Volume 33 (1), pp. 1–20
http://orion.journals.ac.za

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)

New reinforcement learning algorithm for robot soccer

M Yoon* J Bekker† S Kroon‡

Received: 22 June 2015; Revised: 14 October 2015; Accepted: 1 February 2016

Abstract

Reinforcement Learning (RL) is a powerful technique to develop intelligent agents in the field of Artificial Intelligence (AI). This paper proposes a new RL algorithm called the *Temporal-Difference value iteration algorithm with state-value functions* and presents applications of this algorithm to the decision-making problems challenged in the RoboCup Small Size League (SSL) domain. Six scenarios were defined to develop shooting skills for an SSL soccer robot in various situations using the proposed algorithm. Furthermore, an Artificial Neural Network (ANN) model, namely Multi-Layer Perceptron (MLP) was used as a function approximator in each application. The experimental results showed that the proposed RL algorithm had effectively trained the RL agent to acquire good shooting skills. The RL agent showed good performance under specified experimental conditions.

Key words: Reinforcement learning, multi-layer perceptron, neural network, machine learning, RoboCup, robot soccer.

1 Introduction

Operations research (OR) applies a large suite of methods to improve decision-making, including mathematical modelling, heuristics and artificial intelligence (AI). Various AI techniques such as reinforcement learning, neural networks, decision trees and support vector machines are employed in the context of OR [4, 6, 7, 31]. Aiming to develop "intelligent agents" [18], AI is widely used in real-world applications including speech recognition, image processing, machine translation, game playing, automation, medical diagnosis, robotics and many more.

The victory of a computer, Deep Blue, over the human chess world champion in 1997 was probably the most outstanding achievement of AI at the time. It was not only a great

*Department of Industrial Engineering, University of Stellenbosch, South Africa
†Corresponding author: Department of Industrial Engineering, University of Stellenbosch, South Africa, email: jb2@sun.ac.za
‡CSIR-SU Centre for AI Research, Computer Science Division, Stellenbosch University, South Africa

http://dx.doi.org/10.5784/33-1-542

breakthrough but also became a turning point of mainstream AI research. The focus then shifted to more complicated problems, that is, developing intelligent agents working in dynamic, uncertain environments.

RoboCup [23], an annual international robot soccer competition, is one such attempt to promote AI. Aiming at developing a team of fully autonomous soccer-playing robots, teams participating in RoboCup competitions are challenged to incorporate various technologies such as design principles of autonomous agents, computer vision, hardware design and control, real-time reasoning, strategy acquisition, robotics and multi-agent collaboration [32]. There are several leagues in RoboCup, namely the Small Size League (SSL), the Middle Size League (MSL), the Simulation League, the Standard Platform League and the Humanoid League [23]. The focus in this paper is on the SSL.

An AI method, *reinforcement learning* (RL), is widely used to solve real-world problems. Examples of RL applications to real-world problems are filtering personalised web-documents [35], steering an automobile [22], controlling power systems [34], solving job-shop scheduling tasks [9], autonomously flying artistic maneuvers with a helicopter [2, 15], operational space control [17], and implementing real-time strategy games [1].

In robot soccer domains, RL is applied to learn hardware controls such as walking patterns for humanoid robots [5, 16] and ball trapping for four-legged robots [13]. It is also applied to learning individual soccer skills and team-level strategies. Examples of learning individual soccer skills include kicking patterns [20], shooting skills [8], dribbling [21], aggressive defence behaviours [21] and scoring penalty goals [11]. Stone's *KeepAway* task [29] is a good example of an RL application to develop team-level naive attacking strategies. The work inspired other researchers to develop more agressive attacking strategies based on reinforcement learning [12, 14].

In this paper, reinforcement learning was used for an SSL soccer robot. In particular, a new RL algorithm was proposed for infinite Markov Decision Process (MDP) problems with the dynamics of the environment known, and applied to develop shooting skills under various scenarios. The purpose of this paper is to show that the proposed RL algorithm can equip an SSL soccer robot with the mentioned shooting skills, thus allowing the soccer robot to make better decisions during a match. This is, to the best of our knowledge, the first application of RL in the RoboCup SSL domain.

The rest of the paper is organised as follows: In the following section, reinforcement learning methods are discussed and the new RL algorithm is introduced. A brief overview of the RoboCup SSL architecture is provided in the next section, followed by descriptions of the scenarios and the design of the RL experiments. The results of the RL experiments are discussed next, and the last section contains concluding remarks and mentions possibilities for future work.

2 Reinforcement learning background

A reinforcement learning agent tries to learn its behaviour policy to achieve a specific goal through repeated interactions with the environment. At each time-step the RL agent observes the current *state* (an environmental situation in which the agent finds itself),

takes an *action* (from a set of actions allowed in that state) and receives a *reward* (or cost) accordingly. The environment is transitioned to a new state at the next time-step as a result of the action taken by the agent. A sequence of these steps or interactions between the agent and the environment that has a natural end is called an *episode*.

2.1 Reinforcement learning

Reinforcement learning problems are normally formulated as Markov Decison Processes (MDP) [19]. That is, the transition probability from a state s to the next state s' depends only on the current state s and the chosen action a, regardless of the states that the agent passed through to reach the current state s. In a finite-MDP, where the state and action variables are finite, the dynamics of the environment are defined by the following two functions. Let

$\mathcal{P}^a_{ss'}$ be the probability of a possible next state s' given a state s and an action a and
$\mathcal{R}^a_{ss'}$ be the expected value of the reward given any current state and action, s and a, along with any next state, s'.

These functions are given by

$$\begin{aligned} \mathcal{P}^a_{ss'} &= P(s_{t+1} = s' \,|\, s_t = s,\, a_t = a), \text{and} \\ \mathcal{R}^a_{ss'} &= \mathbb{E}(r_{t+1} \,|\, s_t = s,\, a_t = a,\, s_{t+1} = s'), \end{aligned}$$

where s_t and a_t denote the state and action at time-step t, respectively, and r_{t+1} represents the reward given to the agent at time-step t. If an RL problem is formulated as a finite-MDP and $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ are known for all $s \in \mathbb{S}$ ($\mathbb{S}$ denotes the set of possible states), then it is called a *model-based* problem.

The goal of the RL agent is to obtain the optimal policy, *i.e.* to learn how to choose the best action in a state to fulfil the task given to the agent. To do this, RL algorithms employ the notion of *value functions*, which represent the value of each state or the value of each state and action pair. Typically the value of state s is denoted by $V(s)$ and the function V is called the *state-value function*. Similarly, the value of a state-action pair is denoted by $Q(s, a)$ and Q is called the *action-value function*. If R_t represents the sum of rewards the RL agent receives after time-step t and r_{t+1} is the reward given to the RL agent as a result of its action at time-step t, then $V^\pi(s)$, the value of a state s under a policy π, is defined as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\{R_t \,|\, s_t = s\} \\ &= \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\, s_t = s \right\}, \end{aligned} \tag{1}$$

where $\mathbb{E}_\pi\{\cdot\}$ denotes the expected value given that the agent follows policy π. The symbol γ is used in equation (1) for a discount rate to determine the present value of future rewards ($0 \leq \gamma \leq 1$). Thus $V^\pi(s)$ represents the value of a state s based on the expected sum of future rewards, assuming the RL agent starts from that state and follows policy π.

