Motion

A kicking motion can be described as a trajectory, which the foot moves through. The trajectory will be defined by a set of key-frame positions, which describe the foots position in Cartesian coordinates, relative to the initial pose of the robot. Each key-frame describes the foot's position with coordinates x, y, z and joint rotations θx, θy, θx.

The kicking motion will be split into six phases: Shifting the weight to the balancing leg; lifting the kicking leg; moving the kicking leg backwards; moving the

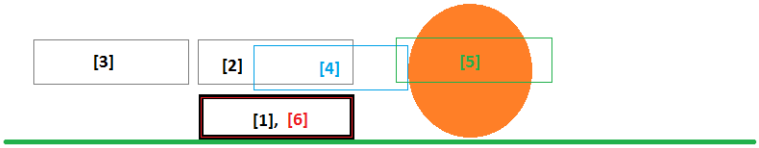

Figure 5.1: The blocks represent the foot position at each stage in a kicking motion

kicking leg forwards to make contact with the ball; the follow-through point; and returning the kicking leg and redistributing the weight to both legs. The six positions are illustrated in Figure 5.1. Phases 1, 2, and 6 have been calculated efficiently outside of the reinforcement learning problem in section 4.4. These phases can be considered independent and unaffected by phases 3, 4, and 5 which make up the propulsive motion. Therefore, the initial state for RL will be the predetermined state of phase 2, and the terminal state in the RL algorithm will be in phase 5.

Between key-frames, the motion needs to be interpolated to calculate the required velocity of the motors in the leg to get from one key-frame to another in a set trajectory. This can be calculated through a linear or a curved motion. Linear interpolation is suitable for motions which travel in one direction and contain only two points, for instance between phases 2 and 3, and phases 5 and 6. At phase 3, the direction of travel reverses in the x-axis. Phases 3, 4, and 5 travel in the same direction along the x-axis, but linear interpolation between these points would result in a jerky motion due to sudden changes in the motor speeds and differing y-coordinates. As such, a curved transition between the three key-frame would produce gradual differences in the motor velocities and a smoother motion. Phases 1, 2, and 6 will use Alderbaran's linear interpolation module for the trajectories, as these phases have sharp changes in direction that should not be smoothed. The trajectory between the key-frames for the propulsive phases are calculated using Alderbaran's automatic Bézier spline interpolation module. To move the end-effector in Euclidean space, Alderbaran's motion modules calculate the joint angles for the actuators using inverse kinematics. A full description of how these work can be found in section 2.2.

In an attempt to further reduce the size of the search-space the possible degrees of freedom of the end-effector were studied, to determine their possible influences in the kicking motion. The roll of the foot is unable to apply force to the ball in a forward kicking motion, and as such will be left perpendicular to the ground in all states. The yaw of the foot is controlled by the coupled hip joint by one motor (see Figure 2.2). Altering the yaw of the foot in either direction reduces the range of the foot, as the knee joint of the robot obstructs any forward kicking motions. As such yaw is also left unchanged in all states.

By pitching the foot so that the top of the foot makes contact with the ball instead of the curved edge, the accuracy of a kick may be increased further. For a ball to be kicked in the intended direction, the ball must be in the centre of the kicking foot. If it is not, then the angle of the kick is changed exponentially, the further from centre it moves (due to the curved edge). If contact is made with the flat top of the foot, the ball may travel in the correct direction, even if central contact is not made. It is not feasible to quickly and accurately move the robot to align the ball with the centre of the foot in RoboCup, as can be witnessed in many previous games. By exploring contact with the top of the foot, the average accuracy could be increased. Another potential advantage of pitching the foot down, is that the ankle motor may be used when contact is made with the ball to add further force on impact. If, when the foot makes contact with the ball, the ankle is pitched up again, the extra movement of the ankle could aid in the impact velocity. When the foot is pitched down, the foot must be raised higher from the ground to give the required clearance. This reduces the possible leverage given by the hip and knee joints (see Figure 5.2) and may potentially counteract the advantages of the ankle motor movement.

Due to the thickness of Nao's legs and the coupled hip joint, the foot needed to be rotated around all three axes (θx, θy, θz), by 0.25, 0.7, 0.2 radians respectively. This rotation allows for a clean swing from rear to front of robot, using the same limits as the normal kicking motion. When using the ankle pitch in a kicking motion, all key-frames prior to ball contact will have the above rotations applied. At the contact point with the ball, the key-frame rotations will be set back to zero radians for all axes. This will make the ankle motor drive the foot through the ball. As the ankle is pitched, we must also modify the minimum hight value (z) from $2cm$ to $5.5cm$ to ensure the toe edge does not contact with the floor.

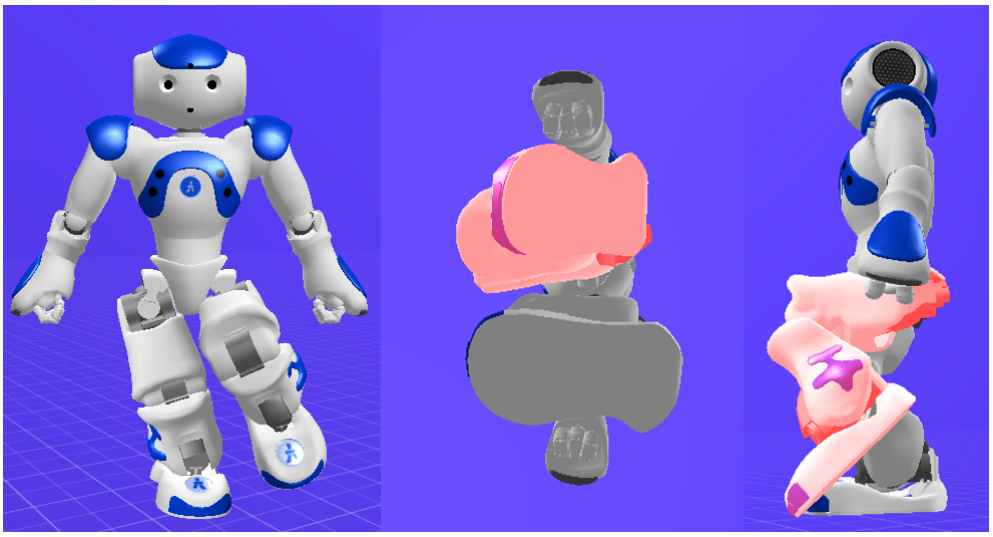

Figure 5.2: Once the ankle is pitched, to expose the top of the foot to the ball, little maneuverability is available to the other leg motors.

The current Edinferno behavioural module initiates the kicking module when the ball is $18mm$ away from the front of either leg. It is therefore reasonable to assume that the distance to the ball will be no more than $18mm$ away in this paper. As such, $18mm$ will be used as the contact point phase (phase 4).

5.4 State-space and Actions

To reduce the complexity of the state-space, each state in the reinforcement learning task will be a key-frame description of the robot's kicking leg. Each state will use the representation seen in (5.1) where x, y, z are a Cartesian coordinate relative to the robot's initial pose, $\theta x, \theta y, \theta z$ are radian values for the ankle's rotation, and t is the execution time from the previous state to the current state.

$$[x, y, z, \theta x, \theta y, \theta z, t] \tag{5.1}$$

It could be argued that execution-time is part of the action, not the state. Time was included in the state description to distinguish between two state-action pairs with key-frames that share the same Cartesian coordinates, but with different interpolation times to be sent to the motion module. As we do not retain the

past interpolation time, our states do not hold true to the Markov property, but it is appropriate to still think of the state as an approximation to the Markov property. As stated in [5, Ch. 3.5, pp. 64], Reinforcement learning can be applied to a problem that is not strictly Markov.

By using reinforcement learning, we remove the need for training data to guide the agent's learning, but we can still reduce the infinitely large search space through reasoning and basic physics. This reduction process can be considered as the rules of kicking; in relation to a real world learning problem, before one can start to learn a card game, they must first know the rules. It is necessary to reduce the search-space for the reinforcement learning methods in order to accelerate convergence of the value-function and policy. The following reductions can be seen as a model for the learning agent, preventing it from exploring states that will not affect the ball or will alter the desired direction of the shot. This model prevents our agent from exploring states that will gain no, or negative, rewards.

The first reduction comes from Newtons 2^{nd} Law: The acceleration and direction of a body is parallel and directly proportional to the net for acting on the body [50]. The contact point must therefore be met with a flat, perpendicular force to the centre of the ball in the direction of the goal location. In our testing, we place the ball directly in front of the kicking leg, at a distance of $18mm$. As such, we can first reduce the dimensional search space to a single Cartesian plane through the ball and the goal location. As we are learning for straight kicks at a bearing of $0°$, we reduce to the sagittal plane, removing all possible $y - axis$ motion. We further reduce the plane to the range of the robot's limbs in both the x and the z directions (height and forwards/backwards) as found in section 4.3. A final reduction can be made to remove all states above the height of the ball ($7cm$ high), as none of these states touch the ball. $1cm$ is further reduced from the height according to the findings in the preliminary testing (section 4.6). States below $2cm$ are also removed as they may intersect the floor if the robot is unstable. We are left with an effective search space range seen in table 5.2. Figure 5.1 depicts the reduction process.

The search-space must now be converted into the state-space to be used by the learning agent. For this, we must map the motion range to a set of key-frames.

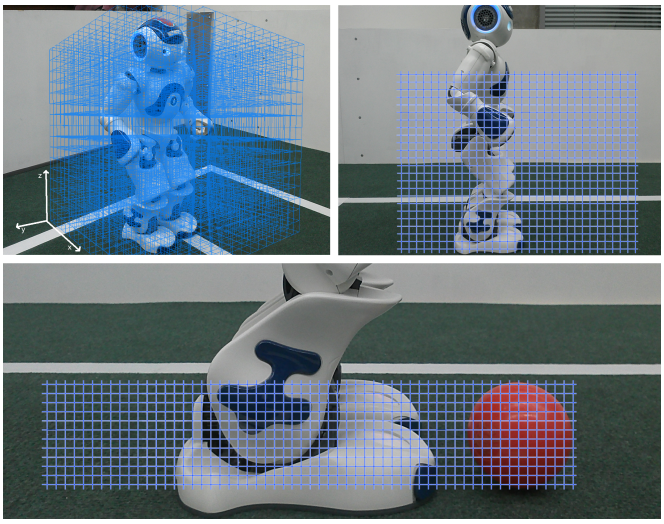

Table 5.1: Reduction the Euclidean search-space to a limited sagittal plane

Axis	Min	Max
x	-80mm	+100mm
y	0mm	0mm
z	+20mm	+60mm

Table 5.2: Search-space range

Remembering that we now only have two varying coordinates for the Cartesian representation, each key-frame must have differing x, y and t (interpolation time) values.

Given that the propulsive action is divided into 3 phases (backswing, contact point, and follow through), and that these phases are to be used as the time-steps, we have three possible x-values. All backswing phases will be assigned the extreme rear value of $x = -80mm$. The contact point, as previously mentioned, happens at $x = 18mm$ and so will have $x = 18mm$. The follow through will use the extreme limit of $x = 100mm$.

Testing is to be conducted on a real robot. Therefore tests must be limited to a low number of episodes to prevent over heating and damage to the motors. To ensure an optimal policy can be found in the episodes the number of states per time-step, and the number of interpolation times must be restricted. However, we must retain as much detail from our search-space as possible in the process. From the limits of the search-space it was decided that $1cm$ intervals in the z-values retained enough detail to describe varied trajectories. This gives 5 states per time-step.

The available actions in a given state and time-step are described as the motion

command that begins the interpolated movement to the next point. These motion commands take the next state and an interpolation time. From section 4.3, the interpolation times must be within 0.1 seconds and 0.3 seconds to produce a kick. We will use 5 different interpolation times, including 0.1 and 0.3 seconds, giving 0.5 second steps. As we include the interpolation times in the states, this gives a total of 15 different states at each time-step. Using equation m^n, where n is the number of states and m is the number of actions, we have a total of 5^15 possible policies.

Once Alderbaran add environment control features to their simulator, the program implemented to set up and define the environment for the agent can easily be modified to take more states, interpolation times, and even more time-steps. The program was created to take inputs for the number of states, the number of interpolation times, and the number of time-steps, and produce the required state table, state-value table, and action-value table. Part (a) of algorithm 2 uses initial variables for ball contact distance, interpolation time limits, and Cartesian limb limits to calculate the step size for all variables of the Cartesian key-frames. Part (b) calculates the y value at the contact point for future work with reinforcement learning on angled motions. It uses the distance percentage between the start and the endpoint to find the corresponding point between the y coordinate limits.

5.5 Reinforcement Learning

With the state-space and actions defined, we now turn to the implementation of the learning agent and reward-function. As mentioned in section 2, the main components of robotics are: actuators, effectors, sensors, and a decision making process. The former three have been defined as part of the environment which the state-space now describes. The latter can be thought of as a learning agent. Through a numerical reward, the learning agent will calculate a policy for choosing the actions available in the states. In this section we discuss the implementation, and supply the pseudo code, for the development of the learning agent and reward function.

