Learning robotic soccer controllers with the Q-Batch update-rule

João Cunha*†, Rui Serra*, Nuno Lau*†, Luís Seabra Lopes*† and António J. R. Neves*†
*Department of Electronics, Telecomunications, and Informatics
University of Aveiro
†Institute of Electronics and Telematics Engineering of Aveiro
University of Aveiro

***Abstract*—Robotic soccer provides a rich environment for the development of Reinforcement Learning controllers. The competitive environment imposes strong requirements on performance of the developed controllers. RL offers a valuable alternative for the development of efficient controllers while avoiding the hassle of parameter tuning a hand coded policy. This paper presents the application of a recently proposed Batch RL update-rule to learn robotic soccer controllers in the context of the RoboCup Middle Size League. The Q-Batch update-rule exploits the episodic structure of the data collection phase of Batch RL to efficiently evaluate and improve the learned policy. Three different learning tasks, with increasing difficulty, were developed and applied on a simulated environment and later on the physical robot. The performance of the learned controllers is mostly compared to hand-tuned controllers while some comparisons with other RL methods were performed. Results show that the proposed approach is able to learn the tasks in a reduced amount of time, even outperforming existing hand-coded solutions.**

I. Introduction

The development of efficient robotic controllers is a demanding task. Classical approaches involve model-based solutions which require a prior expert knowledge of the system. Additionally, even in the presence of a reliable model, sensor and actuactor noise create additional difficulties when fine tuning a controller. In this context, Reinforcement Learning approaches emerge as a valuable alternative. Guided by rewards and penalties the agent is able to learn directly from interacting with the environment. This greatly reduces the required input from the developer.

Robotic competitions such as the RoboCup robotic soccer leagues present a valuable opportunity to apply RL approaches in a context of an extremely competitive and adversarial environment. When applying RL methods in robotics, data efficiency gains extreme importance. As opposed to simulation scenarios, gathering the interaction data in real world carries an associated cost. There are often situations during learning that can result in damage for the environment. Additionally, if the robot requires too many interactions to learn a task, the repeated execution of certain actions can result in a deterioration of the robot physical platform. Generalization is then a very important aspect of any RL method.

Batch Reinforcement Learning is a class of RL methods that can be combined with function approximators to achieve fast and data-efficient learning solutions. Batch Reinforcement Learning has recently been applied with great success to learning in physical robotic platforms. This paper presents some recent developments in this field, namely the application of Q-Batch, a newly developed update-rule in a number of learning tasks involving physical robots. The learning tasks were developed in the context of the CAMBADA project, an on-going research at the University of Aveiro, which developed a team of cooperative mobile robots that compete in the RoboCup Middle Size league.

The remainder of this paper is structured as follows: Section II presents the main concepts of Batch Reinforcement Learning. Section III discusses the proposed approach. The learning tasks developed are discussed Section IV and Section V concludes the paper.

II. Reinforcement Learning

Reinforcement Learning [1] is a sub-area of Machine Learning drawing inspiration on biology and animal behavior learning. Although not a new field of study, it has received a lot of attention in recent years from researchers worldwide having sprawled in a multitude of different methods [2], [3].

A class of methods that has showed promising results when applied to robotics is Batch Reinforcement Learning. Batch Reinforcement Learning [4] differs from other RL methods in that it estimates a policy π from a set $\mathcal{F}$ composed of all the N transitions sampled from the environment. A key characteristic of Batch RL methods is that the value function is updated synchronously in a global manner by processing the entire set $\mathcal{F}$ in a single step. While purely online methods, such as Q-Learing, update a Q-function as soon as a transition is observed, Batch RL methods update a Q-function once for all state-action pairs in $\mathcal{F}$. To achieve a synchronous update, a so-called Batch RL operator is applied, in a kernel fashion, to all collected transitions, generating a pattern $\mathcal{P}$ mapping state-action pairs to the target Q-function value, $\mathcal{P} = \{(s_i, a_i), \bar{Q}(s_i, a_i) | i = 1, \ldots, N\}$. This presents an opportunity to apply batch supervised learning methods to build an approximate Q-function $\hat{Q}$, through regression. The approximate Q-function $\hat{Q}$ is then able to generalize the obtained return to states not visited in $\mathcal{F}$. Since the update is synchronous, the function approximator can also be optimized in a global manner using more sophisticated approaches such as RProp [5] to obtain a robust approximation.