Bertsekas & Tsitsiklis [3] showed that for each MDP there exists an optimal policy π^* that satisfies, for any policy π, $V^{\pi^*}(s) \geq V^\pi(s)$ for all states. All RL algorithms seek

to estimate the true value of $V^{\pi^*}(s)$ or $Q^{\pi^*}(s,a)$ through repeated interactions with the environment. The RL agent starts with random values of $V(s)$ or $Q(s,a)$. As it explores through the state space and receives rewards, it updates the estimated value function according to the experience.

One important feature of value functions is that they satisfy a recursive relationship known as the *Bellman equation*, defined by

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\{R_t \,|\, s_t = s\} \\ &= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\, s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|\, s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,|\, s_t = s\right\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|\, s_{t+1} = s'\right\}\right] \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right]. \end{aligned} \tag{2}$$

Here $\pi(s,a)$ denotes the probability of choosing action a in state s under the policy π. The Bellman equation shows the relationship between the value of a state s and the values of all possible following states of s. This particular property of value functions is used throughout RL algorithms in the estimation of the value function.

Value iteration (Algorithm 1) is an important RL algorithm to estimate $V^{\pi^*}(s)$ for model-based problems. It uses the expected sum of future rewards over all possible next states ($\sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]$) in estimating the value of state s (see Line 6 in Algorithm 1), but with the maximum value, not the expected value as seen in equation (2), over possible actions. The symbol $\pi(s)$ in Line 9 denotes the action to be taken in state s under the policy π.

Algorithm 1: The value iteration algorithm [30].

1 Initialise $V(s)$ arbitrarily, for all $s \in S$;
2 **repeat**
3 $\quad \Delta \leftarrow 0$;
4 $\quad$ **for** *each* $s \in S$ **do**
5 $\quad\quad \mathfrak{v} \leftarrow V(s)$;
6 $\quad\quad V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
7 $\quad\quad \Delta \leftarrow \max(\Delta, |\mathfrak{v} - V(s)|)$;
8 **until** $\Delta < \delta$ *(a small positive number)*;
9 Output a deterministic policy π such that $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;

For *model-free* problems, where the dynamics of the environment are not known, the Temporal-Difference (TD) algorithm (Algorithm 2) is typically used to estimate $V^{\pi}(s)$ for a given policy π. The TD algorithm does not go through all possible states as the value

Algorithm 2: The TD algorithm for estimating V^{π} [30].

```
1 Initialise V(s) arbitrarily, π to the policy to be evaluated;
2 forall episodes do
3     Initialise s;
4     repeat (for each step of episode)
5         a ← given by π for s;
6         Take action a, observe reward r, and the next state s′;
7         V(s) ← V(s) + η[r + γ V(s′) − V(s)];
8         s ← s′;
9     until s is a terminal state;
```

iteration algorithm does, but it goes through episodes, updating the estimation of the value of states which the agent meets in the episode. As the dynamics of the environment are not known in model-free problems, it is impossible for the algorithm to use the expected sum of future rewards over all possible next states as the estimate of the value of the state. Instead it uses the value $r + \gamma V(s')$, which was just learned from the experience, as the target value for the estimation of the state. However, the target value $\mathfrak{v} = r + \gamma V(s')$ is not fully assigned to $V(s)$ because it is not the expected sum of future rewards. The value of the state is updated towards the target $\mathfrak{v}$ from where it is $(V(s))$ by adding a portion of the difference between the target and the current value $(\eta[r + \gamma\, V(s') - V(s)])$ (refer to Line 7 in Algorithm 2). This is called the Temporal-Difference algorithm because it uses the difference between the target value and the current value of the state s, but the difference is only temporarily effective, that is, only in that iteration where r and s' were observed.

Though effective, Algorithm 2 does not provide value functions for an optimal policy π^*. It only provides the estimation of $V^{\pi}(s)$, the value of states for a given policy π. To obtain an optimal policy from this model, a process called "policy iteration" is required. Further information on policy iteration can be found in Sutton & Barto [30, pp.113–125].

Q-learning [33], shown in Algorithm 3, is often used to find an optimal policy for model-free problems. It is essentially a TD value iteration algorithm for model-free problems that uses action-value functions. It goes through episodes and updates the value of the current state-action pair $Q(s, a)$ based on the information it has just experienced, as in the TD algorithm. The adopted TD target $\mathfrak{q} = r + \gamma \max_{a'} Q(s', a')$ is the maximum value over possible actions as in the value iteration algorithm, which makes it possible for the algorithm to search for an optimal policy π^*.

The three algorithms introduced in this section (Algorithm 1, 2 and 3) are popular RL algorithms. Algorithm 1 is for problems modelled as finite-MDPs when the dynamics of the environment are known. Algorithms 2 and 3 are for both finite- and infinite-MDP problems when the dynamics of the environment are not known. There exist problems, though not common, that are modelled as infinite-MDPs when the dynamics of the environment are known. A novel RL algorithm called the *Temporal-Difference value iteration algorithm with state-value functions* is proposed in this paper for such problems. Algorithm 4 shows the pseudo-code of this algorithm.

The value iteration algorithm (Algorithm 1) can be used for this kind of problem by dis-

cretising the continuous state space. However, it does not seem as effective as Algorithm 4 because the value iteration algorithm exhaustively visits all states including states that the agent is not likely to visit. In contrast, in Algorithm 4 the RL agent explores the state space in episodes and updates only the values of the visited states.

Infinite-MDP problems with the dynamics of the environment known can also be dealt with using the Q-learning algorithm (Algorithm 3), which uses action-value functions. Using action-value functions instead of state-value functions requires more resources. In addition, it will take more time to estimate optimal values $Q^{\pi^*}(s,a)$ for all state-action pairs than to estimate $V^{\pi^*}(s)$ for all states. When the dynamics of the environment are not known, Q-learning is a good option. However, when the dynamics of the environment are known, then the optimal policy can be obtained by greedily exploiting $V^{\pi^*}(s)$ according to

$$\pi^*(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi^*}(s')].$$

Therefore Algorithm 4 can be used effectively for problems modelled as infinite-MDPs when the dynamics of the environment are known, which are the cases dealt with in the RL experiments discussed in later sections. In general, the proposed algorithm can be applied to all RL problems in this category (infinite-MDPs with the known dynamics of the environment).