Initialise Variables:
$minX$, $minY$, $minZ$, $maxX$, $maxY$, $maxZ \leftarrow$ motor limits
$minTime$, $maxTime \leftarrow$ interpolation time limits
$contactDist \leftarrow$ distance from foot to ball
$timeSteps \leftarrow$ number of time-steps
$states \leftarrow$ number of states per time-step
$interpTimes \leftarrow$ the number of different interpolation times
$s \leftarrow states \times interpTimes$

(a) Calculate step size for z and interpolation time:
$zStep \leftarrow \frac{maxZ - minZ}{timeSteps - 1}$
$tStep \leftarrow \frac{maxTime - minTime}{timeSteps - 1}$

(b) Calculate the values of y at contact point:
$percentTowardsMax \leftarrow \frac{contactDist - minX}{maxX - minX}$
$midY \leftarrow percentTowardsMax(maxY - minY) + minY$

Algorithm 2: Step size for x, y and $interpolationTime$

5.5.1 Learning Agent

Reinforcement learning tests, depending on the complexity of the number of states, number of actions, and RL method used, can take upwards of thousands of iterations before the value-function converges. It is known that an optimal policy can be found many iterations before the value-function stabilises [5, Ch. 4.2, pp. 93-95] (see the preliminary RL implementation in section 4.6). Therefore, we do not need to run the required number of episodes to convergence the value-function to their final value, but only the required amount to reach the optimal policy.

As previously mentioned, [9] shows that model-based RL will converge more quickly than model-free algorithms. However, the search-space and number of actions in this problem are of a much higher dimensionality to that in [9]. Building a model of the environment, even with the generalisation algorithms discussed in [9], would not be possible, given the limited number of episodes possible with the robot. Therefore, a model-free algorithm will be used.

We will therefore use the off-line temporal difference method, Q-learning. [9] proves Q-learning to be a fast model-free method, capable of finding good policies

quickly in relatively few episodes (see section 3.4.1). Research has shown that in a static, finite MDP, Q-learning will converge to a optimal policy [40]. We base the learning algorithm on Watkin's Q-learning principles [30, 40]. We use an ϵ-greedy policy to balance exploration and exploitation. $soft-max$ policies were also considered, but it was decided that ϵ-greedy would ensure maximum exploration. 3 is a pseudocode description of the algorithm implemented.

Initialise Environment:
Set: $noTimeSteps$, $noInterpolationTimes$, $noStates$
Calculate all $s \epsilon S$, and $a \epsilon A(s)$ given x, y, z, t step-sizes from alg: 2
$\gamma \leftarrow 1$
$\alpha \leftarrow 0.03$
π, $\pi^* \leftarrow$ arbitrary policies
$Q(s, a) \leftarrow$ all zero

$episodeNumber \leftarrow 0$

while $episodeNumber < 100$ **do**
$episodeNumber \leftarrow episodeNumber + 1$
$\epsilon \leftarrow \frac{1}{0.03(episodeNumber)}$

(a) For each t in $timeSteps$:
Get s and a for each t in episode, given π and ϵ
(b) Get $keyFrame$ and $interpTime$ from state-table using reference (s_t, a_t)
Execute episode using $motionModule(keyFrame, interpTime)$
(c) Calculate return using alg: 4
(d) For each s, a pair from the episode:
$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} = \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
(e) For each t in $timeStep$:
$a_t^* \leftarrow \arg\max_a Q(s_t, a_t)$
$\pi^*(s_t) \leftarrow a_t^*$
(f) For all $a \epsilon A(s)$:

$$\pi(s, a) \leftarrow \begin{cases} 1\text{-}\epsilon + \epsilon/|A(s)| & \text{if } a = a^* \\ \epsilon/|A(s)| & \text{if } a \neq a^* \end{cases}$$

end while
return π^*

Algorithm 3: Off-line Q-learning

Firstly, the environment is described in the state-space using algorithm 2 described in section 5.4, using the number of states, time-steps, and interpolation

times defined by the user. γ is set to a value of 1 as we have very few time-steps, thus future values should not be discount. The step-size parameter α controls how far a state moves towards its successors value. We use a value of $\alpha = 0.3$ to move a state's value one third of the way towards the difference between it and its future greedy state as seen in step (d). A value of 0.3 was chosen from experiencing the effects of high valued step-size parameters in a preliminary test [1]. The action-value function is set to zero for all state-actions, as no model is used. The off-line greedy policy, and on-line episode policy are both set to zero. ϵ is calculated as a very large number using the equation $\epsilon \leftarrow \frac{1}{0.03(episodeNumber)}$, which ensures 100% exploration for the first 33 episodes, after which the exploration rate drops until it reaches 33% exploration by episode 100. This helps to move the action-value function towards its true value quickly.

For each episode, in (a) the learning agent selects the actions to be taken in each time-step using the current ϵ-greedy policy. Each action selection corresponds to a key-frame in the state-table, which is retrieved and added to the episode key-frame list. Once all key-frames for the time-steps are retrieved, the interpolated motion is executed, which will hopefully move the ball. In (c) the user measures the environmental changes, including the distance of the ball, and the bearing of the robot. The user input is converted to a numerical reward using algorithm 4.

Parts (d), (e), and (f) refer to the policy iteration process. In (d), each state-action pair executed in the episode uses bootstrapping to evaluate and move their value to be more like their greedy successor (see section 2.4.4 for more detail). In (e), the off-line greedy policy is improved by iterating through each time-step selecting the largest valued action for its policy, and modifying the on-line policy probabilities to reflect this.

[1] With a step-size value of 0.8, a terminal state-action, with a current value of 90, received a negative reward of -150 for changing bearing in an episode. Using the action-value update rule, the state-action's new value was $-150 = 90 + 0.8(-150 - 90)$. This made this good action selection become a poor selection. with a step-size value of 0.3 instead, the new action-value would have been 18.

5.5.2 Reward-function

The best kicking motion implemented in the preliminary section (4.4) kicked an average of $520cm$, in a time of 0.17 seconds; this is the distance we wish to beat. Our least acceptable distance would be that of the existing Edinferno kicking motion at $348.5cm$, with an execution time of 0.8 seconds. These times exclude the weight distribution phases, as these are static and not included in the reinforcement learning. The reward function should be balanced around around these minimum values; any less incurs a negative reward, and any more a positive reward.

As +1 reward per $10cm$ was found to weight the distance criteria too little, in the following tests the agent will receive +2 for every $10cm$ achieved. The input distance is rounded to the nearest 10. The execution time for each kick will be the state's immediate reward to encourage faster executions. The minimum time-step possible is 0.01 seconds, and as we wish to reduce execution time, we use a minimum time to goal reinforcement function, giving an immediate reward of -1 per 0.01 seconds spent interpolation between key-frames (time-steps). It was found that the minimum execution-time between two frames must be greater than 0.1 seconds, or the robot's motors do not have enough time to execute any action. The reward (or punishment) for time therefore starts at 0.1 seconds, so a value of 10 is added to the immediate time reward. Therefore, with a distance of $350cm$ and an execution time of 0.8 seconds (the current robot abilities), the agent will receive a reward of 0. This should encourage the agent to improve upon the current implementation.

If the robot falls during the kicking motion, or does not kick the ball, the agent will be rewarded -500 and -150 respectively. If the robot's bearing changes, the agent will also be rewarded -150. A reward of -500 (for a fall) was selected, as this is how long it would take for the robot to return to its feet, according to the above execution time reward function. -150 was selected for a change in the robot's bearing, or no kick, as this corresponds with the execution time of having to execute a second kick (if using the benchmark motion, which takes 1.54 seconds).

The terminal state reward for each episode will need to be entered manually after

measuring the resulting kick. From preliminary testing, a variance of 23.2cm distance was found when executing the same kicking motion 50 times on the Edinferno pitch. It was concluded that the variance in the distance was due to elements uncontrollable by the agent being tested. As such, the rewards must reflect this variance, to ensure that the agent is not penalised for uncontrollable environmental errors. For the tests in which a kick of a certain distance is desired, a maximum numerical value of 100 will be rewarded for a ball measurement within 20cm of the desired distance. Therefore every 20cm further or less than the desired distance, a numerical value of 20 will be deducted from the reward, until at 100cm from the desired distance the reward will be 0.

Algorithm 4 shows how the user input of a distance reading, or negative reward (for fall, bearing change, or no kick) into the return for the terminal state. $strategyType$ is defined when the user begins the testing and is a numerical value that relates to the distance, in metres, the user wants to optimise for. A numerical value of zero is used to express the furthest distance possible. The output from this return function is added to the interpolation time, which is calculated within the program as the immediate reward.

5.6 Kicking Module

The final production will be a fully functioning kicking module, which will select and modify the appropriate kicking motions given a goal location. Through empirical testing it was found that, relative to the robot's bearing:

- with the ball displaced by 1cm, the ball was kicked at an angle of 10°
- with the ball displaced by 2cm the ball was kicked at an angle of 30°
- with the ball displaced by 2.3cm the ball was kicked at an angle of 45°

The kicking module will require: input of the ball displacement relative to the centre of the robot in centimetres (minus for left, and positive for right), the distance the ball should be kicked (in metres), and the angle at which the goal is, relative to the robot's bearing (in degrees). The correct set of key-frames will then be selected, depending on the distance, and modified relative to the angle desired

Get command line input:
$input \leftarrow$ distance (cm) or negative return
(a) Longest Distance strategies:
if $strategyType$ is 0 and $input \geq 0$ **then**
  $return \leftarrow \text{round}(2(input/10))$
else
  if $input \geq 0$ **then**
    $goalDist \leftarrow strategyType \times 100$
    if $(goalDist - 10) \leq input \leq (goalDist + 10)$ **then**
      $return \leftarrow 100$
    else if $(goalDist - 20) \leq input \leq (goalDist + 20)$ **then**
      $return \leftarrow 100$
    else if $(goalDist - 40) \leq input \leq (goalDist + 40)$ **then**
      $return \leftarrow 100$
    else if $(goalDist - 60) \leq input \leq (goalDist + 60)$ **then**
      $return \leftarrow 100$
    else if $(goalDist - 80) \leq input \leq (goalDist + 80)$ **then**
      $return \leftarrow 100$
    else
      $return \leftarrow 0$
    end if
  else
    $return \leftarrow input$
  end if
end if

Algorithm 4: Reward Function Algorithm (return only)

and the displacement from the foot. If the displacement is positive, the robot will use its right leg, else it will use its left. If the combined movement needed is past the limit of $\pm 2.3cm$ for the foot, the robot will only kick at $\pm 2.3cm$ in an attempt to get the ball as close to the target location as possible. The algorithm used for the kicking module is described in 5.

5.7 Experiments

In this section we explain the experiments to be conducted to test the Q-learning algorithm described above, on our reduced state-space. We explain how each test is set up to minimize environmental variations between each episode. Each experiment is completed twice, with each test comprising of 100 episodes. Between experiments, the only thing to change is the backswing distance and the reward-function.

5.7.1 Maximum Distance

The first experiment to be conducted is that of the maximum distance test. This experiment involves using the long distance reward-function to gain the furthest distance possible. We also wish to determine whether shorter backswings are more efficient. As such, three experiments will be conducted with the long distance reward-function; one each for a $2cm$ backswing, $5cm$ backswing, and $8cm$ backswing. The backswing that results in the furthest distance will be used again in an extended test of 200 episodes, to evaluate if a more suitable greedy policy is returned with twice as much experience. For the 200 episode experiment, the episodes will be carried out diagonally across the pitch to give more distance potential. This may introduce variances from the carpet surface and line tape.

5.7.2 Specific Distance

The second type of experiment to be conducted involves set distance reward-functions, to get optimised policies for a specified distance. We will use the appropriate backswing for the distance, discovered through the first experiment. The

Initialise:
$keyFrames \leftarrow []$
$foot \leftarrow 0$ (0 = left foot, 1 = right foot)
$angleMove$, $dispMove$, $move \leftarrow 0$

Round distance and get relative foot displacement:
$dist \leftarrow$ round($distance$)
if $displacement \geq 0$ **then**
  $foot \leftarrow 1$
  $dispMove \leftarrow displacement - 5$
else
  $dispMove \leftarrow displacement + 5$
end if

Get the y displacement from the angle provided:
if abs($angle$) < 15 **then**
  $angleMove \leftarrow 0$
else if abs($angle$) < 25 **then**
  $angleMove \leftarrow 1$
else if abs($angle$) < 35 **then**
  $angleMove \leftarrow 2$
else
  $angleMove \leftarrow 2.3$
end if

if $angle < 0$ **then**
  $angleMove \leftarrow -angleMove$
end if

Get total move of y-coords (ensure not past limit):
$move \leftarrow dispMove + angleMove$
if $move > 2.3$ **then**
  $move \leftarrow 2.3$
else if $move < -2.3$ **then**
  $move \leftarrow -2.3$
end if

Get key-frames, interpolation times and alter y-coords:
$keyFrames$, $times \leftarrow$ getKeyFrames($distance$)
$keyFrames \leftarrow$ modifyY($keyFrames$, $move$)

Execute kick for leg specified by $foot$:
for all keyFrame **in** keyFrames, and time **in** times **do**
  interpolateMotion(keyFrame, time)
end for

Algorithm 5: Kicking Module($distance$, $displacement$, $angle$)

different reward-functions for different distances is set by defining the *strategyType* for algorithm 4 when running the learning agent.