Among other methods, we highlight the Neural Fitted Q Iteration (NFQ) [6], an instance of the class of FQI methods,

which relies on multilayer perceptrons, a powerful function approximator, to represent the approximate Q-function $\hat{Q}$. NFQ builds the pattern $\mathcal{P}$ by applying a Dynamic Programming adapted version of the Q-Learning update-rule, over all the collected transitions:

$$\bar{Q}(s_i, a_i) = r_i + \gamma \max_b \hat{Q}(s'_i, b), \forall i \in 1..N \tag{1}$$

where s_i, a_i and s'_i are the current state, action chosen and following state of the i^{th} transition, respectively.

III. Q-Batch

A common approach on learning tasks using Batch RL methods involves building the batch incrementally, usually at the termination of an episode, appending the recently collected interactions to the existing batch. The batch is then processed in order to extract an improved policy. The process can then be repeated alternating data collection and policy improvements phases until the policy obtained can fulfill the learning task at hand. Assuming this workflow, then within the same episode the transitions are sampled along connected trajectories. Each episode i is then a time consistent sequence of T_i states, actions and rewards. This representation allows for a more compact data set $\mathcal{F}$ since the following state of a given transition and the current state of the following transition are now redundant information, thus we can avoid storing following states s' explicitly. Considering the timestep j of episode i, the corresponding state, action and reward are now represented by $s_{i,j}$, $a_{i,j}$ and $r_{i,j}$, respectively.

The Q-Batch [7] update-rule aims to exploit this structure. Q-Batch attemps to perform a rollout over the trajectory of the episode. Each rollout is evaluated according to the concept of n-step returns. This concept builds on the basis that the return can be calculated from an intermediate number of n steps of real rewards and the estimated value of the state visited after n transitions. Therefore a one-step return is based on the first reward and the value of the state one step later, a two-step return is based on the two first rewards and the value of the state two steps later, and so on as shown in (2):

$$\begin{aligned} R_t^1 =& r_{t+1} + \gamma V(s_{t+1}) \\ =& r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) \\ R_t^2 =& r_{t+1} + \gamma r_{t+2} + \gamma^2 \max_{a \in A} Q(s_{t+2}, a) \\ & \vdots \\ R_t^n =& \sum_{i=0}^{n-1} \gamma^i r_{t+1+i} + \gamma^n \max_{a \in A} Q(s_{t+n}, a) \end{aligned} \tag{2}$$

Instead of trying to find an optimal value for n, which is data dependent, Q-Batch seeks the rollout that yields the maximum return, in other words, $\bar{Q}$ is the maximum n-step return found. To reduce the computational complexity the batch is processed in reverse order, taking advantage of *memoization* and the recursive relation between maximum n-step returns, $\max_k R_t^k = \max(R_t^1, r_{t+1} + \gamma \max_{k'} R_{t+1}^{k'})$. The final Q-Batch update-rule is presented in (3).

$$\begin{aligned} \bar{Q}(s_{i,j}, a_{i,j}) &= \max_k R_t^k \\ &= \max(R_t^1, r_{t+1} + \gamma \max_{k'} R_{t+1}^{k'}) \\ &= \max(R_t^1, r_{t+1} + \gamma \bar{Q}(s_{i,j+1}, a_{i,j+1})) \\ &= r_{t+1} + \gamma \max(\max_b \hat{Q}(s_{i,j+1}, b), \bar{Q}(s_{i,j+1}, a_{i,j+1})) \end{aligned} \tag{3}$$

IV. Learning Tasks

This section presents the developed learning tasks and discusses the obtained controller performance after the learning procedure. All the tasks were applied in the Middle Size League robots of the CAMBADA team. In this league, two teams composed of 6 robots play on a 18 m $\times$ 12 m field. The environment is structured, with the elements of the game constituted mainly by uniform colours. The field is green with white field lines, the ball has a dominant colour (usually orange or yellow) and the robots are mainly black. Each robot must not exceed a squared base of 52 cm with a maximum height of 80 cm. The robots are completely autonomous being the human-interaction restricted to the human refeere commands which are relayed to the robots through a control station via a WiFi network. All sensors of the robots are on-board along with the computing components. The robots can also share information with each other and a control station that acts as a coach.