2.2 Function approximation

Thus far, it was assumed that the value functions $V(s)$ or $Q(s,a)$ are represented as a lookup table storing a value for each state s or for each state-action pair (s,a). For RL problems with a large number of states, or with continuous state variables, value functions need to be represented approximately by a set of parameters $\mathbf{z}$, as follows:

$$\begin{aligned} V(s) &= [F(\mathbf{z})](s), \\ Q(s,a) &= [F(\mathbf{z})](s,a). \end{aligned}$$

The problem of estimating $V(s)$ or $Q(s,a)$ is now changed to the problem of searching for the parameter vector $\mathbf{z}$ that represents $V(s)$ or $Q(s,a)$ as accurately as possible. This is called *function approximation.*

Algorithm 3: The Q-learning algorithm [30].

```
Initialise Q(s,a) arbitrarily;
forall episodes do
    Initialise s;
    repeat (for each step of episode)
        Choose a from s using policy derived from Q (e.g. ε-greedy);
        Take action a, observe reward r, and the next state s';
        Q(s,a) ← Q(s,a) + η[r + γ max_a' Q(s',a') − Q(s,a)];
        s ← s';
    until s is a terminal state;
```

Algorithm 4: The TD value iteration algorithm with state-value functions.

State Initialise $V(s)$ arbitrarily;
forall *episodes* **do**
  Initialise s;
  repeat (for each step of episode)
    $\mathfrak{v} \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
    $V(s) \leftarrow V(s) + \eta[\,\mathfrak{v} - V(s)\,]$;
    $a \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$ with ϵ-greedy;
    Take action a and observe the next state s';
    $s \leftarrow s'$;
  until *s is a terminal state*;

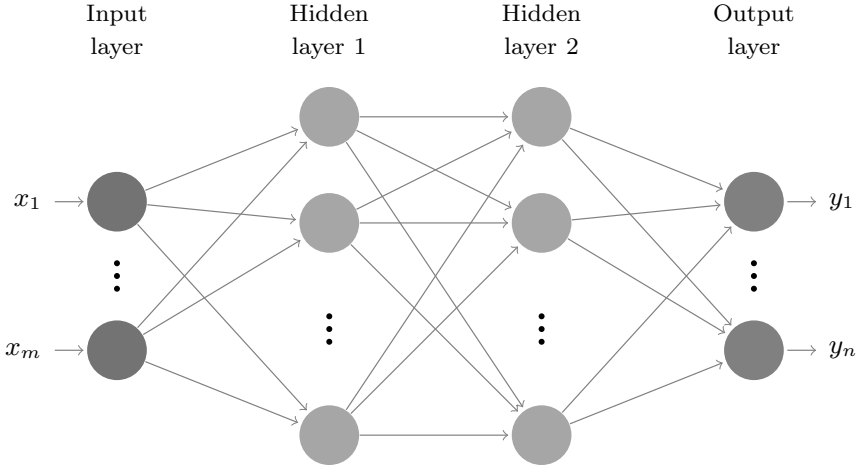


Figure 1: A multi-layer perceptron model. It is assumed that the multi-layer perceptron has two hidden layers with m inputs and n outputs. The number of neurons in the hidden layers are not specified in this figure.

Function approximation is an instance of supervised learning, which is to learn a mapping from inputs to outputs based on a training data set so that the mapping provides the output correctly for any input that is not included in the training data set. The Multi-Layer Perceptron (MLP), a popular supervised learning model, was employed in this research to approximate the value functions.

The MLP is an artificial neural network model that has inputs, outputs and a number of neurons arrayed in different layers between inputs and outputs. Figure 1 shows an example of an MLP. Each connection (denoted by an arrow in Figure 1) between the inputs, outputs and neurons has a weight. An MLP model learns the mapping by adjusting the weights according to the training examples provided. An algorithm called back-propagation (BP), proposed by Rumelhart *et al.* [25], is typically used in updating the weights of the MLP. The algorithm starts with random initial weights and at each iteration the weights are adjusted to minimise the error between the target value of the input, which is provided in the training data set, and the actual response of the network for the given input. More information on the BP algorithm can be found in Haykin [10, pp.161–173].

When an MLP is used as a function approximator for RL problems, the *batch mode* BP algorithm is known to be more efficient than the *sequential mode* [21]. In the sequential mode of the BP algorithm, the weights are updated every time each example in the training set is fed to the network. The batch mode BP algorithm, on the other hand, accumulates the errors and makes the update once after all examples in a training data set are presented.

Algorithm 5 shows the pseudo-code of the TD value iteration algorithm with state-value functions (Algorithm 4) combined with an MLP as a function approximator. Therefore, the value $V(s')$ in Lines 6 and 8 is the output of the MLP when the state s' is given as an input. The TD target $\mathfrak{v}$ and the current input s form a training example. These training examples in the training set $\mathbb{T}$ are used in the batch mode BP algorithm once the episode finishes (see Line 12 in Algorithm 5). Algorithm 5 is used in all the experiments discussed in later sections.

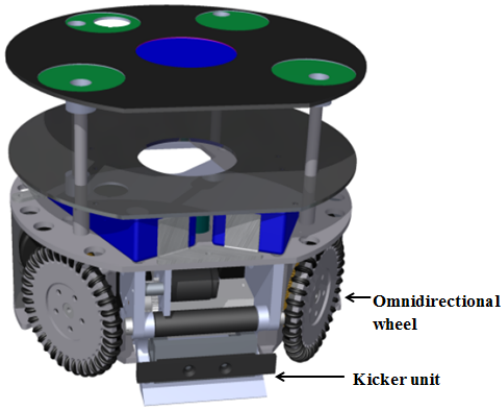
Algorithm 5: The TD value iteration with MLP (batch mode).

1 Initialise $\mathbf{z}$ arbitrarily;
2 **forall** *episodes* **do**
3 Initialise the training data set $\mathbb{T} = \emptyset$;
4 Initialise s;
5 **repeat** (for each step of episode)
6 $\mathfrak{v} \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
7 $\mathbb{T} \leftarrow \mathbb{T} \cup \{(s, \mathfrak{v})\}$;
8 $a \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$ with ϵ-greedy;
9 Take action a and observe the next state s';
10 $s \leftarrow s'$;
11 **until** s *is a terminal state*;
12 Update $\mathbf{z}$ using the batch mode BP algorithm with training data set $\mathbb{T}$;

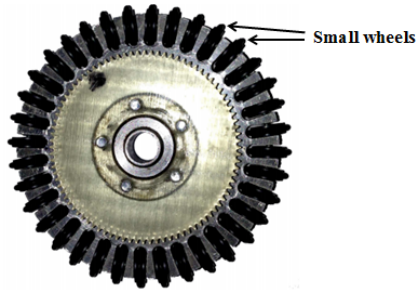
3 The RoboCup SSL

A brief overview of the RoboCup SSL is presented to explain the context of the research. In the RoboCup SSL, teams consisting of maximum six robots play soccer games using an orange golf ball on a pitch of 6 050 × 4 050 mm. The robots' shape and size are confined to a cylinder with a diameter of 180 mm and a height of 150 mm. Figure 2(a) shows an example of the hardware design of a typical SSL robot. The robot has four omnidirectional wheels (Figure 2(b)) and a kicker unit. The omnidirectional wheels have numerous small wheels all around the circumference. These small wheels rotate freely, enabling the robot to move in any direction without having to turn. This feature had a significant effect on the design of the RL experiments discussed in later sections.