5.7.3 Ankle Rotation

We will modify the key-frames, as described in section 5.3, to include the rotation of the ankle. We also set the $minZ$ value in the algorithm 2 to $5.5cm$ to prevent the toe edge intersecting the floor. This test will perform the same as the maximum distance tests, attempting to get the furthest distance possible.

5.7.4 Optimised Kicks

Once the optimal policies for all distances have been discovered, we will test the final key-frames of each kick 20 times to find their standard deviation and variances. This will be used to evaluate how effective the final policies are. To give a more accurate scenario as would be expected in a RoboCup match, the kicks in this experiment will be taken from the goal keeper's position, shooting down the pitch and over the line marking tape. It is expected that this will increase the variances observed between each kick.

5.7.5 Angled & Dynamic Kicks

Each policy will then be tested in the final kicking module. The ball will be placed at different locations relative to the centre of the kicking foot, and the displacement, the goal distance, and an angle will be passed to the module. Although the displacement will change, the distance from the foot of the robot will always remain at $18mm$, as specified by the behaviour module. We will specifically test:

- straight kicks with a ball displacements of $\pm 1cm$ and $\pm 2cm$ for all distance types
- 20° angled kicks for all distances and no displacement
- 20° angled kicks for all distances and $\pm 1cm$ displacement

- 30° angled kicks for all distances and no displacement
- 45° angled kicks for all distances and no displacement

From these tests, we will evaluate how well the final kicking module meets the aims and objectives of this paper. Specifically, the module's ability to kick a ball the required distance and angle, in a timely manner, while staying stable.

5.7.6 Test Procedures

For the testing we will be using a H25 v3 Alderbaran Nao robot, with 25 degrees of freedom, 11 of which we will be utilising in the kicking motion. We use the robot's pressure and motor sensors to read the state of the robot, but all other environmental reading, such as ball travel and shot angle, are measured by hand. The testing will be conducted on the Edinferno pitch, at the University of Edinburgh. The pitch is 4x6 metres and is laid with a green carpet, similar to that on the full sized RoboCup regulation pitches (see Figure 5.3). To ensure that the lines of the pitch do not interfere with the the travel of the ball, the testing will be conducted on the sideline of the pitch, where there are no lines. To protect the robot from damage, the robot will have a loose harness under each arm to prevent collapse in the event of a fall. The harness will allow all movements to go unhindered, and will be measured to catch the robot in the seated position (Figure 5.4).

To ensure that each test begins in the same position, the robot will be placed in its harness, with the rear of the feet parallel, $15cm$ away from the back wall, and $10cm$ from the side wall. A template (Figure 5.5) will be used to place the ball in the same location for each test. The ball will be placed $18mm$ away from the front of the kicking foot, as this is the distance used to trigger the kicking module in the current Edinferno behaviour module.

The kicking distance is measured using an outstretched measuring tape. The tape runs the length of the pitch and the measurements will be taken in centimetres. The angular accuracy of the kicks can be measured by taking a distance reading from the ball to the measuring tape and using basic trigonometry. The speed of the kick is measured and recorded in the algorithm and is the sum of the

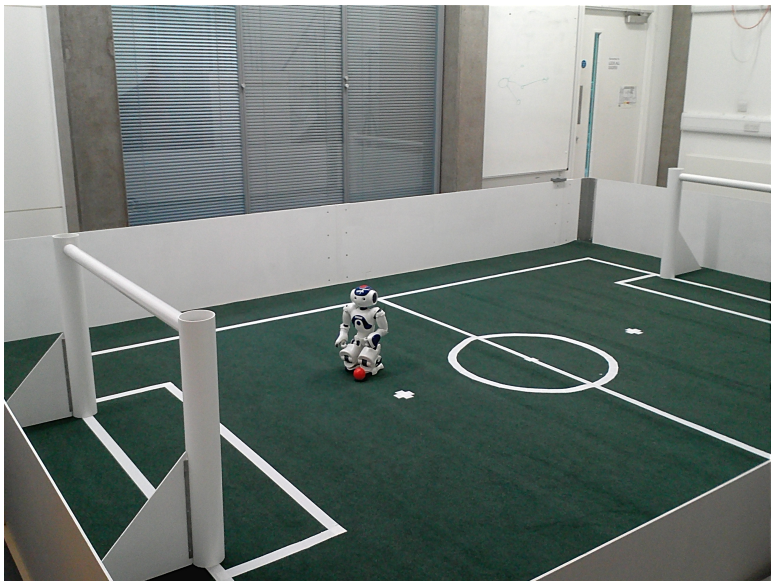

Figure 5.3: The Edinferno Pitch and the H25 academic Nao

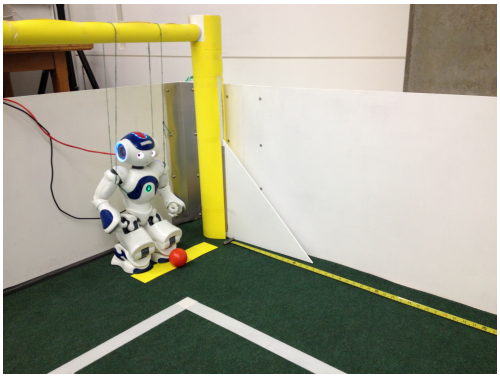

Figure 5.4: Test harness

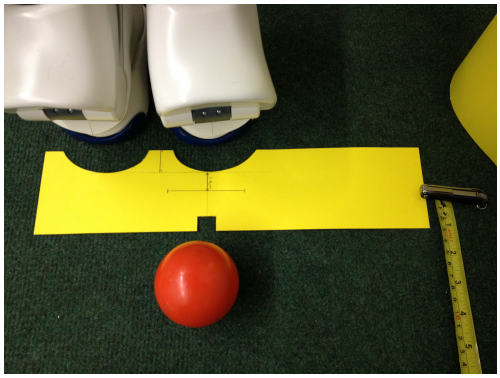

Figure 5.5: Positioning template

interpolation times for each phase.

Although the robot is symmetrical, all testing will be carried out using the left leg to kick only. This is to ensure that the results are not influenced by potential differences in the left and right leg motors. After the optimal kicks are calculated, each policy will be transferred and tested on the right leg to ensure the results are the same.

6. Results

6.1 Overview

This section provides the results of the experiments described in section 5.7. We present the resulting key-frames from each experiment in a graph as a spline curve. The x-axis represents the forward and backward movement of the robot's foot, and the y-axis represents the height of the foot. Both final greedy policies from each test are presented on the same graph for comparison. We present the distance achieved by each episode as a bar chart, with episode count on the x-axis and distance (in centimetres) on the y-axis. When the agent receives a negative reward for either falling or changing bearing, a distance of zero is plotted. Finally, we present the resulting state-value table as a heat map, with the highest valued states coloured red, and lowest valued states coloured blue. Again, the x-axis represents the forward and backward movement of the robot's foot, and the y-axis represents the height of the robot's foot. In Qlearning we record the action-values of states. To produce the state-value, the action-values for each state were summed and divided by the number of visited states-action pairs. By dividing by visited state-action pairs only, we prevent the state-values from being weighted unevenly by unexplored actions. We use the state-value heat maps to evaluate whether the greedy policies complement the state-values.

We present the heat map and distance chart only for the first test, for policy 1. We will therefore discuss greedy policy 1 primarily, using greedy policy 2 for comparison. We evaluate the greedy policy motions against the distance produced by the benchmark kick, and the stability of the robot in terms of the pressure sensor readings. As the benchmark kick has no backswing, it produces a propulsive phase in 0.17 seconds. With a minimum interpolation time of 0.1 seconds and three steps to the propulsive phase, we cannot expect the learned policies to execute faster. Instead, the target is for the execution time to be as close as possible to the minimum, 0.3 seconds. We say that a kicking motion has satisfied the full criteria if it has a faster execution time than the original kick (0.8 seconds), and kicks further than the benchmark kick, while staying stable.

6.2 Maximum Distance

6.2.1 2cm Backswing

From figure 6.1 we see how the agent explores many different policies from start to finish, producing a varied distance output from episode to episode. From episode 82 onwards, we see a slow increase in distance as the policies exploit more states in each episode. We still observe some low scoring episodes being explored, but this is less frequent.

From both tests, the final greedy policies shared the same backswing and follow-through key-frames, in terms of height value and interpolation time. The kicking motion described by policy 1 makes contact with the ball $1.5cm$ below the centre line of the ball ($3.5cm$), whereas policy 2's motion makes contact with the ball $1.5cm$ above the ball. This is equidistant from the centre in both policies. Both greedy policies satisfy the three optimisation criteria: the robot stays stable (keeping pressure on all four foot pressure sensors), the bearing is unchanged, and the execution time of the kick is the minimum possible. The kicking motion, with a maximum distance of $482cm$ was unable to produce a distance greater than that of the benchmark kicking ($523cm$). It did, however, produce a distance $76.4cm$ further than the original kick. In all of the tests the robot stayed stable, never falling, and only gained a negative reward from bearing change 3 times.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.1	0.1
2	0.1	0.1	0.1

Table 6.1: Policy interpolation times

From the heat map of test 1 (6.3), we can see that the first two time-steps in greedy policy 1 are the highest valued states in the respective time-steps. In time-step 3 however (the follow-through phase) we note that a height of $4cm$ is, on average, preferred more than the greedy policy choice of $5cm$.

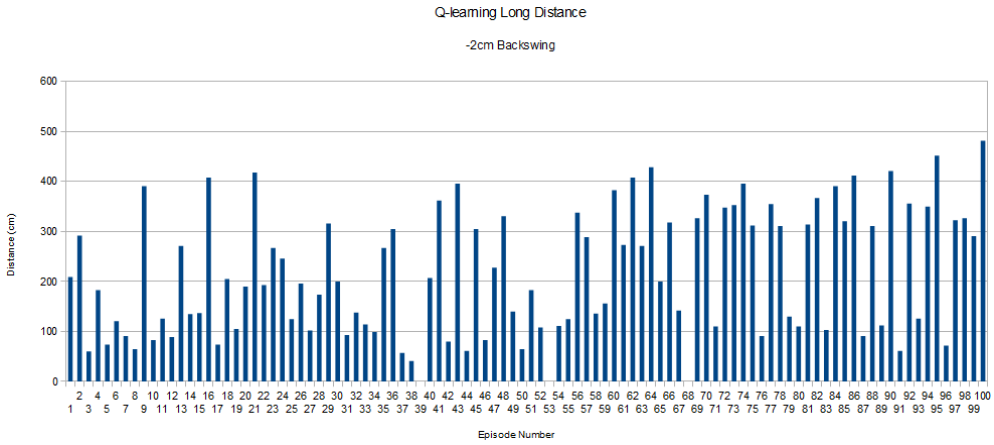

Figure 6.1: Resulting distances from maximum distance test with 2cm backswing

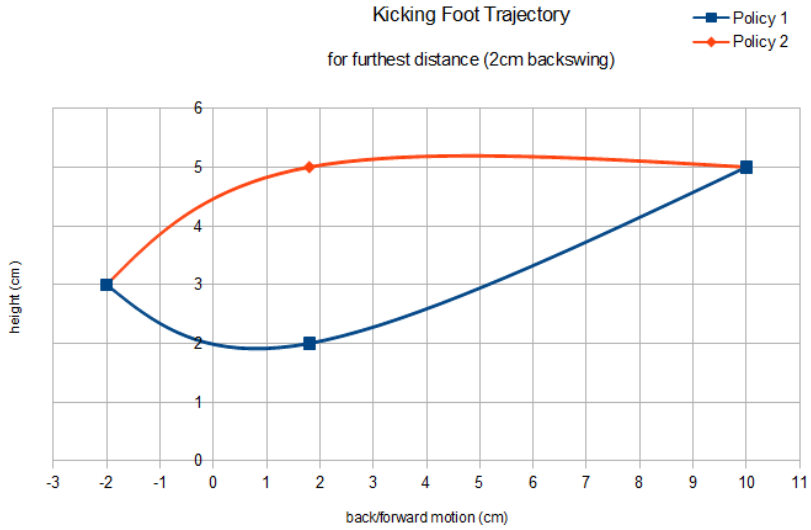

Figure 6.2: Trajectory of the foot motion for maximum distance with 2cm backswing

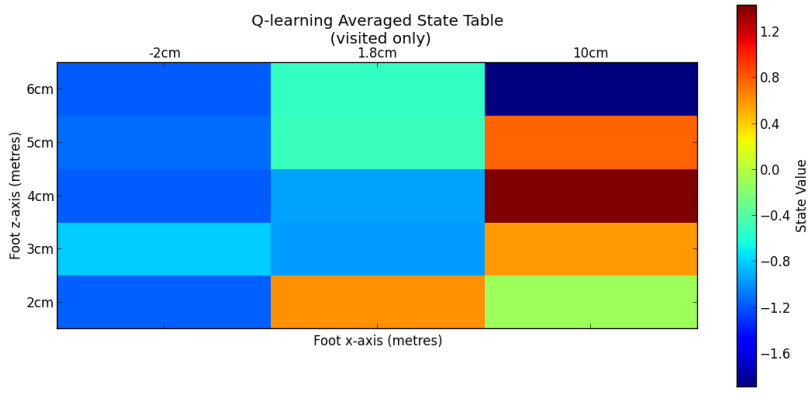

Figure 6.3: Heat map of state-value table for long distance, 2cm backswing

6.2.2 5cm Backswing

With the larger backswing of $5cm$ we observed a maximum distance of $513cm$ from greedy policy 1 and a maximum distance of $546cm$ from policy 2. From averaging these two policies, we get a maximum distance of $529.5cm$, a distance $6.5cm$ greater than the benchmark kicking motion.