Each CAMBADA robot, presented in Figure 1, uses a camera as the main sensor mounted in a catadioptric system, ensuring a 360 degrees field of view around the robot. This allows the robot to perceive most of the environment without requiring the robot to face any particular direction. The robot moves by means of an omnidirectional drive composed of three swedish wheels. Additionally the robot possesses an active mechanism for ball retention.

Fig. 1. A CAMBADA Middle Size League robot.

A key aspect in applying Reinforcement Learning in real robotics, highlighted by Lauer et. al, is to ensure the Markov property. Similarly to other approaches [8], in order to cope with non-neglegible system delays, we predict the state of the world after the effects of the actuators, to a point where the current decision being made will take effect [9].

In the design of the learning tasks, an available simulation environment was used since it allowed for an easier gathering of data and tuning of the learning parameters. After learning in simulation, the procedure was repeated in the physical robots. To learn the different tasks the Neural Fitted Q Iteration method was applied, using a multilayer perceptron (MLP) to

approximate the Q-Function. MLPs output a smooth approximation of training target data. With this in mind, throughout the learning tasks we will apply a smooth reward function, proposed in [10], that can generate state-action value functions easier to approximate by type of function approximator. Having defined a target point s^{target}, a region of width δ is defined around the target where the cost decreases smoothly, from 95% of a base cost down to 0. The reward function is defined by (4) and represented in Figure 2. Instead of rewards, we found it more intuitive to define the learning tasks in terms of costs. Therefore learned policies will choose the actions that minize the sum of costs.

$$r(s,a,s') = C \times \tanh^2\left(|s^{target} - s| \times \frac{\tanh^{-1}(\sqrt{0.95})}{\delta}\right) \quad (4)$$

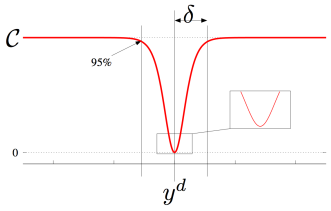

Fig. 2. Smooth reward function used in the learning tasks developed, adapted from [10].

A. Rotating to a point

The first task defined involves having the robot rotate over itself to face a given orientation. While far from a great challenge this presents an opportunity to develop a proof-of-concept learning task in the CAMBADA team.

To minize the state dimensions and increase generalization, the problem is defined according to a robot centered coordinate frame. Since no linear movement is required, the state is a two-dimensional vector composed of the angular error to the target, and the robot angular velocity, $\langle \theta_e, \omega \rangle$. The action set is composed of angular velocities, in the robot frame. These are then converted to motor setpoints, using differential inverse kinematics, in a separate module of the CAMBADA software architecture. The problem can be simplified if we only pass the agent the absolute value of its orientation error, and a modified angular velocity value, where whether it is positive or negative means that the velocity is directed to the target orientation or away from it. Additionaly, we also restrict the actions to be values greater than or equal to zero, so that the actual command sent to the motors of the robot is always towards minimizing the orientation error. The action is modified by the signal of the robot orientation error, meaning that when the orientation error is greater than zero, the action that is returned by the agent is applied directly; otherwise, we apply a negative action with the same magnitude as the one returned. This way, we are able to reduce the orientation error range from $[-\pi, \pi]$ to $[0, \pi]$. We settled for an action set with three actions: $[0, \frac{\pi}{2}, \pi]$ rad/s.

In this learning task, we intend to minimize the time it takes for the robot to face the desired orientation. Not only has the robot to minimize the orientation error as fast as possible it also has to actively maintain the error close to zero afterwards. We are then facing a regularization problem. The reward function chosen follows (4), with $C = 0.01$ and $\delta = 0.1$ rad.

1) Learning procedure: For this behavior, we chose to use a neural network with 3 input neurons, 2 layers of 10 hidden neurons each, and an output layer of a single neuron. The 3 input neurons account for the 2 state dimensions and 1 action dimension. The pattern set was approximated with 300 epochs of the RProp algorithm. The Q-Batch update-rule was applied with no discounting, that is $\gamma = 1.0$. A common technique applied in the NFQ framework is the addition of *hint-to-goal* patterns. These patterns are added to the pattern set $\mathcal{P}$ with output zero, that hint the learning agent to the specified states, while also helping to clamp the value of the $\hat{Q}$ close to zero in these states. The *hint to goal* heuristic was used, with an artificial experience set composed by the target state ($\langle \theta_e = 0, \omega = 0 \rangle$), repeated 100 times for each action from the action set.

