Using Q-learning to Control Robot Goal Keeper Behaviour

Joscha-David Fossel

September 15, 2010

Abstract

In this paper we tackle the problem of robot goal keeping on the Aldebaran Robotics' Nao. Therefore we define input and output available to the Nao. We then present two approaches to create goal keeper behaviour control. The first one is using reinforcement learning (in particular Q-learning), the second one is using hard-coded rules. For the Q-learning based approach, we test two action policies, ϵ-Greedy and Softmax. The experiments reveal that Softmax is more suited for robot soccer goal keeping than ϵ-Greedy. When testing the Q-learning based approach against the hard-coded one we were not able to discover an advantage of Q-learning in terms of performance.

1 Introduction

In August 2007, Aldebaran Robotics' Nao [6] was introduced as the robot used in the Robot Soccer World Cup (Robocup) Standard Platform League, an international robotics competition. In the Standard Platform League, a designated player of every team (the goal keeper) has the duty of actively preventing the opponent from scoring a goal. Furthermore, as in many real life sports, special rules apply to the goal keeper position in case of Robocup: The goal keeper is the only player allowed to stay within the penalty area of its own team and furthermore touch the ball with its arms and hands while beeing in it's penalty area [ref].

This paper presents two approaches, learning and nolearning, to create robot behaviour control for operating the goal keeper. The learning approach discussed in this paper is using Q-learning [11] to evaluate which movement is appropriate depending on the position the Nao is in. The nolearning approach is manually setting the behavioural rules for the robot (hard-coded).

However, in order to employ either of those approaches, certain ways of perceiving (input) and acting (output) in the enviroment the agent is in are requiered. Input and ouput will be discussed in Section 2. Afterwards, Section 3 presents the learning, and Section 4 the nolearning approach. Subsequently, experiments and results are given in Section 5. Finally, this paper closes with drawing conclusions on the presented approaches of controlling goal keeper behaviour in Section 6.

2 Interaction between Nao and Enviroment

In this section the Naos ways to interact with it's enviroment are specified, namely:

- Input: Perception, which enables the Nao to gather information about the enviroment.
- Output: Motions, which enable the Nao to change the enviroment.

2.1 Input

To determine the position of the ball, the image provided by the Naos camera is scanned for a blob, whose RGB components match the ones of the ball. If such a blob is found, the distance and angle between Nao and ball can be estimated depending on which pixels are occupied by the blob.

Using distance and angle, the relative position of the ball in dependency to the goal keeper's position can be derived:

$$Xpos = ballDist * cos(ballAngle) \qquad (1)$$

$$Ypos = ballDist * sin(ballAngle) \tag{2}$$

From taking taking the (X,Y)-position of the ball two times at different points in time ($P_1(X_1, Y_1)$; $P_2(X_2, Y_2)$, see Figure 1) the direction and also the speed of the ball can be derived. Additionally the direction of the ball can be used to calculate the (impact) point of intersection between the ball and the ground line. Therefore the distance of the point of impact to the goal keeper can be calculated, determining if the ball will be in reach to save it.

$$ballSpeed = \sqrt{\triangle X^2 + \triangle Y^2} \tag{3}$$

$$a = \triangle Y / \triangle X \tag{4}$$

$$impactPointY = a * X_2 + Y_2 \tag{5}$$

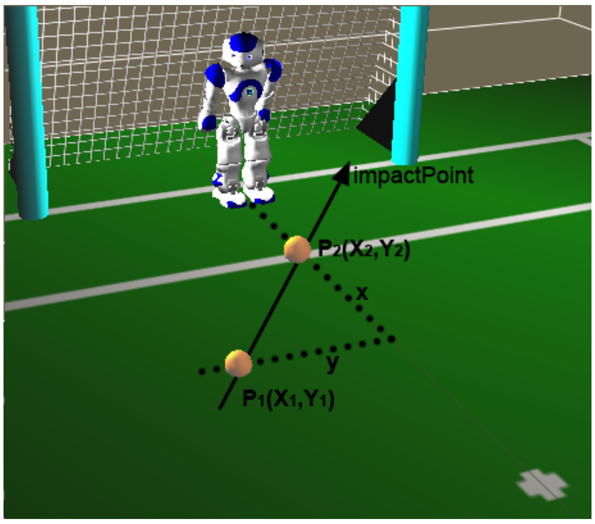


Figure 1: Impact point calculation

2.2 Output

To enable the goal keeper to interact with the enviroment it is provided with the following set of motions:

Strafe: Move sidewards without rotating.