From the policy trajectories in figure 6.5, we see both trajectories making contact at $4cm$ high on the ball ($0.5cm$ higher than the centre). The follow-through phases are more diverse, with policy 1 moving up by $1cm$, and policy 2 dropping by $1cm$. Policy 2 uses a lower backswing, level to the ball with a longer interpolation time of 0.2 seconds. We note that in episodes 87 and 88, a backswing of 0.1 seconds was explored, which resulted in a negative reward for bearing change. This episode ended with the state representing a $4cm$ follow through. This negative reward can be seen in the heat map (6.6) as the dark blue state. As heat maps are an average of all action-values, the heat map has been biased by this negative reward.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.1	0.1
2	0.2	0.1	0.1

Table 6.2: Policy interpolation times

A backswing of $5cm$ produced a further kicking motion in both tests, and also satisfied the distance criteria (beating the benchmark kick). The interpolation time was able to go as fast as possible at 0.3 seconds, but sometimes gained a negative reward for bearing change, when the robot used the fastest interpolation times. The robot did not fall in any of the episodes.

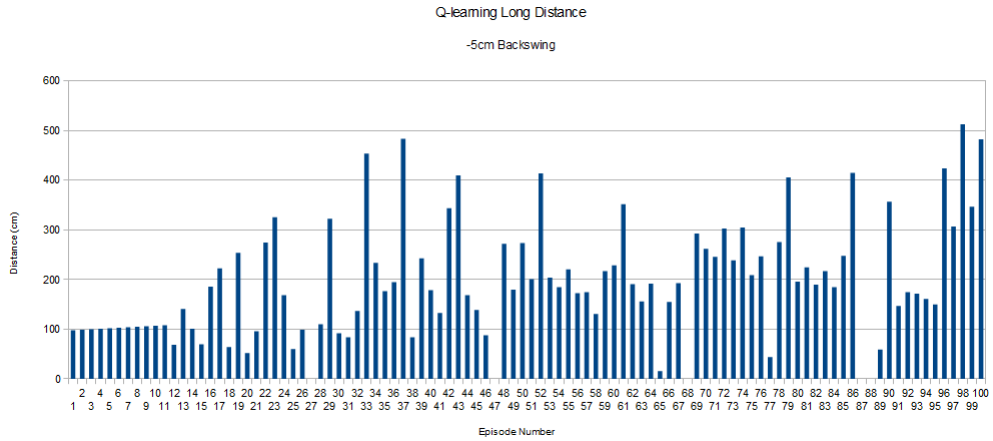

Figure 6.4: Resulting distances from maximum distance test with 5cm backswing

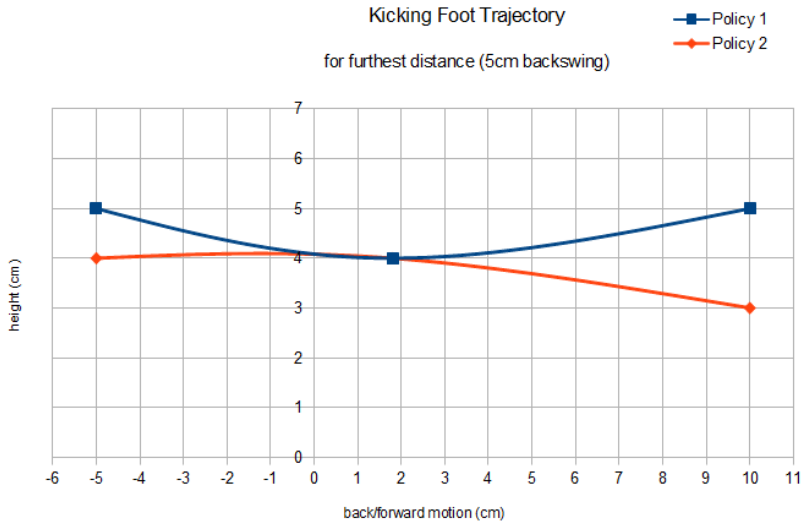

Figure 6.5: Trajectory of the foot motion for maximum distance with 5cm backswing

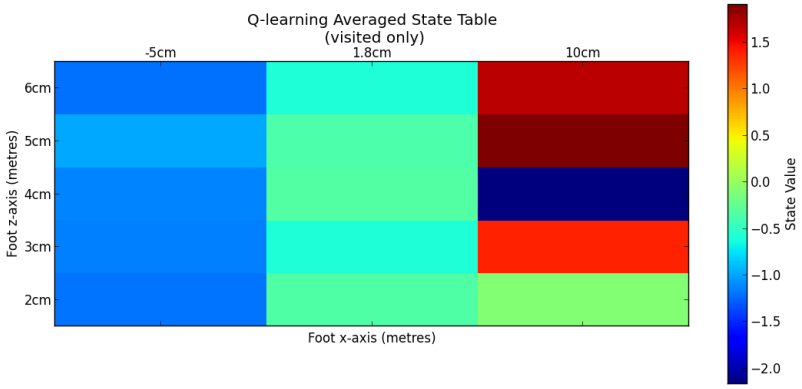

Figure 6.6: Heat map of state-value table for long distance, 5cm backswing

6.2.3 8cm Backswing

A backswing of $8cm$ produced a maximum distance of $532cm$ in the first test, and $526cm$ in the second. Greedy policy 1 follows a similar curved motion seen with the the $5cm$ backswing test, but with a higher backswing at $6cm$. Both tests produced a further distance than the benchmark of $532cm$. We observe again, as seen in both $2cm$ and $5cm$, that policy 2's curve is the inverse of policy 1. In Figure 6.8 we see the foot make contact no more than $1.5cm$ above the ball.

The kicking motions from both policies best the distance set by the benchmark and keep the robot stable. However in both tests the total execution time was 0.1 seconds longer than the policies found with a $2cm$ backswing. With an $8cm$ backswing the robot fell twice and received a penalty of -150 from changing bearing twice. This is almost twice the number of penalty-incurring policies than a $2cm$ backswing and $5cm$ backswing.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.1	0.2
2	0.1	0.1	0.2

Table 6.3: Policy interpolation times

From the heat map of policy 1, we see that the policy does not reflect the favoured state for the follow-through time-step, as it chooses the second most favoured state. The state-value scale does, however, reflect the number of penalty episodes that were incurred, with the highest valued state having a value only marginally greater than zero. In comparison, the highest valued state in the $5cm$ backswing had an average value of over 1.5.

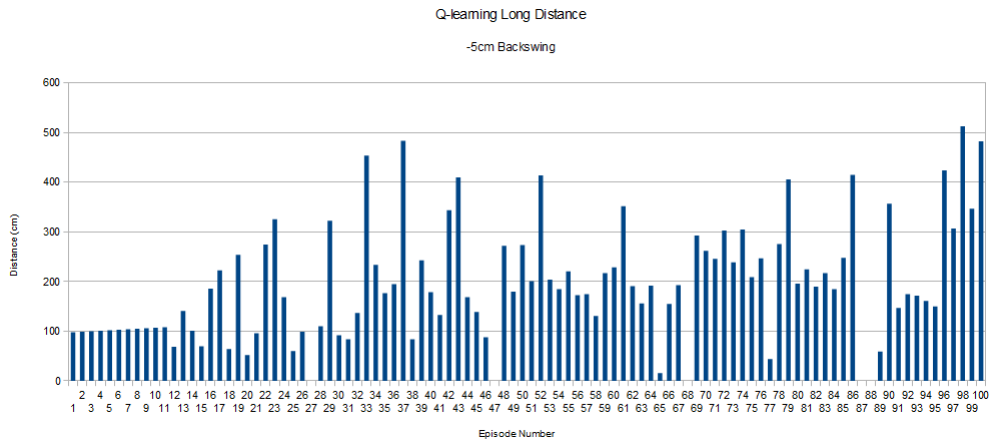

Figure 6.7: Resulting distances from maximum distance test with 8cm backswing

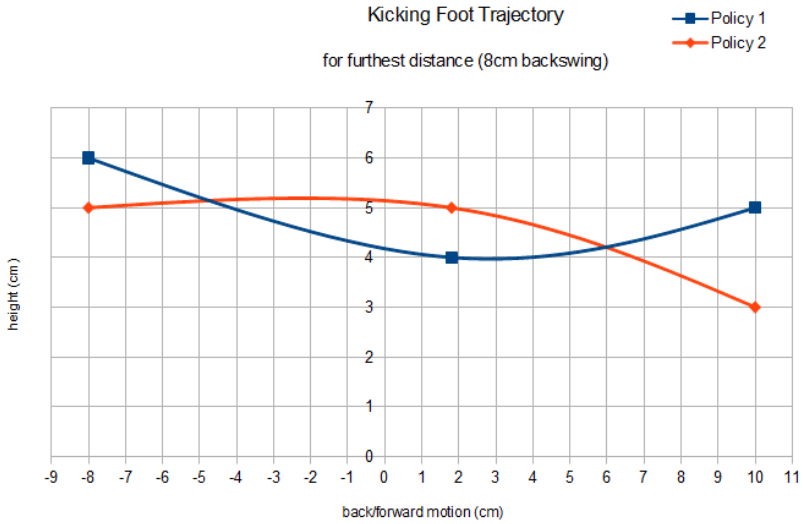

Figure 6.8: Trajectory of the foot motion for maximum distance with 8cm backswing

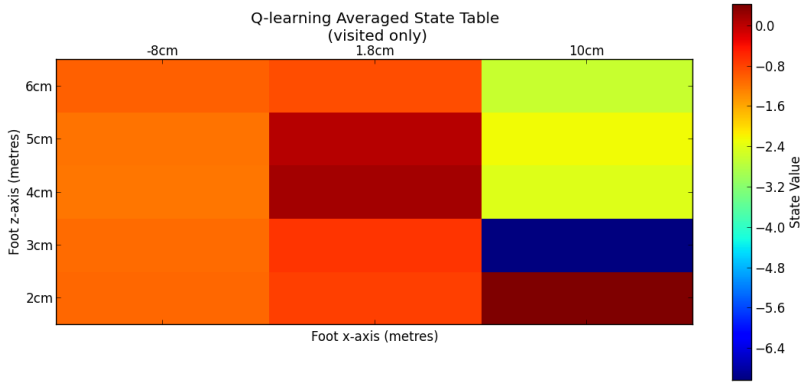

Figure 6.9: Heat map of state-value table for long distance, 8cm backswing

6.2.4 5cm Backswing, Extended Test

This experiment used the $5cm$ backswing, as the previous experiments proved it to be the most stable, the fastest, and to produce the furthest distances. This experiment ran with the same configuration as in the $5cm$ backswing experiment, but for 200 episodes and diagonally across the pitch (for more distance). First, we notice that a maximum distance of $573cm$ was achieved by greedy policy 2. Greedy policy 2 was also explored in test 1 at episode 81 (see Figure 6.11). This persisted as the greedy policy until receiving negative rewards in episode 169, 171, and 176. This can be seen as a large dip in episodes as the agent searched for a new greedy policy. The negative rewards were given for bearing changes in the robot. The bearing change did not affect the angular accuracy of the kick, but did see a drop in distance by almost a metre. In episode 192 of test 1, the final greedy policy was found, which produced a distance of $553cm$ while staying stable with no bearing change. The greedy policy was stable for 8 episodes.

We can see from Figure 6.11 that both policies intersect the ball at $4cm$ high, and only differ by $1cm$ in the backswing and follow-through phases. Both greedy policies have the minimum execution time possible.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.1	0.1
2	0.1	0.1	0.1

Table 6.4: Policy interpolation times

The heat map (Figure 6.12) shows very little variance in the values for the backswing states, but a clearly favoured contact point at $5cm$. We also see that $5cm$ and $6cm$ are the favoured states in the follow-through phase.