The learning procedure interleaves phases of experience gathering with policy evaluation. For every interaction phase, 10 loops of policy evaluation are performed, to speed up the learning procedure. Each iteration of experience collection is composed by 10 learning episodes, with each episode lasting for 100 control cycles. In the CAMBADA platform, a new control cycle is started every 0.033 seconds, so, since there are no terminal states, each learning experiment accounts for around 33 seconds of interaction time. At the start of each episode a random target orientation is chosen sampled from a uniform distribuition.

2) Results: This task was applied initially in the simulation environment. A good controller was found after 38 learning iterations, which accounts for roughly 14.5 minutes of interaction time. While the learned controller is able to reduce the angular error, the used hand-tuned PD controller is more robust. We attribute this mainly to the reduced number of actions and also because, for situations when the orientation error is small, these are too strong, i.e. the controller is over actuated. Nonetheless, an acceptable result is achieved. This goes to show how, even with a less-than-optimal action set, it is possible to learn a suitable controller. However, if we were to use this behavior in competition situations, or if our sole objective was to surpass the performance of the explicitly coded behavior, we would need to find a better action set. As this is only a proof-of-concept, we chose to focus our efforts in the rest of the tasks.

Learning on the real robot lasted for 24 learning iterations, during which 16800 experience tuples were gathered. This accounts for around 9 minutes of interaction time. In Figure 3, we can see a comparison of the absolute orientation error between the explicitly coded behavior and the learned behavior, averaged after 10 experiments. Again, we see that the learned behavior is able to reach the target orientation very quickly. The robot omniwheels appear to have a very tighter grip on the field surface, so the robot never overshoots as much as in the simulated environment.

Since good learning controllers can be obtained with a relative ease, we took the opportunity to compare the learning performance of Q-Batch with batch version of Q-Learning, (1). In order to draw conclusions over their performance, 10 learning experiments with a maximum of 50 learning iterations, which interleave one episode of experience collection with one policy evaluation phase, were carried out for each learning rule. The results were aggregated and are displayed in Figure 4. As

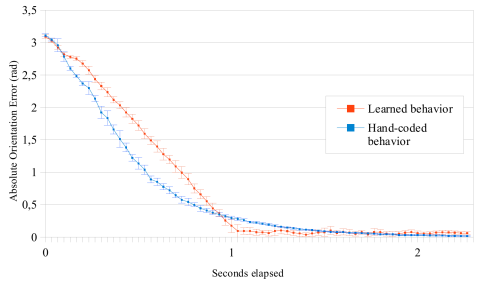

Fig. 3. Performance comparison between the learning and the hand-coded controllers in a real environment on the rotating learning task.

we can see, the Q-Batch update rule is faster to achieve a lower mean cost per cycle. Also, after the first 5 learning iterations, the average results for Q-Learning were never better than the average Q-Batch results. This is an interesting observation, because it shows that Q-Batch can reach better performances than Q-Learning in short learning times.

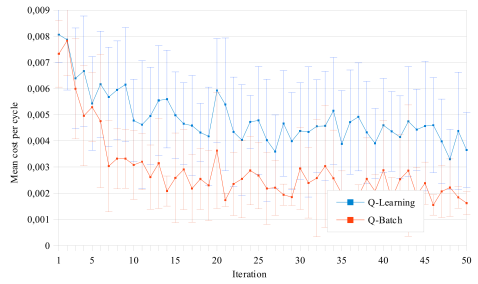

Fig. 4. Learning performance over time of the two update-rules used on the rotating learning task.

B. Dribbling the ball

Dribbling is one of the basic set of skills of a soccer robot. However to dribble across the field while performing sharp turns without losing the ball is a very challenging task. Additionally, the Middle Size League rules stipulate that a robot may not cover more than a third of the ball, which means that the contact between the ball and the robot is very small.