Stand-still: Hold position.

Forward: Walk forwards.

Backward: Walk backwards.

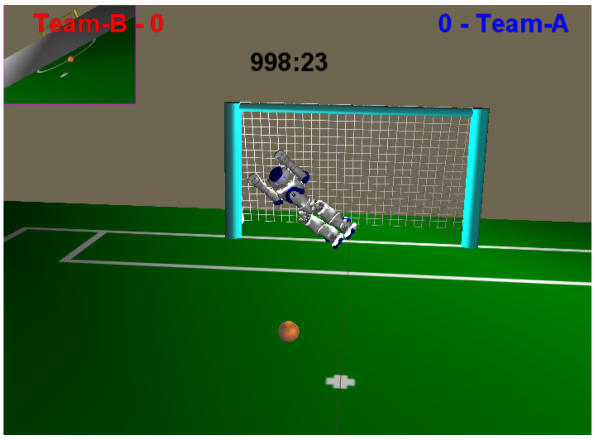


Figure 2: Dive motion

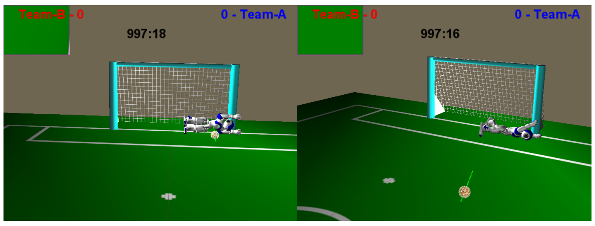


Figure 3: Roll motion clearing the ball

Stand-up: Return to an erect position after diving.

Dive: Jump either to the left or to the right, resulting in the robot being horizontally on the ground. See Figure 2.

Roll: Roll sidewards to remove the ball from the penalty box. Requires the goal keeper to lie on the ground. See Figure 3.

While this set of motions is quite basic, it allows the goal keeper to pursue its task of preventing the opponent from scoring a goal. Adding extra motions, e.g. omni-directional walking [8], or a sophisticated shooting motion would most likely increase the performance, but are so complex that they need to be treated seperately.

3 Machine learning

This section introduces a way of enabling the goal keeper to learn what movements are appropriate depending on the position it is in. Options how to implement this are [7]:

- Supervised learning: Training data including both inputs and desired outputs is required. From that training data the learner learns a function that should allow it to generalize from the training data to unseen examples. An example for supervised learning methods are Artifical Neural Networks [5].
- Unsupervised learning: The learner is only provided with examples from the input space. From that it seeks to determine how the data is organized. Examples are data mining [2] or Kohonen maps [4].
- Reinforcement learning [1]: Seeking to maximize the reward, the learner explores an enviroment and receives rewards or penalties. However it can be unknown to the learner for what action in particular it receives the reward or penalty. An example for this principle is chess, where the player knows the outcome of the game, but not which moves in particular were good or not.

We opt for reinforcement learning, because it does not require correct input/output pairs and has a focus on on-line performance.

To implement reinforcement learning, a Markov Decision Process representing the domain of robot soccer goalkeeping is presented in Subsection 3.1. In Subsection 3.2, Q-learning, a reinforcement learning technique we apply to the Markov Decision Process, is presented. Two action policies (ϵ-Greedy and Softmax) are given in Subsection 3.3.

3.1 Markov Decision Process

The task of robot soccer goalkeeping can be translated into a Markov decision process, i.e. it satisfies the Markov property [9]: The effects of an action taken in a state depend only on that state and not on the prior history.

The basic reinforcement learning model applied to Markov Decision processes [1] consists of the following items:

- A set of possible world states S.