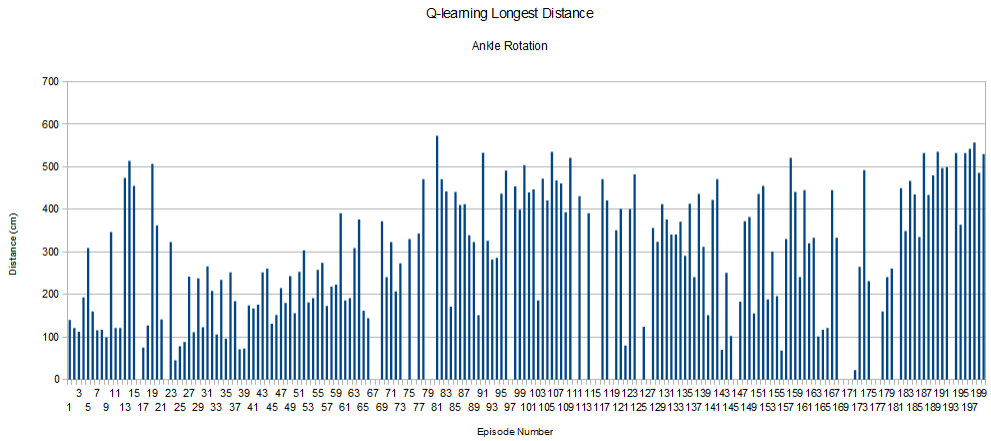

Figure 6.10: Resulting distances from the 200 episode, maximum distance test with 5cm backswing

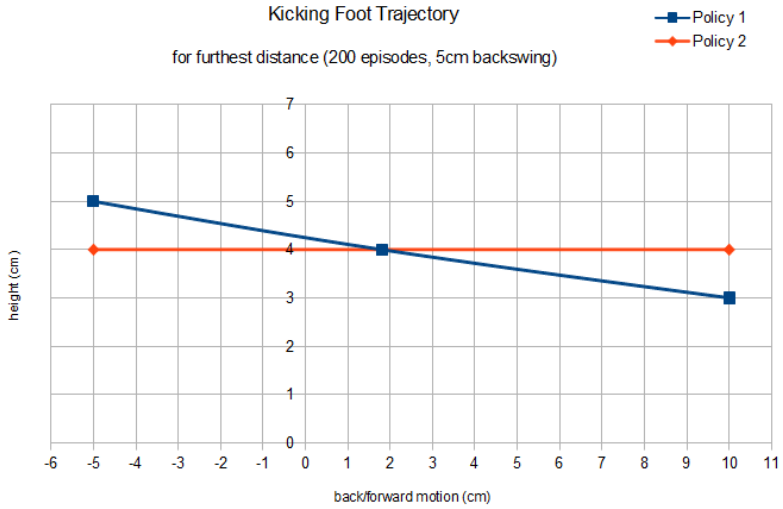

Figure 6.11: Trajectory of the foot motion for maximum distance with 5cm backswing

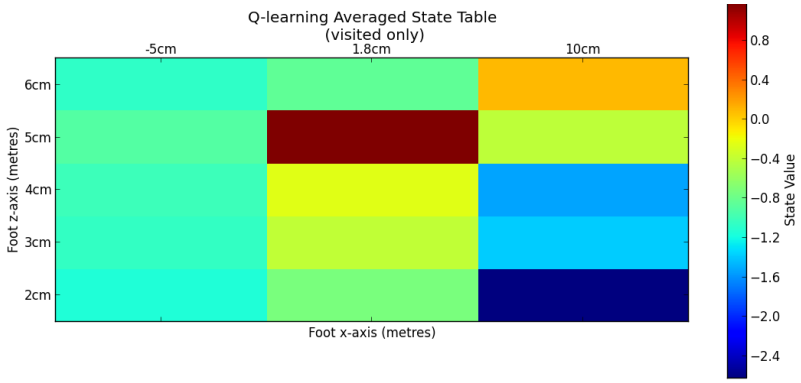

Figure 6.12: Heat map of state-value table for long distance, 200 episodes

6.3 Specific Distance

A $2cm$ backswing produced less momentum for all interpolation times, and achieved a distance of up to $482cm$. As such, it was used for the 1 metre and 2 metre experiments. The $5cm$ backswing produced similar average distances to the $8cm$ distance, while being more stable. The $5cm$ backswing was therefore used for the 3 and 4 metre tests.

6.3.1 1 Metre Kick

In Figure 6.13, we see that by episode 37, when the exploitation rate reached 25%, the agent's exploitations lead to low distance kicks, with only 12 further episodes resulting in a distance over $150cm$ (only 5 of which were further than $200cm$). The target of $100cm$ was met 4 times in test one. The greedy policy in test one was stable for 11 episodes, and in policy 2 the greedy policy was stable for 9 episodes. Policy 2 hits the ball high and follows through low, causing a slight backspin on the ball, preventing it from rolling too far. Policy 1 follows a similar curved motion as seen in the previous test results. The agent was penalised twice for not kicking and three times for changing bearing during the backswing.

Both policies managed to achieve exactly $100cm$ at least once during their exploitation. Thus, as the robot was stable, did not change bearing, and produced the specified distance in a shorter period of time than the original kicking motion ($0.8seconds$), both succeed in the optimisation criteria. From the two, policy 1 was more optimal as it had a shorter execution time.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.15	0.1	0.25
2	0.1	0.3	0.3

Table 6.5: Policy interpolation times

The heat map shows a clear high value for $2cm$ high at the contact point and $5cm$ for the follow-through phases. All backswing phases have a low average score due to the optimal policies for short kicks having a higher execution time.

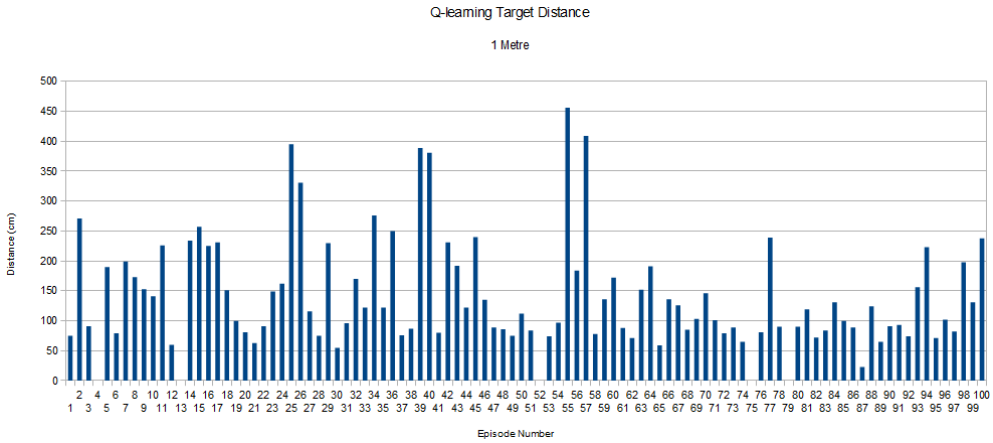

Figure 6.13: Resulting distances from the 1 metre reward function test

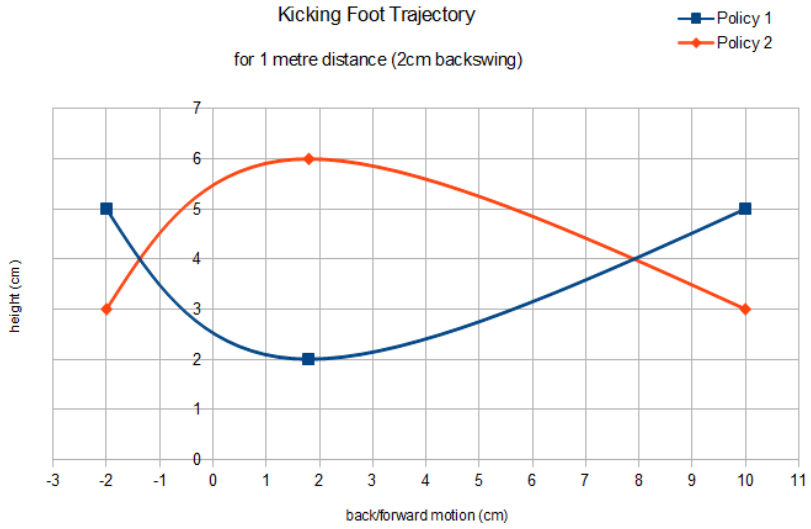

Figure 6.14: Trajectory of the foot motion for 1 metre distance

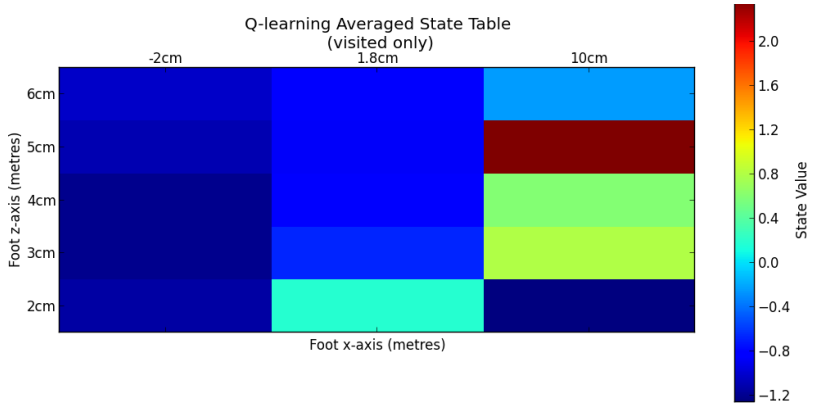

Figure 6.15: Heat map of 1 metre state-value table

6.3.2 2 Metre Kick

We see that by episode 71, the agent has enough information about the state-space to begin to exploit optimal policies, as seen in Figure 6.16. After episode 71, 19 of the remaining episodes place a kick within the $20cm$ top reward category. The closest value received was $199cm$ by greedy policy 1.

The greedy policy was stable for 13 episodes in test one, and for 11 in test two. Both policies share a similar trajectory (see 6.17), and time interpolations (see 6.6), differing only in the backswing. As can be seen from the heat map (6.18), all backswing states were similarly valued after 100 episodes. Both kicking motions resulting from the policies kept the robot stable and accurate during the kicking motion, but policy 1 took 0.5 seconds less time to complete.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.15	0.15
2	0.15	0.15	0.15

Table 6.6: Policy interpolation times

Heat map 6.18, as in the previous two tests, shows very little separation in the backswing state values. $3cm$ is preferred in for the contact point, and $6cm$ is the preferred follow-through by an average value of 2.5. This is the largest valued state seen in all tests thus far.

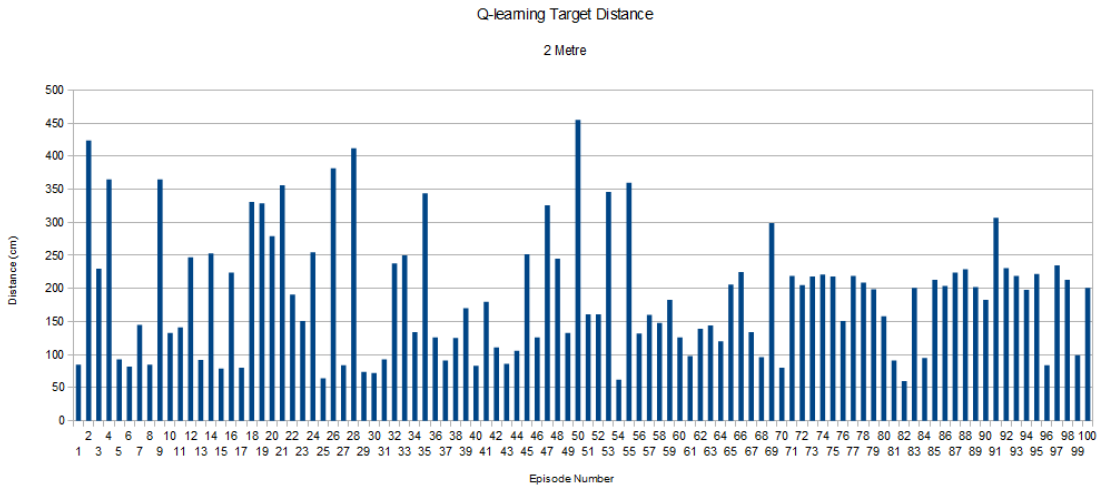

Figure 6.16: Resulting distances from the 2 metre reward function test

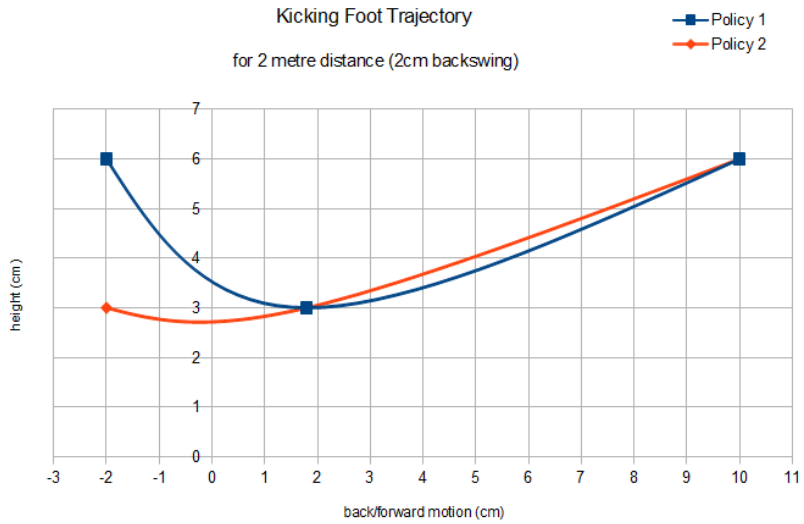

Figure 6.17: Trajectory of the foot motion for 2 metre distance

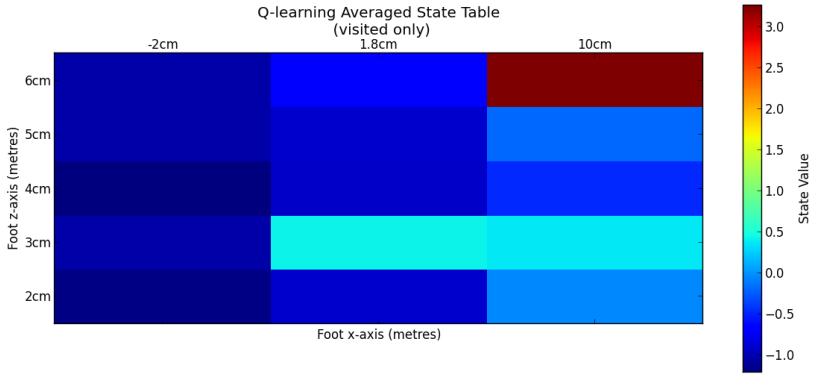

Figure 6.18: Heat map of 2 metre state-value table

6.3.3 3 Metre Kick

Similarly to the 2 metre distance experiment, policies start to gain the highest level reward by episode 74 (Figure 6.19). In the 3 metre experiment we begin to see a larger variance between distances, even from the same policy. The greedy policy was stable for 29 episodes in test 1, and 21 in test 2. The closest distance found was $400cm$ in test 2 by policy 2.