The SSL control process is as follows: activities on the field are captured by two cameras mounted above the field and the corresponding information, such as the positions of robots and the ball, is processed by open-source software called SSL-Vision [28] on an off-field computer. Using this information, an independent computerised decision-making module (DMM) produces team strategies for the robot's actions and sends commands to the



(a) CAD design of the SSL robot showing the location of the omnidirectional wheels.



(b) Omnidirectional wheel with many small wheels on its circumference.

Figure 2: Hardware design of an SSL robot [26].

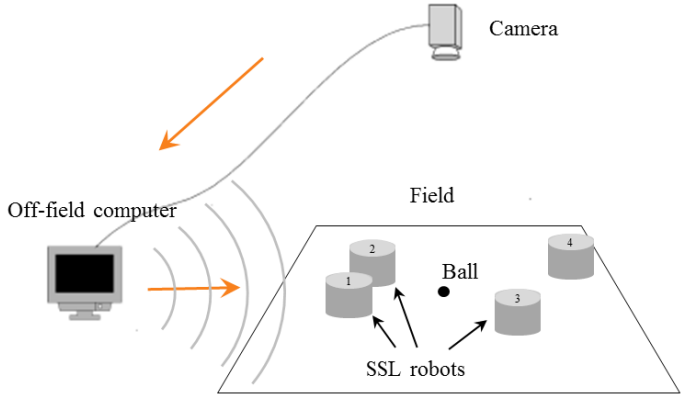


Figure 3: A schematic representation of the RoboCup SSL system [24].

robots in a team via a wireless radio link. Changes on the field caused by movements of the robots and the ball are again captured by the cameras and the processes described above are repeated throughout the game. This control loop iterates approximately 60 times per second. Figure 3 shows the basic elements of the RoboCup SSL system.

The SSL control system described above is centralised. The robots perceive their environment via the cameras and not as individuals, and the DMM makes decisions for all the robots in the team; they are thus under the control of the DMM of their team. Designing and implementing an intelligent DMM is therefore one of the major challenges for teams in the RoboCup SSL.

One of the most important features in the design of the DMM is that it is built in a layered architecture with different levels of abstraction. In a layered architecture, individual soccer skills such as **Shoot**, **Pass**, **Intercept**, *etc.* are developed first, then these skills are used to implement high-level team strategies such as **AgressiveAttack** or **Defence**. Figure 4

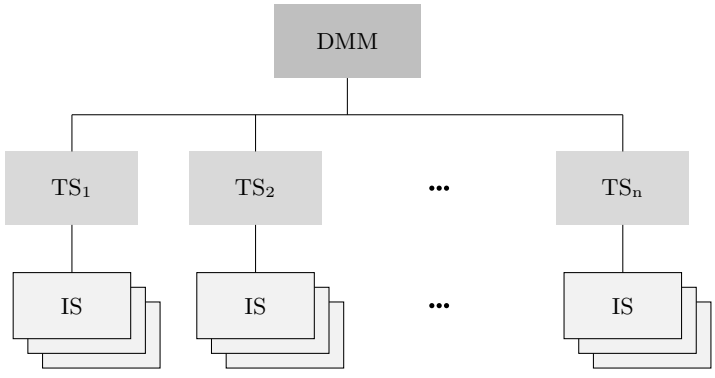


Figure 4: A layered architecture of a DMM. The DMM consists of n team strategies (TS_1, ..., TS_n) and each team strategy has multiple individual soccer skills (IS).

shows an example of a layered architecture of a DMM.

In general, individual skills and high-level team strategies are programmed based on human knowledge and experience, and are then fine-tuned manually. But no matter how fine-tuned they are, these hand-coded skills and strategies are error prone for it is almost impossible to account for all potential game situations by human logic [8]. Also, the hand-coded, human-driven solutions are not guaranteed to be optimal because humans, even experts, are likely to be biased.

Various OR techniques have been applied to improve existing hand-coded skills and strategies in robot soccer. These techniques include linear programming, reinforcement learning, evolutionary algorithms, neural networks and case-based reasoning. Of these, reinforcement learning is particularly widely used in robot soccer domains because its specific settings for problem-solving are well suited to the learning tasks of autonomous robots in a highly dynamic and complex environment such as robot soccer. The next two sections describe one of these attempts: RL applications to the developement of shooting skills for an SSL soccer robot.

4 Experimental design

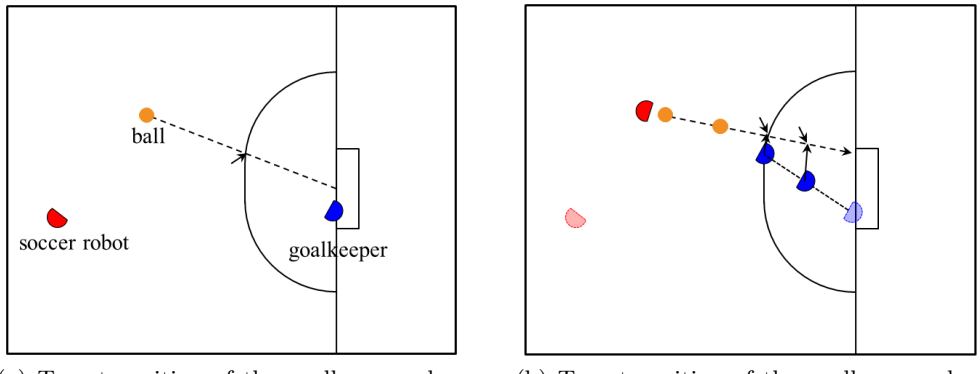
To verify the ability of the proposed algorithm in the robot soccer domain, several experiments were conducted. The purpose of the RL experiments was to develop shooting skills for an individual soccer robot, in various scenarios. The scenarios were defined based on the settings of the experiments such as whether the ball is stationary or moving, or whether there is a goalkeeper or not. When a goalkeeper is assumed, the scenarios were refined according to the behaviour of the goalkeeper. The following tasks were given to the RL agent in each experiment to acquire the basic shooting skills (we refer to the soccer robot as the RL agent):

- Experiment 1: Shooting a stationary ball assuming no goalkeeper.
- Experiment 2: Shooting a moving ball assuming no goalkeeper.
- Experiment 3: Shooting a stationary ball against a confined goalkeeper.