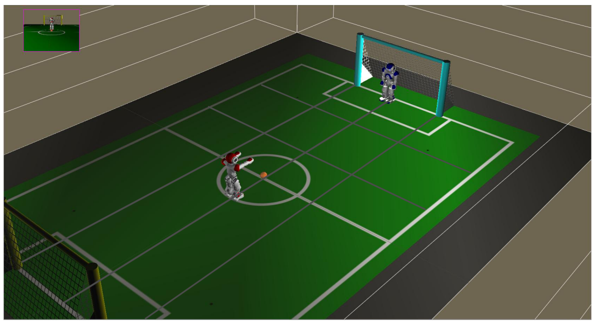


Figure 4: Playing field distributed into buckets

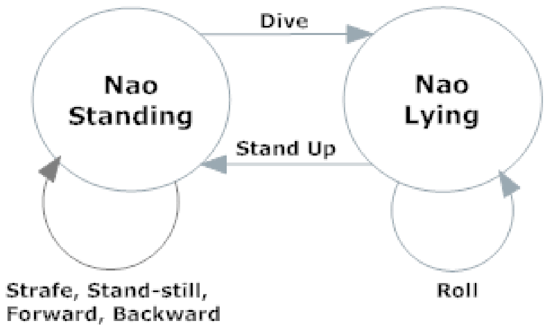


Figure 5: Robot soccer MDP

- A set of possible actions A.
- A real valued reward function $R(s)$.

If S, A, $R(s)$ are defined as follows, the Markov property is fulfilled in robot soccer:

State $s \in S$ of the world is defined by the position the agent is in (e.g. lying on the floor covering the left side of the goal), and the position, speed and impact point of the ball (see Subsection 2.1). These continous valued properties are distributed into buckets (see Figure 4) in order to gain a finite number of discrete states.

The sets of possible actions A the agent can perform are defined as the available motions discussed in Subsection 2.2. See also Figure 5.

The reward function $R(s)$ is defined depending on the outcome of an attack. The rewards are set as following:

- No goal, Nao lying on the ground: Reward 75.
- No goal, Nao in erect position: Reward 100.

- Goal: Penalty -10.

Those reward/penalty values contain the information that preventing a goal by diving is less favorable (due to the limited mobility afterwards) than preventing a goal without diving. Obviously not preventing a goal at all is the least favorable course of action, and therefore results in a penalty.

3.2 Q-learning

In 1989 Watkins[11] introduced a reinforcement learning technique called Q-learning. It allows an agent to optimize its performance through a system of rewards and punishments without being supervised. By trial and error search the agent explores which actions are suitable depending on the position it is in. Q-learning enables the agent to optimize not only immediate, but also delayed rewards. To do so, Q-learning learns an action-value function that approximates the utility of taking an action in a certain position. This action-value function is learned by exploring the state-space following a certain action-policy (defined in Subsection 3.3).

We apply Q-learning to the Markov decision process modeling robot soccer defined in Subsection 3.1.

To implement Q-learning we introduce the following two matrices R and Q:

- The environment reward matrix R stores state dependent rewards/penalties. If for example the ball enters the goal a penalty of -10 is applied. See $R(s)$ at Subsection 3.1.
- Matrix Q, initialized as a zero matrix, represents what the agent has learned so far. It consist of the state the agent is in on the one axis, and the actions it can perform on the other.

In the beginning the agent assumes that all actions are equally good (Q matrix initialized as zero matrix - the agent has no experience about what actions will yield reward). Therefore the agent selects random actions and updates the Q matrix, depending on the reward (or punishment) that result from its actions (see Formula 6). This procedure is repeated until the agent encounters a state where the Q matrix is not zero. In that case it might chose not to perform a random action, but act according to its action policy (see Subsection 3.3). Continuing this, the agent learns which actions will lead to a maximization of rewards, so that it can act appropriate in every position.