As seen in the trajectory chart in Figure 6.20, both tests produced the same key-frames for the greedy policy, but with differing interpolation times (see 6.7) The policy follows a curved motion as seen in previous experiments, with a contact height of $2cm$. As policy 1 executed faster, uses the same key-frames as policy 2, and was stable for 29 episodes, policy 1 best meets the optimisation criteria.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.2	0.1
2	0.1	0.2	0.15

Table 6.7: Policy interpolation times

The greedy policy from both tests complements the average state-values observed in heat map 6.21. We note that the greedy follow-through state with a height of $3cm$ holds a value very close to, but less than the value of, the state with a height of $4cm$.

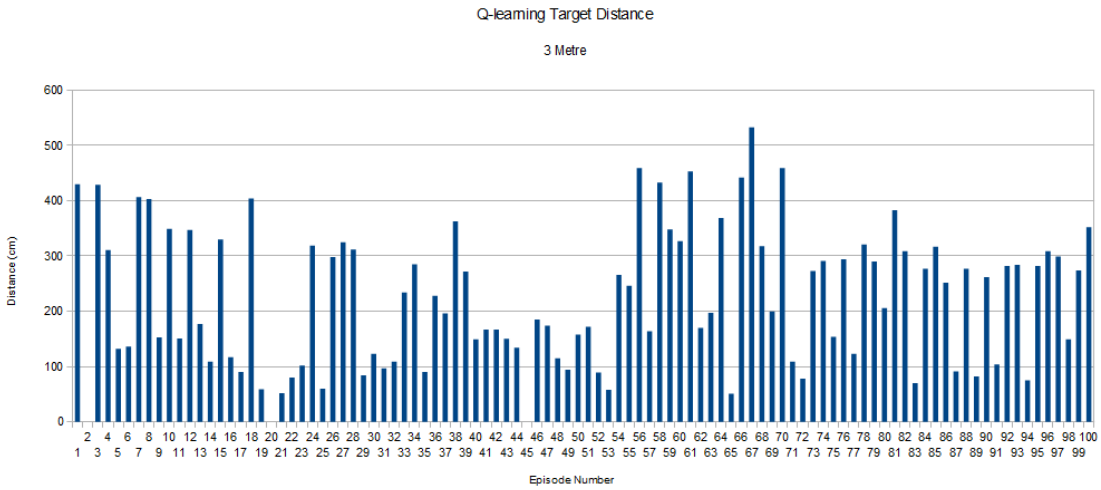

Figure 6.19: Resulting distances from the 3 metre reward function test

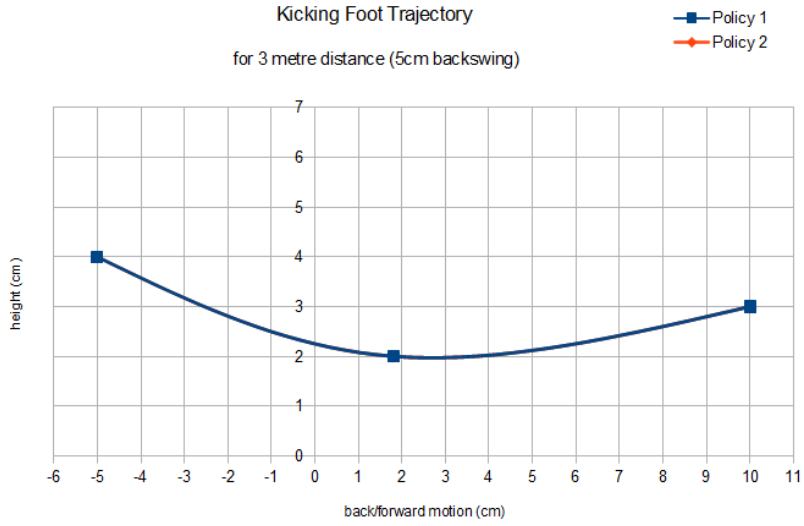

Figure 6.20: Trajectory of the foot motion for 3 metre distance

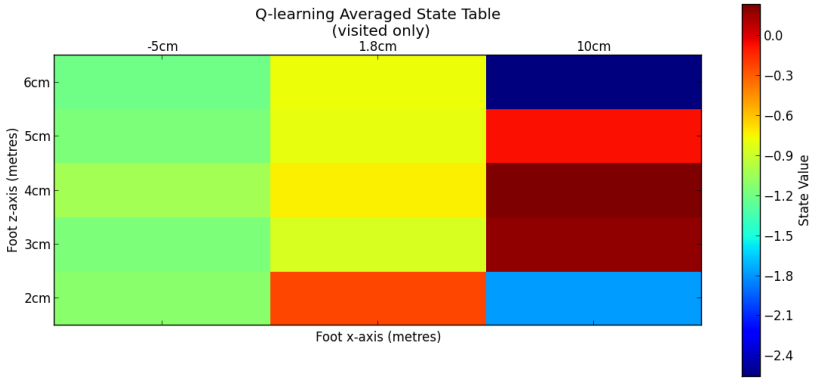

Figure 6.21: Heat map of 3 metre state-value table

6.3.4 4 Metre Kick

In test 1, the greedy policy was stable for 42 episodes after kicking a distance in the top reward category 4 times. The best distance from both tests was found from policy 2, with a distance of $402cm$. In Figure 6.22 we see a large variance in the distances, even from those of the final 20 episodes that are fully exploiting. Nevertheless, we still see convergence towards the goal distance, as observed in all of the specified distance experiments.

The trajectory of policy 2 is similar to that seen for the furthest distance in the 200 episode experiment - flat through $4cm$ high, except for a slight lift for the follow-through phase. Both motions resulting from the policies execute in 0.4 seconds, which is half the time of the original kicking motion.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.1	0.2	0.1
2	0.1	0.1	0.2

Table 6.8: Policy interpolation times

From heat map 6.24 we observe the low average values for all states, as depicted by the value range. When observing the distances achieved in 6.22, it is clear that the low average value for all states comes from the majority of episodes not getting close enough to the target distance.

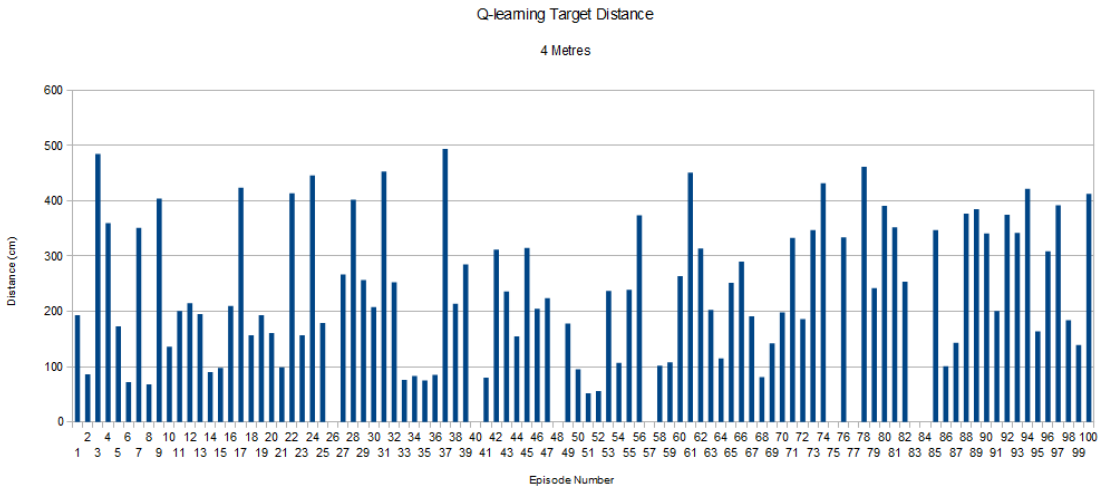

Figure 6.22: Resulting distances from the 4 metre reward function test

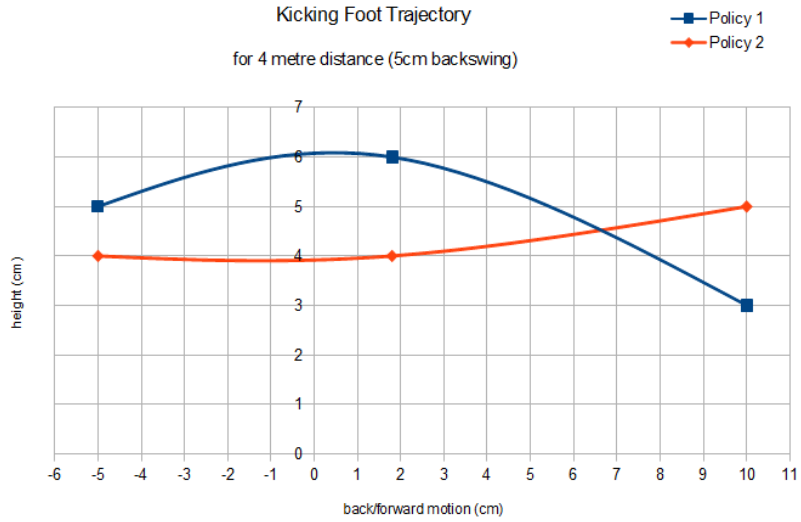

Figure 6.23: Trajectory of the foot motion for 4 metre distance

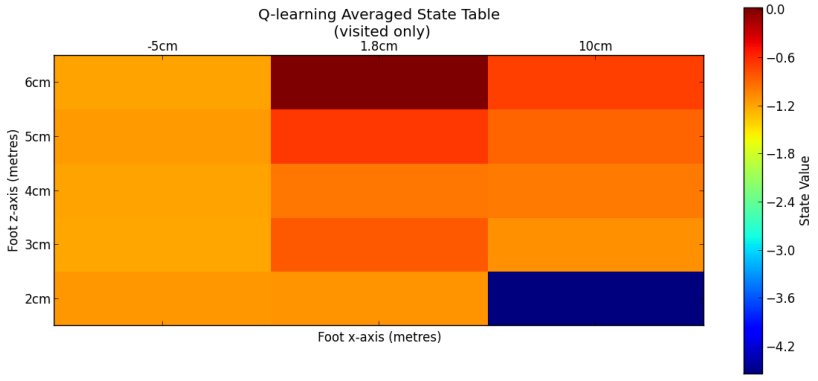

Figure 6.24: Heat map of 4 metre state-value table

6.4 Ankle Rotation

In this experiment the robot fell over 80 times, was penalised for bearing change 8 times, and did not kick 4 times; the robot kicked the ball only 8 times. It achieved a maximum distance of 420cm, using the greedy policy depicted by the trajectory in Figure 6.25, and interpolation times in 6.9. With an execution time of 0.6 seconds, the ankle kicking motion takes twice as long as all of the other maximum distance greedy policies found. When an execution time was shorter than that of the greedy policy, the hip motor locked, making the knee and foot joints kick out backwards; this is how the majority of falls occurred. From the trajectory curve (Figure 6.25) we see that the foot height increase in each of the forward phases. However, when the ball was kicked without a fall, the ball went straight ahead in the intended direction.