- Experiment 4: Shooting a stationary ball against a smarter goalkeeper.
- Experiment 5: Shooting a moving ball against a confined goalkeeper.
- Experiment 6: Shooting a moving ball against a smarter goalkeeper.

The behaviour of the confined goalkeeper is as follows: The goalkeeper is initially located at a random position on the goal line between the two goal posts. It waits at its initial position until the ball is kicked. Once the goalkeeper detects that the ball is kicked, it tries to block the ball by moving on the goal line. The goalkeeper is not allowed to move forward off the goal line.

A smarter goalkeeper can move off the goal line before the ball is kicked as long as it remains in the defence area. The goalkeeper comes forward as soon as the episode starts, to minimise the open angle for shooting, *i.e.* the angle between the goalkeeper and either side of the goal posts. It moves towards the point on the border of the defence area where the line connecting the ball and the centre of the goal intersects (see Figure 5(a); the target position is marked with an arrow.) If the goalkeeper reaches the target position before the ball is kicked, it waits there until the ball is kicked because the goalkeeper is not allowed to move outside the defence area. When the goalkeeper detects that the ball is kicked, whether it has reached the target position or not, it tries to block the ball by moving from its current position to the closest point on the path of the ball. Figure 5(b) shows the new target position of the goalkeeper in both cases.



(a) Target position of the goalkeeper when the episode starts.

(b) Target position of the goalkeeper when it detects that the ball has been kicked.

Figure 5: Target position of the smarter goalkeeper.

Three state variables are used for the experiments dealing with a stationary ball: 1) the distance D between the ball and the RL agent, 2) the angle α between the orientation of the RL agent and the line connecting the ball and the target, and 3) the angle β between the line connecting the robot and the ball, and the line connecting the ball and the target. These variables are from Duan *et al.* [8]. The geometric relations of the objects in the field and the state variables are illustrated in Figure 6. It is assumed that the target is the centre of the opponents' goal box in this figure. Four additional state variables were used for experiments dealing with a moving ball: v_{bx} and v_{by}, the velocity of the ball in x- and y-direction respectively, and v_x and v_y, the velocity of the RL agent in x- and

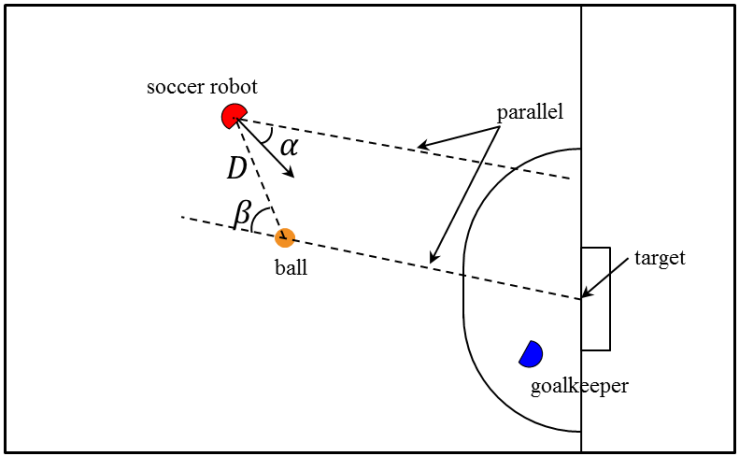


Figure 6: Schematic (not to scale) showing the soccer robot on the playing field and the associated state variables for a given orientation and distance from the ball. A goalkeeper is also present.

y-direction respectively. Note that the state variables are continuous, thus making the problems infinite-MDPs.

Three action variables were considered to form an action space: moving direction of the robot, θ (in rad), its moving speed v (in m/s) and its angular velocity ω (in rad/s). These three action variables are used for all experiments, but the number of actions used is different in each experiment. As the skills in later experiments are more complex and harder to achieve, more refined control of actions is required, which led to the difference in the number of actions in each experiment.

Thirty-six moving directions (evenly distributed between 0 and 2π) were used in Experiments 1 and 2. The number of moving directions was extended to 72 in Experiments 4–6. For the moving speed, the value $v = 0.5$ or $v = 1$ m/s was used for Experiment 1, and $v = 0.5$, $v = 1$ or $v = 2$ m/s for the remaining experiments. Nine different values were used for the angular velocity. They are

$$\left\{-2\pi, -\pi, -\tfrac{1}{2}\pi, -\tfrac{1}{4}\pi, 0, \tfrac{1}{4}\pi, \tfrac{1}{2}\pi, \pi, 2\pi\right\}$$

for the first two experiments and

$$\left\{-\pi, -\tfrac{1}{2}\pi, -\tfrac{1}{4}\pi, -\tfrac{1}{8}\pi, 0, \tfrac{1}{8}\pi, \tfrac{1}{4}\pi, \tfrac{1}{2}\pi, \pi\right\}$$

for the remaining experiments.

By virtue of its omnidirectional wheels, the RL agent can move from one position to another without first having to turn to face the target. Therefore an action is defined by the combination of the three action variables. In addition, nine more actions are possible when the RL agent does not move but only turns at a certain angular velocity, including when $\omega = 0$. This special case is considered as the *kick* action. A constant kicking force of 50 N was assumed. Table 1 summarises the total number of actions used in each experiment.

Experiments	Number of values in Θ	Number of values in $\mathbb{V}$	Number of values in Ω	Total number of actions
1	36	2	9	$36 \times 2 \times 9 + 9 = 657$
2	36	3	9	$36 \times 3 \times 9 + 9 = 981$
3 to 6	72	3	9	$72 \times 3 \times 9 + 9 = 1\,953$

Table 1: Number of values in action sets and the total number of possible actions for each experiment. The action sets Θ, $\mathbb{V}$ and Ω contain possible values of moving direction θ, moving speed v and angular velocity ω, respectively.

A cost-based method was used for the reward system, as was done in the work of Riedmiller *et al.* [21]. In a cost-based method, the RL agent is given a cost c_{t+1}, instead of a reward r_{t+1}, as a result of an action a_t at a state s_t, and the task is to minimise the overall cost in the long run. The cost function $c(s, a, s')$, the cost given to the agent when it chooses an action a at a state s and the following state is s', is defined as

$$c(s,a,s') = \begin{cases} 0.00 & \text{if } s' \in \mathbb{S}^+, \\ 1.00 & \text{if } s' \in \mathbb{S}^-, \\ 0.01 & \text{else,} \end{cases} \tag{3}$$

where $\mathbb{S}^+$ and $\mathbb{S}^-$ denote the set of terminal states with success and failure, respectively. Punishing the agent with a small constant cost 0.01 in all non-terminal states encourages the agent to achieve the goal as soon as possible. The costs for terminal states were determined as such because the output of the MLP is in the range of $[0, 1]$ due to the activation function used in the BP algorithm (refer to Haykin [10, pp.12–15] for more information on the activation function).