Algorithm 1 Q-learning Algorithm

```
Set parameter gamma, alpha, and R
Initialize matrix Q as zero matrix
For each episode:
 Determine initial state
 Do while not reach goal state
  Select a according to action policy
  Take action a, observe r, s'
  Compute the new Q(s,a) with formula (6)]
  s <- s'
 End Do
End For
```

The associated algorithm is shown at Algorithm 1, where y is the so called discount factor with $0 \leq y < 1$. y near 0 will result in the agent considering only immediate rewards and y towards 1 in considering future rewards with greater weight. Parameter α is the learning rate that determines the importance of newly acquiered information. A value of 0 for α prevents the agent from learning anything, while a value of 1 makes the agent consider only the newest information. The Transition Rule [9] that approximates the Q-Values is shown at Formula 6.

$$\begin{aligned} E(s,a) &= y\, max_{a'}[Q(s',a')] - Q(s,a) \\ Q(s,a) &= Q(s,a) + \alpha\,[R(s,a) + E(s,a)] \end{aligned} \tag{6}$$

3.3 Action Policies

The process of selecting one action among all possible actions needs to be elucidated. So called action policies define what action is to be chosen in any position encountered. These action policies define when to explore and when to exploit, i.e. when not to select an action that is so far known to give the highest reward, and instead explore if there might be a strategy that gives even more reward.

Two common action policies are:

- ϵ-Greedy[9]: Most of the times the action with the highest so far known reward will be selected (greedy), but with a certain probability ϵ, a uniformly distributed random action is chosen instead in order to provide the possibility of exploration. Therefore if there are infinite runs the optimal policy will be discovered since all possible actions are explored.

- Softmax[9]: The drawback of ϵ-Greedy is that it is explored evenly among all actions, therefore the worst action has the same likelihood of being selected as the second best. If dealing with a problem where the worst possible action is very bad this might be unwise. When exploring, the Softmax policy differs from ϵ-Greedy to that extent that it considers the actions estimated values and weights them accordingly, linking probability of an action being chosen for exploration to their value estimates:

$$W(s,a) = \frac{e^{Q(s,a)/r}}{\sum_{b=1}^{n} e^{Q(s,b)/r}} \tag{7}$$

  The positive parameter r is the temperature that determines the sensitivity of the relation between weights and estimated values. With a high temperature larger differences in the estimated values also cause high weight differences, while a temperature of zero results in same weights for every action independent of their value estimates (ϵ-Greedy).

4 Hard-coded Behaviour

As an alternative to the in the previous section introduced Q-learning based approach, this section introduces a rather simple way of controlling the goal keeper. Viz. manually defining how to behave in certain situations, deduced from what seems reasonable from a human point of view.

This approach is implemented by defining rules that cover all relevant situations.