	Time-step Interpolation time (sec)		
Policy	1	2	3
1	0.3	0.15	0.15

Table 6.9: Policy interpolation times

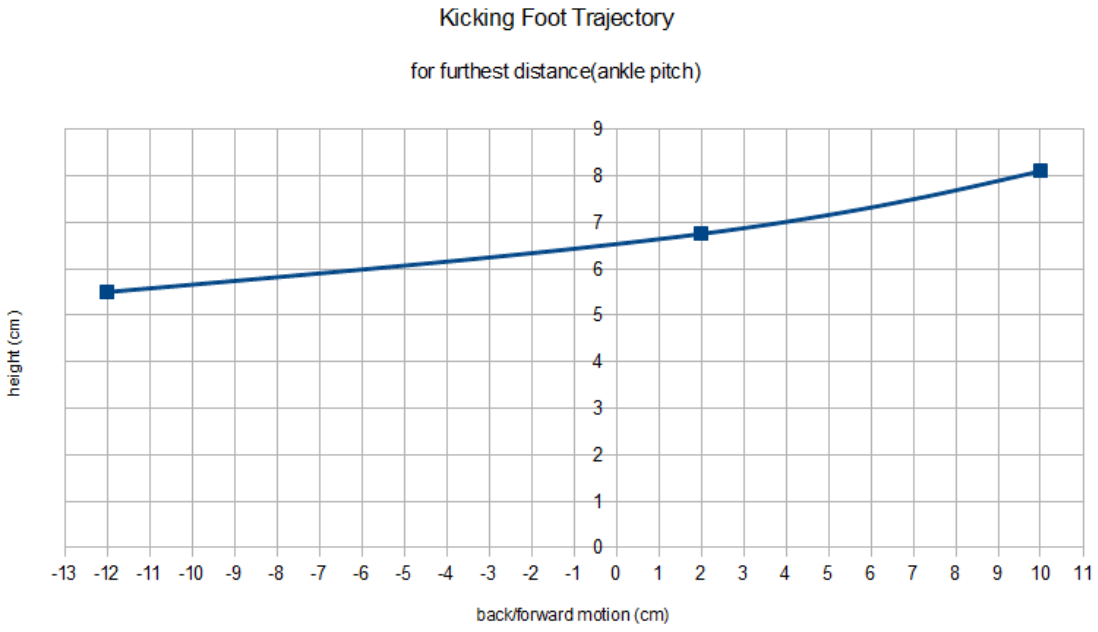

Figure 6.25: Trajectory of the foot motion for maximum distance, with ankle rotation

6.5 Optimised Kicks

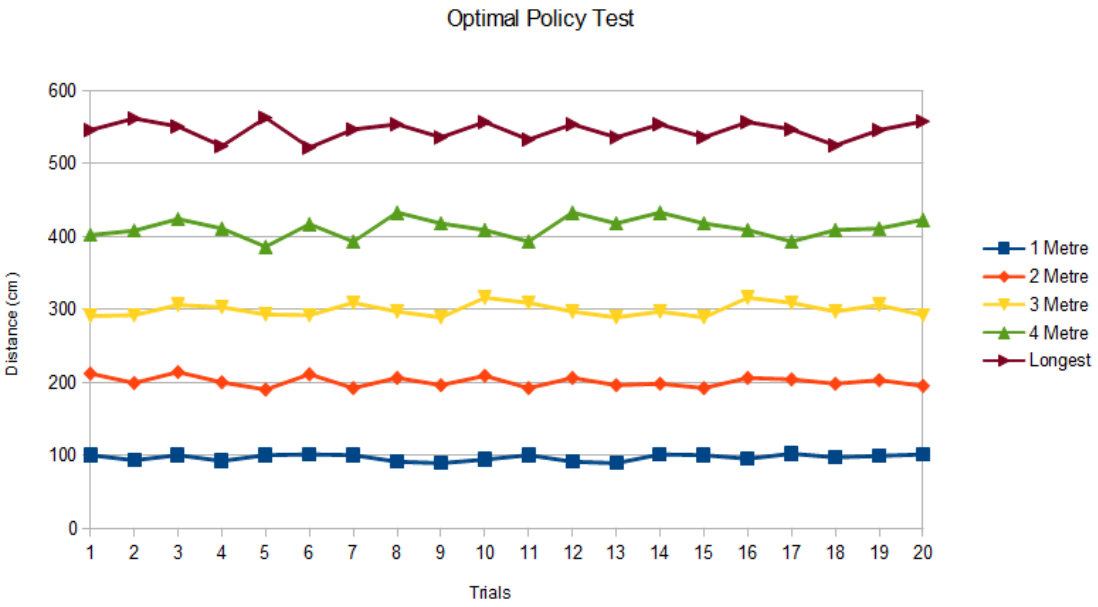

Figure 6.26: The optimal kicking motions for each distance, tested 20 times

For the greedy policies found in each of the five distance tests, we state the standard deviation and variances found when executing 20 straight kicks (Table 6.10). In Figure 6.26 we see the distances achieved by each policy in the 20 trials. We observe that the standard deviation (σ) of the kicking motions increases with the desired distance. Since normally distributed samples characteristically differ by 2σ from the average 95% of the time [23], in the worst case scenario, a kick will be within $27.68cm$ of the target.

Policy Type	Standard Deviation (cm)	Average (cm)	Variance (cm)	Trials within SD
1 Metre	4.38	96	19.2	60%
2 Metres	7.63	202.9	58.29	50%
3 Metres	8.69	298.8	75.56	60%
4 Metres	13.32	410.1	177.29	50%
Maximum	13.84	546.2	191.56	50%

Table 6.10: The resulting standard deviation and averages for each greedy policy

6.6 Angled & Dynamic Kicks

The reach limits of the leg limited the possible lateral movements possible to $2.3cm$ in either direction (in and out). $1cm$ needed to be removed from the reach of the follow-through step, making the most forward reach of the kick $9cm$. Even with this newly enforced limit, all kicking motions were unaffected when testing non-deviated motions (no lateral update). The kicking motions were also able to produce near identical distances for non angled kicks for $1cm$, $2cm$, and $2.3cm$ of inward lateral modifications. A similar pattern was also noted for an outward lateral update for $1cm$ and $2cm$. However, as the kicking motions were updated laterally outward for $2.3cm$, an average drop of $53cm$ distance was seen for the 4 metre kick, and a large average decrease of $58cm$ for maximum distance. The kicking module was able to accurately update the y-value of the kicking motion and was able to correct the foot to kick with the same angular accuracy seen in all test in which the ball was placed directly in front of the foot.

	Distance for Angled Kicks (°)					
Policy Type	$-45°$	$-30°$	$-20°$	$20°$	$30°$	$45°$
1 Metre	54	69	96	91	71	63
2 Metres	113	167	182	187	162	129
3 Metres	143	225	284	285	264	156
4 Metres	165	303	354	372	322	194
Maximum	218	392	492	486	376	232

Table 6.11: Angled policy update

The distance for each angled kick can be seen in Table 6.11. There is a large drop-off in distance observed between 30° and 45°, with some kicks not even reaching 50% of their potential for straight kicks. At an angle of 20° the kicks are still able to achieve distances that fall within the variance for for the kicking motions when unmodified.

7. Discussion

The results from the final trials, with the kicking motions described by greedy policies learnt through reinforcement learning, proves the hypothesis that kicking motions can be optimised to satisfy multiple criteria through reinforcement learning. The final policies for each experiment fully meet the criteria of kicking the set distance in as little time as possible, as accurately as possible, while staying stable. Through the use of heat maps, we have shown that the average state-values start to converge with little experience. Qlearning with an ϵ-greedy on-line policy found high valued state-action pairs relatively quickly. In some cases, this took a while longer to be reflected back to the first time-step, causing a delay in early states realising the potential future rewards. This could be avoided in future by using a one-step eligibility trace (see section 3.4.1).

From the backswing experiments, it was discovered that a large backswing did not improve the kicking distance over a smaller backswing. However, if the backswing was too small, such as the case with a $2cm$ backswing, the maximum distance achievable decreases. There are two possible explanations for these findings. The first is that the robot's motors could not build up enough speed within $2cm$. This theory is disproved by the benchmark kicking motion, which had no backswing and still produced a maximum distance of $523cm$. The second is that of unchanged minimum interpolation times. With a smaller backswing, the robot should, theoretically, be able to use shorter interpolation times. But, as the minimum interpolation time was unchanged between experiments, the motors would be rotating at a slower speed to travel between the backswing and the contact point in the same time as the larger backswings. This could also explain why we see a more curved trajectory in the $2cm$ experiments in comparison to the $5cm$ and $8cm$ backswings; the agent is increasing the distance to the contact point with a curved trajectory, which in theory would move the motors faster. The interpolation time was not changed between experiments for one reason; interpolation times less than 0.1 seconds created too much momentum, causing the robot to become unstable.

With an extended number of experiences, the agent was able to produce a kicking

motion capable of travelling further than any other team (B-Human being the closest with an average only $3cm$ shorter). The kicking motion was a linear motion, travelling through the centre of the ball. This was the expected follow-through motion. With all of the maximum distance policies using the minimum execution time, future experiments should lower this limit to discover whether even faster motions are possible. As discussed above, this should be possible at least with a $2cm$ backswing.

For set distances, the reinforcement learning agent was able to find suitable kicking motions; given their fulfilment of the multiple criteria, it can be argued that these must be optimal policies, given the states that describe the environment. The majority of policies produce what can be described as a pendulum-like trajectory. Given that the robot's leg is connected to a central pivot (the hip), this would be the expected trajectory if the hip motor was being fully utilised. In 5 out of the 9 different experiments, one of the greedy policies resembled an inverted pendulum of sorts. This is a very unexpected outcome when the mechanics of a leg are considered. With an inverted pendulum-like motion, all motors would be running at speeds a fraction of their capability, as they are extending the limb outwards, away from the body in a pushing motion. This suggest that the robot was able to achieve similar distances from a pushing motion with the legs, as a swinging motion. An alternative (and more likely) explanation is that the agent received a high reward for a motion that included a high contact state, and after exploiting this state, found higher returns when the foot was lowered again for the follow through, rather than staying high. This seems a logical explanation for the unexpected inverted pendulum trajectories.

The ankle experiment, in many respects, produced expected results. It was expected that the leg would have limited range to utilise all of the motor, but it was not expected to be so difficult to manoeuvre. The main problem caused by pitching the foot down, was that the knees of the robot got very close to each other. To counter this, the hips needed to be twisted, and as a coupled joint this involved the whole robot rotating. This rotation, only exacerbated the excessive movement needed to lift the foot high enough, caused the robot to be very unstable. We must note, however, that during the preliminary testing to find the correct joint rotation, using the top of the foot to kick produced consistently straight kicks. It is a shame therefore that this increased accuracy was unable to

be demonstrated in a stable and powerful kick.

In many of the experiments that we observed, the kicking motions made contact with the ball at $4cm$. Throughout all of the final testing, we noticed a slight lean in the robot's hip motor while balancing over one leg, making the robot's foot slightly lower than the desired height. As such, the robot's foot could actually have been making contact at the equator of the ball at $3.5cm$. This is one of the disadvantages of testing with physical robots - fatigue.

The final tests of each greedy policy provided very conclusive results, with each having a very low standard deviation. Considering that the kicks were performed on the pitch while kicking over the centre lines, an average deviation (over all 5 kicks) of $19.87cm$ for 95% of the kicks, provides enough evidence that the kicking motions satisfy the given criteria.

The final kicking module, using the 5 kicking motions, was able to correct the latitudinal position of each key-frame for the selected motion without affecting stability or accuracy. The distance was slightly affected, the further the leg had to extend outwards. The kicking module, however, did not produce great distances for angled kicking motions. The larger the angle, the less power was put behind the ball. This could be remedied by changing the implementation of the angled trajectory. Currently, the kicking line is offset, to make contact with the side of the ball. This was implemented because it was the only possible option with the limited latitudinal range of the foot at this height. By lowering the hip, the foot would be able to reach further, allowing a change in the actual trajectory as described in [35].

From the results, we see that our greedy policies do in fact meet the criteria set. We can therefore conclude that we were able to balance the priorities of multiple objectives to gain satisfactory results. Our research project is therefore complete, having met the original aims and objectives. Let us now go on to discuss future work in this area.

8. Future Work

8.1 Simulators and Search-spaces

In the late stages of writing this paper, Alderbaran released a new simulator, one which would allow seamless use of NAOqi and environment manipulation: Webots for Nao. This simulator is built on the well known, and widely used, Webots simulator. It was built specifically for the sole purpose of testing the Nao. Unfortunately, a stable release was not ready for use in this project, but any future work should certainly involve the use of this simulator. Although still in its infancy, the simulator offers very accurate models of the Nao robot, and uses the proven physics engine of Webots.

The learning agent and environment builder implemented for this research has been developed with the potential to run with as many states, interpolation times, and time-steps as the user wishes to test for. With the Webots for Nao simulator, the learning agent's true potential could be realised very quickly by increasing the detail of the environment (more states etc.) and running a simulation for thousands of iterations. The results would either confirm that we are able to find optimal policies in a limited number of episodes on the robot, or find even better policies, capable of being faster and more powerful.