All experiments were conducted in a simulation environment with the length of time-step 50 ms. An open-source library called *Open Dynamics Engine* (ODE, Smith [27]), was incorporated in the simulator to calculate the next state (caused by the chosen action) based on the motion dynamics of the robot and the ball. Algorithm 5 was used in all experiments. Each episode was initialised with random positions of the ball, the RL agent and the goalkeeper where applicable.

5 Results and discussion

To show the performance of the RL agent after each learning episode, a learning curve was drawn for each experiment. Figure 7 shows the learning curves of all six experiments. The performance was measured from the results of five tests that were conducted after each learning episode, but the graph was smoothed by using a moving average over the last 100 tests.

As can be seen in Figure 7, the learning was very effective for the first two experiments. After about 30–35 episodes, the RL agent consistently succeeded in scoring a goal with a success rate of approximately 99% and 96% respectively. The small fraction of failure cases was not because of the inefficiency in learning, but due to the initial set-up of the episode. It was observed, for example, that in all failure cases in Experiment 1, the ball

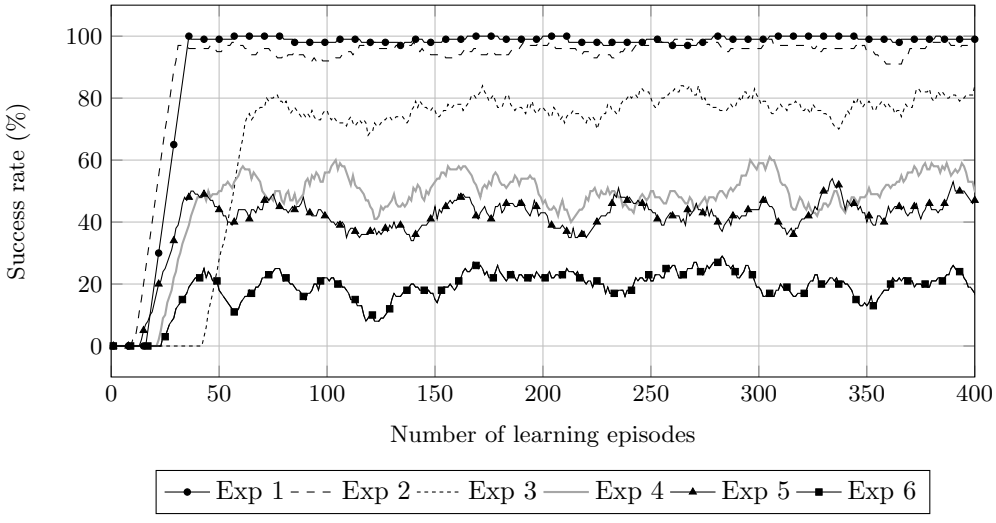


Figure 7: Result: shooting a stationary ball with no goalkeeper present.

was placed too close to the goal line and therefore the shooting angle from the position was too small (see Figure 8). Figure 9 shows an example of the failure cases in Experiment 2 where the ball is placed relatively close to the goal box while the initial position of the RL agent is far from the ball. This gives insufficient time for the RL agent to approach the ball. This case clearly shows that the possibility of scoring with this initial set-up is very low.

In Experiments 3 and 4, the RL agent tries to learn to shoot a stationary ball against a confined and a smarter goalkeeper. Figure 7 shows that the RL agent reached a success rate of approximately 76% and 52% in Experiments 3 and 4, respectively. Most of the failure cases were found again in episodes where the initial positions of the ball and the goalkeeper were determined such that the scoring was impossible in the first place. However, the RL agent failed sometimes in an episode with a seemingly good chance. An intensive investigation of these cases showed that the failure occured when the RL agent could not aim at the target as accurately as necessary due to the discrete action variables. With the action scheme described in the previous section, the minimum angular velocity of the RL agent is $\pm\frac{\pi}{8}$ rad/s. As the time-step in the simulator was set to 50 ms, the RL agent was able to aim at the target position only with an accuracy of $\frac{\pi}{160} = \frac{\pi}{8} \times \frac{1}{20}$ rad. This causes a difference between the ideal orientation and the actual orientation of the learning agent at the final position before shooting. In most cases, the difference was not problematic. In some cases, however, the difference became significant especially when the ball was positioned far from the goal box and thus aiming accuracy was more important.

In Experiments 5 and 6, where the RL agent plays with a moving ball against a confined and a smarter goalkeeper, the success rate reached approximately 40% and 20%, respectively. Besides the usual failure cases due to the initial set-up of the episodes, failures were also caused by the momentum of the moving ball (See Figure 10). The RL agent did not suffer with this problem in Experiment 2, where it also had to deal with a moving ball but without a goalkeeper. In Experiment 5, however, the RL agent had to aim at

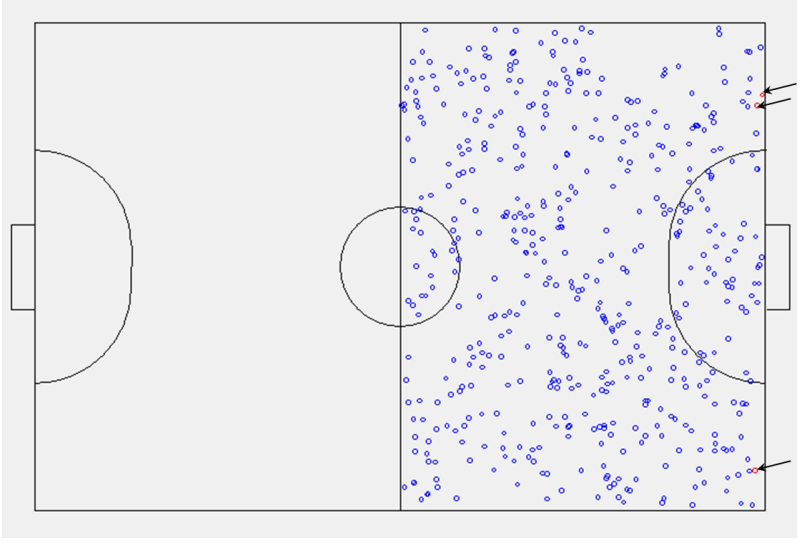


Figure 8: The initial position of the ball in 500 test episodes for Experiment 1. Failure cases are indicated with arrows.