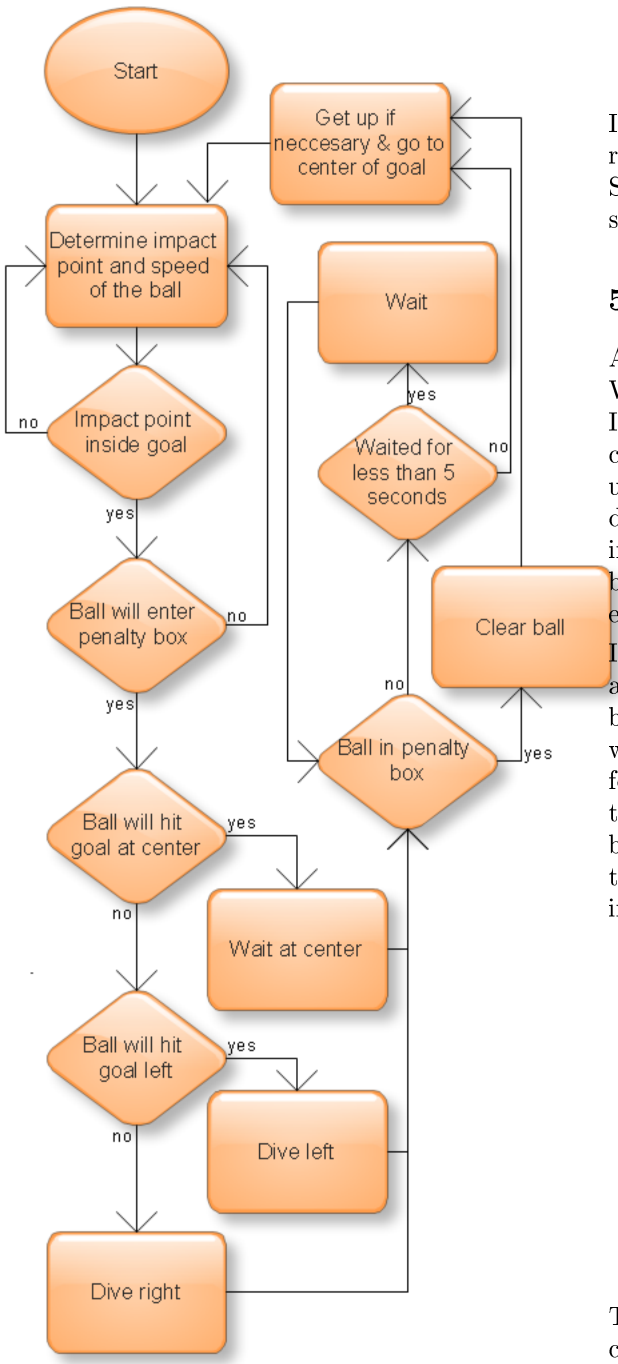
1. In the beginning it is determined if the goal keeper needs to act. Therefore the estimated impact point is taken into account. If that impact point is outside of the goal plus a certain safety threshold (to absorb inaccuracy from perception) the goal keeper obviously does not have to engage, and *1* is repeated. Else it will execute *2*.
2. If, on the other hand, the impact point is inside the goal area, the distance and speed of the ball are used to decide whether the ball will enter the penalty box, or not. In order to determine if the ball is fast enough to hit the goal instead of stopping in front of it, a threshold can be defined - i.e. if the ball exceeds a certain velocity v at a certain distance d the goal keeper is required to act, if not the ball will stop before reaching the goal. Identifying v and d can be accomplished by trial and error. An alternative is using the second derivative to determine if the ball is fast enough - however we were unable to obtain good results using that approach. If the ball is predicted to enter the penalty box, the goal keeper either dives to the left or right according to the impact point prediction, or holds its position in case the ball is estimated to hit the centre of the goal. Afterwards *3* will be executed. If the ball is predicted to not enter the penalty box, 1 is repeated.
3. The goal keeper will hold its current position (i.e. either lying on the ground or standing in the centre of the goal) until the ball has entered the penalty box. If that happens within 5 seconds 4 is executed. If 5 seconds pass and the ball does not enter the penalty box 5 is executed.
4. The goal keeper attempts to clear the ball by either walking towards the ball and kicking it (if in erect position), or by rolling towards the ball, both resulting in the ball being removed from the penalty area. Then it proceeds to *5*.
5. If necessary the goal keeper will start the stand up routine, move to the initial position and go to *1* again.

Figure 6 shows the flowchart of this procedure.

5 Experiments

In this section we try to answer the following two questions:

1. Which of the previously introduced action policies, namely ϵ-Greedy and Softmax (see



Start
Get up if neccesary & go to center of goal
Determine impact point and speed of the ball
Wait
Impact point inside goal
no
yes
Waited for less than 5 seconds
yes
no
Ball will enter penalty box
no
yes
Clear ball
Ball in penalty box
no
yes
Ball will hit goal at center
yes
no
Wait at center
Ball will hit goal left
yes
no
Dive left
Dive right

Figure 6: Flowchart of hard-coded behaviour

Subsection 3.3), is better suited to solve the problem of robot goalkeeping?

2. How does the Q-learning based approach perform compared to the hard-coded approach?

In order to answer these question we conduct a series of experiments described in Subsection 5.1. In Subsection 5.2 the results of these experiments are shown and discussed.

5.1 Set-up

All experiments are conducted on the Cyberbotics Webots simulator *[12]* platform for several reasons: It prevents hardware damage, it allows to by-pass camera noise by using both the real position values provided by the simulator and also the noisy data provided by the simulated camera, and most importantly because it enables to do a high number of unsupervised test runs that would require extensive time if done in reality.