To prevent the need for a large number of experiences, a future implementation of the work in this paper should investigate the use of value function approximation. The current implementation of a kicking motion holds duplicated state-action pairs depending on the previous state. By using a method such as gradient descent [51], these similar state-action pairs can be generalised. This allows multiple states to gain an estimated value without being visited.

8.2 Different Kicking Motions

The next step for reinforcement learning with kicking motions should be to optimise for different styles of kicks, such as a side-sweeping kick using the sides

of the foot, as already used by B-Human, Austin Villa, and other teams [25, 34]. Another kick style discussed in the related work section is that of the walking kick. Reinforcement learning could be used to optimise the parameters discussed in Austin Villa's slow walk [34] to find a powerful kick that can be utilised while on the move. An alternative solution would be to have a collection of policies which describe different kicking motions, depending on the phase of a walking motion. This would allow a kick to be executed at any point during a robot's walk. This would make the Nao robots more adaptable to dynamic situations, such as those experienced in RoboCup.

There is potential to increase the kicking module designed in this research to dynamically update on-line; using the vision system to update the key-frames in real time as currently implemented by B-Human [25]. For a more precise kicking distance, there is potential to research morphing between motions. Using two motions, one could take the key-frames of corresponding phases and find a key-frame between them, by taking the difference and weighting it by the desired distance. The current module could also be improved to expand the reaching limits of the kicking leg by dynamically updating the weight distribution of the robot. Currently, the kicking leg is able to extend laterally by $2.3cm$. This could be improved by lowering the hips when the kick must extend further, giving the robot even more potential to dynamically update kicks without repositioning.

8.3 Balancing Module

The kicking module designed in this research could further be improved by using a balancing module, as described in [46]. Although proven to increase stability from external influences while on one leg, there is no evidence to demonstrate the effectiveness of a balancing module while executing a kick. As the Alderbaran Nao robot has smooth soles and fast moving actuators, the momentum gained from a kicking motion can often throw the robot off-balance, as observed many times during this research. With a dynamic stability module, there is the potential to limit the effects of momentum and thus improve the balance of the robot during kicks.

9. Conclusion

From the resulting kicking module, implemented using the kicking motions found through reinforcement learning, we can conclude that reinforcement learning is a valuable method for learning optimal solutions for multi-objective problems. We successfully described a kicking motion as a finite Markov Decision Process, which provided the means for the problem to be solved using reinforcement learning. We have shown how solutions to complex tasks can be achieved in a limited amount of experience on a physical robot, without a model of the environment, and in the absence of a simulator.

The resulting kicking motions from each experiment produced optimal policies with regards to the criteria set in the objectives: kick a ball a set distance as quickly and as accurately as possible, while remaining stable. The optimal kicking motions, for most tests, resemble an inverted pendulum-like motion, similar to that used by humans. We also conclude that a large backswing is not required to create the required impact on the ball for a long distant kick; on the contrary, this made the robot less stable and increased the execution time of a kicking motion.

The kicking motions defined by the greedy policies were successfully used as the basic key-frames for a dynamic kicking module. The kicking module uses the current ball location, a goal distance, and a kicking angle to calculate the modifications required to produce a kick in the correct direction and distance, without the need to reposition the robot.

With the new Webots for Nao simulator, there is scope for the conclusions from this paper to be tested extensively. There is also vast potential to extend this research further.

Appendix A. Football Rules

As in human football, the robots can score points by scoring a goal in the opponent's net. The match lasts for 20 minutes, 10 minutes per half. If no winner is decided by score in this time, the game turns to a penalty shoot-out, in which both teams take 3 shots at the goal from the penalty spot. If no winner is decided from the penalty shoot out, the penalties continue into sudden death (the first to miss a goal loses). The image below shows the dimensions (in centimetres) of the RoboCup pitch.

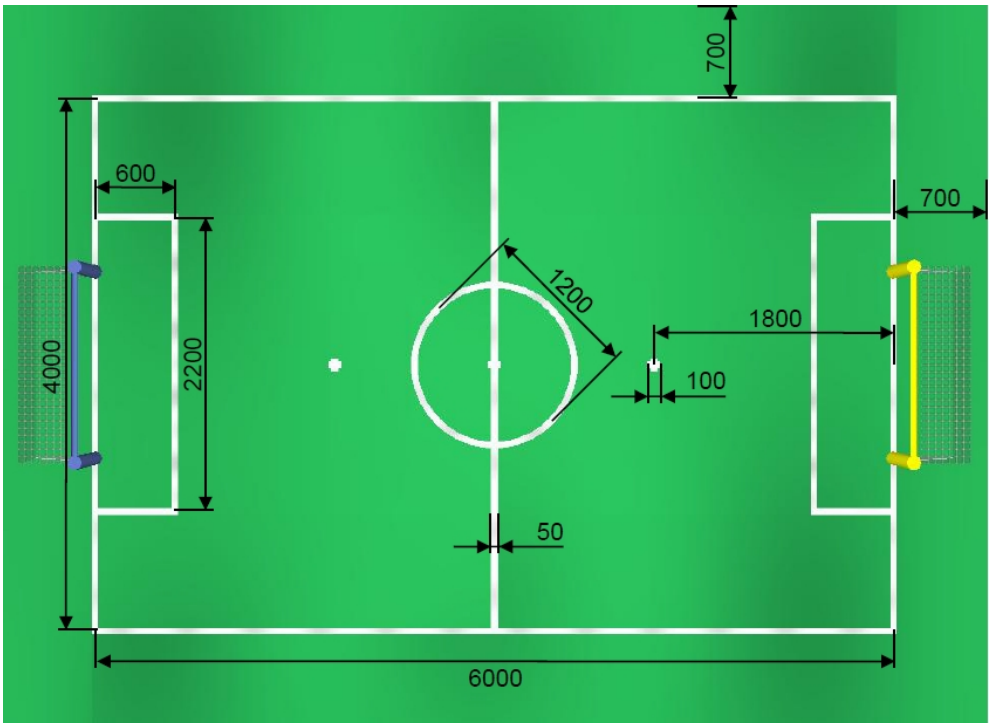

Figure A.1: The pitch dimensions, in centemetres

Bibliography

[1] P. Kopacek, "Advances in robotics," *Computer Aided Systems Theory–EUROCAST 2005*, pp. 549–558, 2005.

[2] W. G. Walter, "An imitation of life," *Scientific American*, vol. 182, no. 5, pp. 42–45, 1950.

[3] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, "The intelligent asimo: System overview and integration," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, pp. 2478–2483, IEEE, 2002.

[4] R. S. Sutton, "Introduction: The challenge of reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 225–227, 1992.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.

[6] O. Kaynak, "Recent advances in mechatronics," *Robotics and Autonomous Systems*, vol. 19, no. 2, pp. 113–116, 1996.

[7] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "The nao humanoid: a combination of performance and affordability," *CoRR, vol. abs/0807.3223*, 2008.

[8] A. Valtazanos, E. Vafeias, C. Towell, M. Hawasly, S. Behzad, P. I. Tabibian, S. Wilson, D. Wren, J. A. Tobin, S. Ramamoorthy, *et al.*, "Team edinferno description paper for robocup 2012 spl," *Only available online: http://wcms.inf.ed.ac.uk/ipab/robocup/research/EdinfernoTDDoc2012.pdf*.

[9] T. Hester, M. Quinlan, and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2369–2374, IEEE, 2010.

[10] H. Kitano and M. Asada, "The robocup humanoid challenge as the millennium challenge for advanced robotics," *Advanced Robotics*, vol. 13, no. 8, pp. 723–736, 1998.

[11] S. Behnke, M. Schreiber, J. Stuckler, R. Renner, and H. Strasdat, "See, walk, and kick: Humanoid robots start to play soccer," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pp. 497–503, IEEE, 2006.

[12] R. T. Committee *et al.*, "Robocup standard platform league (nao) rule book," 2009.

[13] S. Barrett, K. Genter, T. Hester, M. Quinlan, and P. Stone, "Controlled kicking under uncertainty," in *The Fifth Workshop on Humanoid Soccer Robots at Humanoids 2010*, 2010.

[14] C.-P. Chou and B. Hannaford, "Static and dynamic characteristics of mckibben pneumatic artificial muscles," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 281–286, IEEE, 1994.

[15] T. Fukaushima, Y. Kuroki, and T. Ishida, "Development of a new actuator for a small biped walking entertainment robot," in *Power Electronics, Machines and Drives, 2004.(PEMD 2004). Second International Conference on (Conf. Publ. No. 498)*, vol. 1, pp. 126–131, IET, 2004.

[16] A. Robotics, "Nao h25."

[17] D. Mescheder, "Learning to walk a self optimizing gait for the nao," 2011.

[18] A. Goldenberg, B. Benhabib, and R. Fenton, "A complete generalized solution to the inverse kinematics of robots," *Robotics and Automation, IEEE Journal of*, vol. 1, no. 1, pp. 14–20, 1985.

[19] A. Robotics, "Naoqi inverse kinematics."

[20] "Bézier curves."

[21] J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips, *Introduction to computer graphics*, vol. 55. Addison-Wesley Reading, 1994.

[22] R. Brunn, U. Düffert, M. Jüngel, T. Laue, M. Lötzsch, S. Petters, M. Risler, T. Röfer, K. Spiess, and A. Sztybryc, "Germanteam 2001," *RoboCup 2001: Robot Soccer World Cup V*, pp. 215–250, 2002.

[23] J. Müller, T. Laue, and T. Röfer, "Kicking a ball–modeling complex dynamic motions for humanoid robots," *RoboCup 2010: Robot Soccer World Cup XIV*, pp. 109–120, 2011.

[24] H. L. Akın, T. Meriçli, E. Ozkucur, C. Kavaklıoglu, and B. Gökçe, "CerberusâĂŹ 10 team description paper," 2010.

[25] T. Röfer, T. Laue, J. Müller, A. Burchardt, E. Damrose, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. J. de Haas, *et al.*, "B-human team report and code release 2011," *unpublished, available at*, 2011.

[26] S. P. Hohberg, "Interactive key frame motion editor for humanoid robots,"

[27] A. Robotics, "Naoqi framework."

[28] R. Zhang and P. Vadakkepat, "An evolutionary algorithm for trajectory based gait generation of biped robot," in *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2003.

[29] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3, pp. 2619–2624, IEEE, 2004.

[30] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[31] L. N. Kanal and V. Kumar, *Search in artificial intelligence*. Springer-Verlag, 1988.

[32] H. W. CJC, "Learning from delayed rewards," *Cambridge University, Cambridge, England, Doctoral thesis*, 1989.

[33] B. Thomas, "Embedded robotics: Mobile robot design and applications with embedded systems," *Berlin and Heidelberg: Springer-Verlag*, 2003.

[34] S. Barrett, K. Genter, Y. He, T. Hester, P. Khandelwal, J. Menashe, and P. Stone, "Ut austin villa 2012: Standard platform league world champions," *Proceedings of the RoboCup International Symposium 2012*, 2012.

[35] M. Sridharan, "Austin villa 2011: Sharing is caring: Better awareness through information sharing," 2011.

[36] A. Ratter, B. Hengst, B. Hall, B. White, B. Vance, D. Claridge, H. Nguyen, J. Ashar, S. Robinson, and Y. Zhu, "runswift team report 2010 robocup standard platform league," *Only available online: http://www. cse. unsw. edu. au/˜ robocup/2010site/reports/report2010. pdf*, 2010.

[37] J. Boedecker and M. Asada, "Simspark concepts and application in the robocup 3d soccer simulation league," in *Proceedings of SIMPAR-2008 Workshop on The Universe of RoboCup simulators, Venice, Italy*, pp. 174–181, 2008.

[38] Y. Xu and H.-D. Burkhard, "Narrowing reality gap and validation: Improving the simulator for humanoid soccer robot," *Concurrency, Specification and Programming CS&P*, 2010.

[39] R. Ferreira, L. P. Reis, A. P. Moreira, and N. Lau, "Development of an omnidirectional kick for a nao humanoid robot," in *Advances in Artificial Intelligence–IBERAMIA 2012*, pp. 571–580, Springer, 2012.

[40] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[41] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, no. 3, pp. 283–302, 1997.

[42] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3, pp. 3030–3035, IEEE, 2004.

[43] L. Przytula, "On simulation of nao soccer robots in webots: A preliminary report," *http://csp2011.mimuw.edu.pl/proceedings/PDF/CSP2011420.pdf*, 2011.

[44] A. Robotics, "Choregraphe documentation."

[45] A. Robotics, "Force sensor resistor documentation."

[46] S. Czarnetzki, S. Kerner, and D. Klagges, "Combining key frame based motion design with controlled movement execution," in *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 58–68, Springer, 2010.

[47] J. Wilhelms and A. Van Gelder, "Efficient spherical joint limits with reach cones," *Apr*, vol. 17, pp. 1–13, 2001.

[48] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. Bradford Books, 2005.

[49] A. Robotics, "Force sensor resistor documentation."

[50] G. Joseph, "Newton's laws of motion," *Doing Physics with Scientific Notebook: A Problem-Solving Approach*, pp. 157–178.

[51] L. C. Baird III, *Reinforcement learning through gradient descent*. PhD thesis, Brown University, 1999.