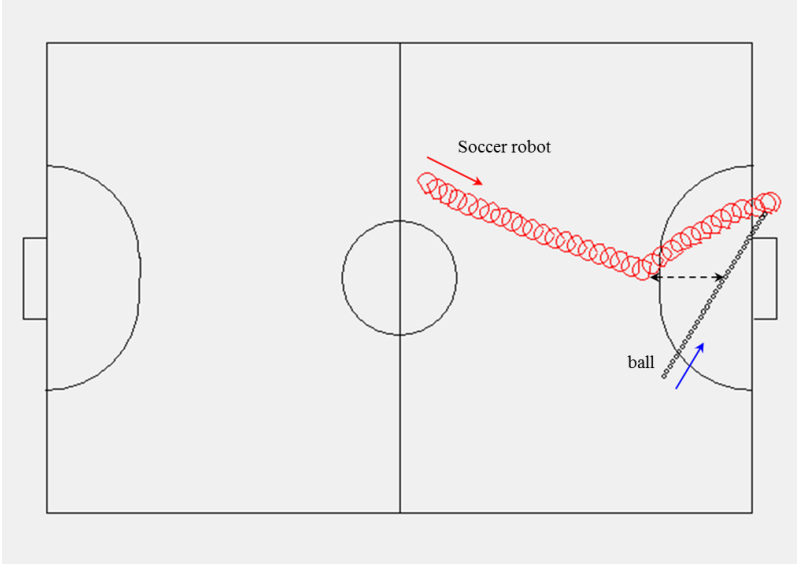


Figure 9: The trajectory of the RL agent and the ball in a failure case in Experiment 2.

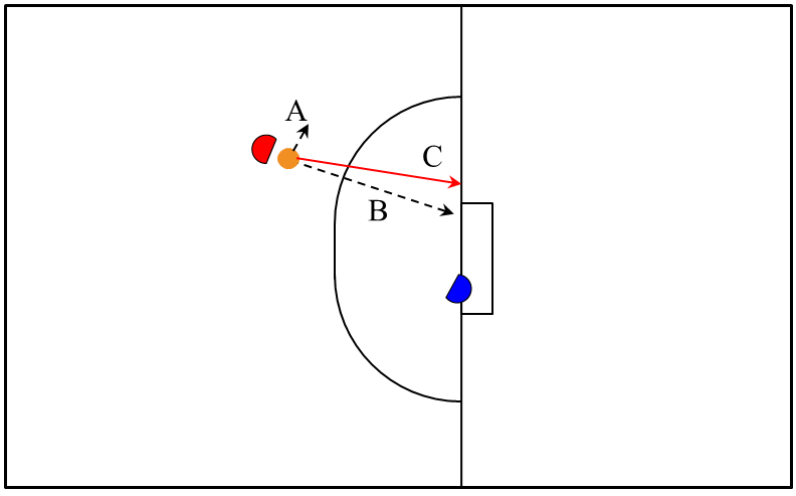


Figure 10: The difference between the target direction and the moving direction of the ball after being kicked. The arrow labelled 'A' shows the original moving direction of the ball, while the arrow labelled 'B' shows the target direction. Arrow 'C' shows the moving direction of the ball after being kicked.

the corner of the goal box (rather than at the center of the goal box as in Experiment 2) to play against a goalkeeper, which posed the problem of the momentum of the moving ball. Interestingly, the same problem was not an issue in Experiment 6, where most of the failures were due to the competence of the goalkeeper. Figure 11 presents an example of the behaviour of the goalkeeper with the trajectory of the moving ball, showing the competence of the goalkeeper.

6 Summary and conclusion

In this research, reinforcement learning (RL) was applied to develop basic soccer skills for soccer robots playing in the RoboCup Small Size League. The *Temporal-Difference value iteration algorithm with state-value functions* was proposed to exploit the knowledge of the dynamics of the environment with an infinite-MDP model. The Multi-Layer Perceptron was employed in the RL algorithm as a function approximator to deal with the continuous state variables.

Six RL experiments were performed to develop shooting skills in various scenarios. Table 2 presents a summary of all the experiments done along with the success rate achieved (rounded up) and the number of learning episodes required to reach the performance.

It was found that the RL agent failed in cases when either the initial set-up of the episode (such as the initial position of the ball and of the RL agent) made it very difficult to achieve the given task, or when the RL agent could not aim at the target as accurately as required due to the discrete nature of the action variables in the experiments dealing with a stationary ball.

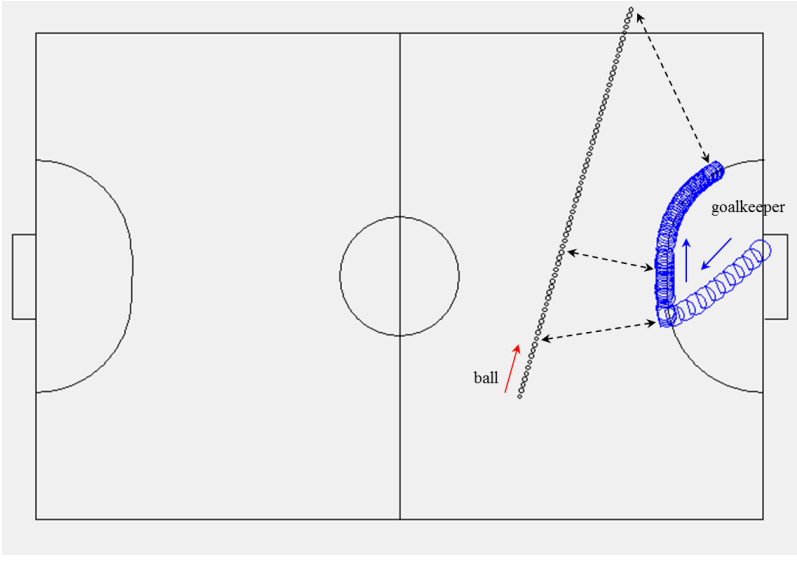


Figure 11: The behaviour of the goalkeeper in Experiment 6. The one-sided arrows indicate the moving direction of the ball and the goalkeeper, respectively. The two-sided arrows with dashed lines show the corresponding positions of the ball and the goalkeeper at the same time-step.

Experiment	1	2	3	4	5	6
Stationary ball	a		a	a		
Moving ball		a			a	a
Goalkeeper (confined)			a		a	
Goalkeeper (smarter)				a		a
Success rate (%)	99	96	76	52	40	20
Number of episodes	35	30	65	40	30	35

Table 2: A summary of the experiments.

In the experiments involving a moving ball, failures occurred due to the difference between the target direction and the actual moving direction of the ball after being kicked. However, in these cases too, the RL agent was able to score a goal in episodes where the difference in question did not have a significant impact. Therefore, it can be concluded that the RL agent was able to score a goal whenever circumstances allowed.

It is also concluded that reinforcement learning using the proposed Temporal-Difference value iteration algorithm with state-value functions effectively trained the RL agent to acquire shooting skills in various situations. The proposed algorithm was applied to robot soccer, but it can be applied to general RL problems with infinite-MDPs and known dynamics of the environment. A comparison study of the performance of Algorithm 4 and other RL algorithms (Algorithm 1 and 3) would be interesting for a future study.

For a better performance of currently developed shooting skills, a new experiment can be

designed for the RL agent to handle a moving ball more precisely, *i.e.* to account for the momentum of the moving ball. This would require a new state variable indicating the angular difference between the moving direction of the ball (Arrow 'A' in Figure 10) and the target direction (Arrow 'B' in Figure 10).