In order to compare ϵ-Greedy and Softmax, the agent is initialized with an empty Q-Matrix for both action policies. Then 100 random shots towards, but not necessarily on the goal are performed (Each attack is one episode). For both action policies the cumulated Q-Values and the number of errors (goals) are observed. The agent is trained with the following parameters that were set intuitively:

- $\gamma = 0.5$, the agent consideres both immediate and future rewards.
- $\alpha = 0.5$, the agent weights old and newly acquiered information evenly.
- $\epsilon = 0.1$, the agent eplores instread of exploiting with a probability of 10%.
- $r = 0.5$, under Softmax, actions with a high estimated value are more likely to be chosen.

To compare the Q-learning based with the hard-coded approach, the agent is initially trained using the Softmax policy (same parameters as in the previous experiment, 1000 episodes). Afterwards 400 evaluation episodes are conducted. The hard-coded approach is also tested by carrying out 400 evaluation episodes, and afterwards the number of

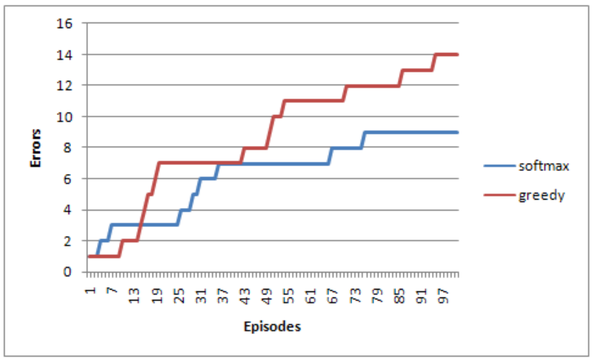


Figure 7: Number of committed errors (goals),ϵ-Greedy vs. Softmax

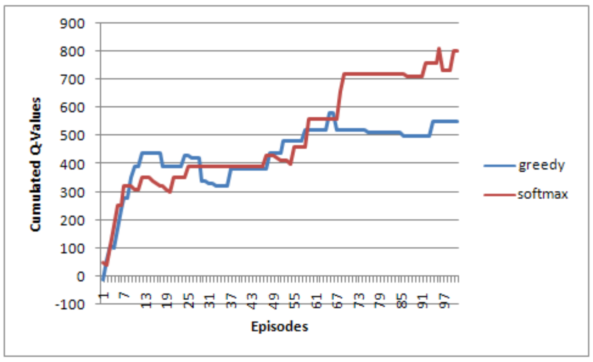


Figure 8: Cumulated Q-values, ϵ-Greedy vs. Softmax

errors are compared. These two experiments are conducted using the NAO camera and using the real positions provided by the Webots simulator, both for training the agent and when running the tests.

In order to ensure comparableness Common Random Numbers[3] are used so that the same random shots are performed.

5.2 Results & Discussion

5.2.1 ϵ-Greedy vs. Softmax

Figure 7 shows the number of errors the agent committed on the one axis and the number of episodes on the other. When following the ϵ-Greedy policy the agent commits more errors than when using Softmax.

This is a result of the possibility of making 'fatal error' in the domain of robot soccer. If, for example, the ball is aimed at the left corner of the goal and still at a sufficient distance, both waiting for the ball to come closer and then acting as well as diving to the left corner is suitable. However, if the agent dives right there is nothing it can do anymore to gain a reward in that episode. As it is more likely to commit such a 'fatal erros' using theϵ-Greedy action policy, Softmax performs better.

Figure 8 shows the cumulated Q-values for both ϵ-Greedy and Softmax. The graph shows that under the Softmax action policy higher cumulated Q-Values are achieved. Therefore we may conclude that when using Softmax, ways to prevent a goal are discovered more quickly.

5.2.2 Q-learning based vs. Hard-coded Approach

As can be seen in Figure 9, the Q-learning based approach does not perform better than the hard-coded approach. When trained with and using the real values as input (Q-real) both perform equally good.