Acknowledgement

The authors thank the South African Council for Scientific and Industrial Research (CSIR) for providing funding for the research.

References

[1] Andersen KT, Zeng Y, Christensen DD & Tran D, 2009, *Experiments with online reinforcement learning in real-time strategy games*, Applied Artificial Intelligence, **23(9)**, pp. 855–871.

[2] Bagnell JA & Schneider JG, 2001, *Autonomous helicopter control using reinforcement learning policy search methods*, Proceedings of the IEEE International Conference on Robotics and Automation, **2(1)**, pp. 1615–1620.

[3] Bertsekas DP & Tsitsiklis JN, 1995, *Neuro-dynamic programming: an overview*, Proceedings of the 34th IEEE Conference in Decision and Control, **1(1)**, pp. 560–564.

[4] Boyacioglu MA, Kara Y & Baykan ÖK, 2009, *Predicting bank financial failures using neural networks, support vector machines and multivariate statistical methods: A comparative analysis in the sample of savings deposit insurance fund (SDIF) transferred banks in turkey*, Expert Systems with Applications, **36(2)**, pp. 3355–3366.

[5] Budden DM, 2012, *Applications of machine learning techniques to humanoid robot platforms*, Doctoral Dissertation, University of Newcastle, Australia.

[6] Dias J, Rocha H, Ferreira B & do Carmo Lopes M, 2014, *A genetic algorithm with neural network fitness function evaluation for IMRT beam angle optimization*, Central European Journal of Operations Research, **22(3)**, pp. 431–455.

[7] Dorigo M & Gambardella L, 2014, *Ant-Q: A reinforcement learning approach to the traveling salesman problem*, Proceedings of the ML-95, Twelfth Intern. Conf. on Machine Learning, pp. 252–260.

[8] Duan Y, Liu Q & Xu X, 2007, *Application of reinforcement learning in robot soccer*, Engineering Applications of Artificial Intelligence, **20(7)**, pp. 936–950.

[9] Gabel T & Riedmiller M, 2008, *Adaptive reactive job-shop scheduling with reinforcement learning agents*, International Journal of Information Technology and Intelligent Computing, **24(4)**.

[10] Haykin S, 2009, *Neural Networks and Learning Machines, 3rd ed.*, Prentice Hall, Upper Saddle River (NJ).

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

[12] Kalyanakrishnan S, Liu Y & Stone P, 2007, *RoboCup 2006: Robot Soccer World Cup X*, Springer, Berlin, Germany, Available from http://dx.doi.org/10.1007/978-3-540-74024-7_7.

[13] Kobayashi H, Osaki T, Williams E, Ishino A & Shinohara A, 2007, *RoboCup 2006: Robot Soccer World Cup X*, Springer, Berlin, Germany, Available from http://dx.doi.org/10.1007/978-3-540-74024-7_8.

[14] Neri JRF, Zatelli MR, Farias dos Santos C & Fabro JA, 2012, *A proposal of Q-learning to control the attack of a 2D robot soccer simulation team*, Proceedings of the Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), pp. 174–178.

[15] Ng AY, Coates A, Diel M, Ganapathi V, Schulte J, Tse B, Berger E & Liang E, 2006, *Autonomous inverted helicopter flight via reinforcement learning*, in *Experimental Robotics IX*. pp. 363–372, Springer, Berlin, Germany.

[16] Ogino M, Katoh Y, Aono M, Asada M & Hosoda K, 2004, *Reinforcement learning of humanoid rhythmic walking parameters based on visual information*, Advanced Robotics, **18(7)**, pp. 677–697.

[17] Peters J & Schaal S, 2008, *Learning to control in operational space*, The International Journal of Robotics Research, **27(2)**, pp. 197–212.

[18] Poole D, Mackworth A & Goebel R, 1998, *Computational Intelligence*, Oxford University Press, Oxford, United Kingdom.

[19] Puterman ML, 2009, *Markov decision processes: discrete stochastic dynamic programming*, **414**, John Wiley & Sons, Hoboken (NJ).

[20] Riedmiller M & Gabel T, 2007, *On experiences in a complex and competitive gaming domain: Reinforcement learning meets RoboCup*, Proceedings of the Computational Intelligence and Games, CIG 2007, IEEE Symposium, pp. 17–23.

[21] Riedmiller M, Gabel T, Hafner R & Lange S, 2009, *Reinforcement learning for robot soccer*, Autonomous Robots, **27(1)**, pp. 55–73.

[22] Riedmiller M, Montemerlo M & Dahlkamp H, 2007, *Learning to drive a real car in 20 minutes*, Proceedings of the Frontiers in the Convergence of Bioscience and Information Technologies, FBIT 2007, pp. 645–650.

[23] RoboCup, *The Robot Worldcup*, [Online], [accessed on 2014-10-21], Available at: http://www.robocup.org/.

[24] RoboCup SSL, *The RoboCup Small Size League*, [Online], [accessed on 2014-10-29], Available at: http://robocupssl.cpe.ku.ac.th/.

[25] Rumelhart DE, Hintont GE & Williams RJ, 1986, *Learning representations by back-propagating errors*, Nature, **323(6088)**, pp. 533–536.

[26] Smit DGH, 2014, *Robocup Small Size League: active ball handling system*, Masters Thesis, Stellenbosch University, Stellenbosch, South Africa.

[27] Smith R, *ODE – Open Dynamics Engine*, [Online], [accessed on 2014-10-29], Available at: http://www.ode.org/.

[28] SSL-Vision, *SSL-Vision: RoboCup Small Size League Shared Vision System*, [Online], [accessed on 2015-03-24], Available at: https://code.google.com/p/ssl-vision/wiki/RequirementsInstallation.

[29] Stone P, Sutton RS & Kuhlmann G, 2005, *Reinforcement learning for RoboCup soccer keepaway*, Adaptive Behavior, **13(3)**, pp. 165–188.

[30] Sutton RS & Barto AG, 1998, *Reinforcement learning: An introduction*, MIT Press, Cambridge (MA).

[31] Üney-Yüksektepe F, 2014, *A novel approach to cutting decision trees*, Central European Journal of Operations Research, **22(3)**, pp. 553–565.

[32] Visser U & Burkhard HD, 2007, *RoboCup: 10 years of achievements and future challenges*, AI Magazine, **28(2)**, p. 115.

[33] Watkins CJCH, 1989, *Learning from Delayed Rewards*, Doctoral Dissertation, University of Cambridge (MA).

[34] Wehenkel L, Glavic M & Ernst D, 2005, *New developments in the application of automatic learning to power system control*, Proceedings of the 15th Power System Computation Conference (PSCC 2005).

[35] Zhang BT & Seo YW, 2001, *Personalized web-document filtering using reinforcement learning*, Applied Artificial Intelligence, **15(7)**, pp. 665–685.