There are several reasons for that. For once, while Q-learning has been proven to converge[10] to the optimal solution, the discretisation of the originally continuous values that define the state the agent is in leads to loss of precision. In order to compensate a large number of states may be introduced, though that would increase the required training time in return because of the additional number of states. In this implementation we used a rather low number of buckets (12 buckets for the playing field, Figure 4). This might be a reason for the bad performance of the Q-learning based approach, as it may prevent it from handling camera noise sufficiently. The experiment shows that the Q-learning based approach is more affceted by camera noise than the hard-coded approach. In contrast the hard-coded behaviour was manually adjusted to reduce the amount of flawed actions resulting from imprecise sensor readings. It is for example programmed to dive even when the shot is (only due to imprecise sensor readings) calculated to slightly miss the goal, while Q-learning has no possibility to determine if a shot is predicted to slightly or clearly miss

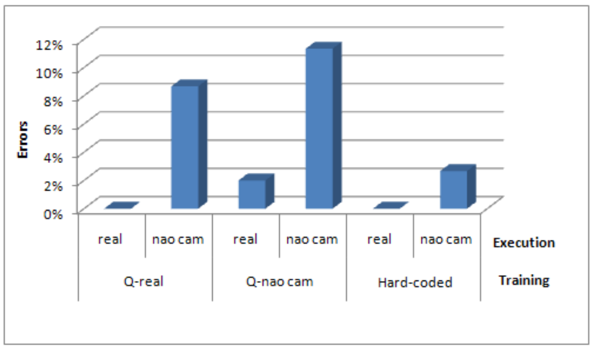


Figure 9: Q-learning based vs. Hard-coded Approach

the goal (due to the small number of buckets).

6 Conclusions & Future Research

In this paper we proposed a Q-learning based and a hard-coded approach to solve the problem of robot soccer goal keeping. We furthermore tested two action policies for the Q-learning based approach. The experiments show that softmax is an adequate action policy in this domain. The experiments furthermore show that our Q-learning based approach does not outperform out hard-coded rule based approach. However, the reason for that might be the implementation that uses only a small number of buckets the continous states are divided in. Also, a learning based approach has the advantage that it is able to adapt to changes, while the hard-coded approach does not. If for example the basic image recognition is improved the hard-coded approach would need to be changed. In contrast, we expect the Q-learning based approach to be able to cope with new image recognition easily.

For future reseach we suggest improving the image recognition, and also integrating more information (about for example friendly and/or enemy agents) into the input. By that the complexity of the problem increases, and hard-coded behaviour might become unfeasible, necessitating another form of controlling the agent, like Q-learning. Likewise the number of buckets should be increased.

We presented a basic framework with decent performance that offers possibilities for future improvements. Therefore we may conclude that Q-learning is a suitable approach to robot soccer goal keeping.

References

[1] Kaelbling, Leslie Pack, Littman, Michael, and Moore, Andrew (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237–285.

[2] Kantardzic, Mehmed (2002). *Data Mining: Concepts, Models, Methods and Algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

[3] Kleijnen, Jack P. C. (1975). Antithetic variates, common random numbers and optimal computer time allocation in simulation. *MANAGEMENT SCIENCE*, Vol. 21, No. 10, pp. 1176–1185.

[4] Kohonen, T., Schroeder, M. R., and Huang, T. S. (eds.) (2001). *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[5] Lawrence, Jeannette (1993). *Introduction to neural networks*. California Scientific Software, Nevada City, CA, USA.

[6] Robotics, AldebaranNao academics datasheet.

[7] Sewell, Martin (2007). 1 introduction machine learning.

[8] Strom, Johannes, Slavov, George, and Chown, EricOmnidirectional walking using zmp and preview control for the nao humanoid robot.

[9] Sutton, Richard S. and Barto, Andrew G. (1998). *Reinforcement Learning, An Introduction*. MIT Press.

[10] Watkins, Christopher J.C.H. and Dayan, Peter (1992). Technical note: Q-learning, machine learning.

[11] Watkins, Christopher J.C.H. (1989). *Learning from delayed rewards*. Ph.D. thesis, University of Cambridge, Psychology Department.

[12] www.cyberbotics.com (2010). Webots reference manual.