In this task, we intend to develop a low-level controller that is able to dribble a ball in a given direction in the minimum time possible. While the CAMBADA robots have a dribbling behaviour, the hand-coded existing solution performs large turns. The controller has to learn how to perform a sharp turn without losing the ball. The task is defined as a regularization problem, this means that after the angular error is close to zero the robot has to maintain the dribbling heading. This can be seen as an extension of an existing learning task in the literature [11].

The state vector of this behavior includes the robot relative orientation error to the target direction, its relative linear and angular velocities and a signal indicating whether the ball is engaged or not. This binary signal is further filtered, so that it is only negative after five sequential control cycles in which the ball was perceived as not engaged by the robot. This is because the robot may lose control of the ball for only one or two control cycles, but regain control right after. In these situations, we choose not to punish the agent, and instead consider as if control of the ball was never lost.

As was described in previous task, we can take advantage of simmetry in order to reduce the number of actions. In this behavior, we again only give the agent the absolute value of the orientation error, and use its sign to modify actions before they are applied. Actions are 3-tuples consisting of desired velocity values in the robot coordinate frame, and follow the form $\langle v_x, v_y, \omega \rangle$, where the y axis points to the front of the robot. Six actions were defined: $\langle 0.0, 2.5, 0.0 \rangle$, $\langle 0.0, 2.0, 2.0 \rangle$, $\langle 0.0, 1.5, 2.5 \rangle$, $\langle 1.5, 1.5, 2.5 \rangle$, $\langle 1.0, 1.0, 3.0 \rangle$ and $\langle -1.0, 1.0, 3.0 \rangle$. The actions are later converted to motor setpoints.

The reward function was designed to have the robot reduce the heading error as fast as possible without losing the ball. To achieve this the reward function gives a harsh penalty every time the robot loses the ball, as presented in (5)

$$r(s, a, s') = \begin{cases} 1 & \text{if ball lost} \\ 0.01 \times \tanh^2(|\theta_e| \times \frac{\tanh^{-1}(\sqrt{0.95})}{0.1}) & \text{otherwise} \end{cases} \quad (5)$$

1) Learning procedure: In the NFQ framework, the learning procedure alternates between 15 episodes of interactions with 10 policy evaluation phases. Each episode lasted for a maximum of 100 cycles, but would terminate earlier if the robot lost the ball during the experiment. The neural network topology used for this behavior consists of 8 input nodes, 2 hidden layers of 20 nodes each and one single output node. The pattern set was approximated over 300 epochs of the RPROP algorithm. The Q-Batch update rule was used to generate the pattern set with a discounting factor of $\gamma = 0.99$. An artifical set of "hint" transition tuples was introduced, with state $\langle \theta, v_x, v_y, \omega, engaged \rangle = \langle 0, 0, 2.5, 0, 1 \rangle$, repeating 100 times for each action of the action set, and target output of zero. At the start of each episode the ball was placed away from the robot to ensure an initial velocity upon ball possession. After the ball was engaged, an uniformly random dribbling direction was chosen signalling a start of experience collection.

2) Results: Learning in the simulated environment was achieved after 34 iterations. Around 17500 experience tuples were gathered during learning, which means the controller was learned under 10 minutes of interaction time. the learned behavior is much more sharper than the hand-coded one. In fact, using the learned behavior, the robot is able to rotate around the ball, even while carrying significant speed. This way, the robot could complete a full 180 degree turn in less space.

Learning in the real robot was achieved after 18 learning iterations. During learning, the robot collected 6535 experience tuples, which means only a little over 3.5 minutes of experimentation were needed. The whole learning process took around 1.5 hours, including batch training, preparation and execution of learning experiments. Figure 5 shows the performance of the learning controller and the hand-coded solution on the real robot. A square is placed every second on each trajectory, in order to compare how long each behavior took to finish the test. As we can see, the learned behavior is sharper, and was able make a harder turn. Additionally, it achieved faster speeds after having reached the target orientation, and completed the test after 134 control cycles. The explicitly coded behavior, on the other hand, is softer and needs more space to achieve the same curve, and never moves as fast once

it is facing the target direction. It finished the test after 164 cycles. However the increased speed comes at a cost, since the learned controller still looses the ball 30% of the time as opposed to 10% on the hand-coded controller.

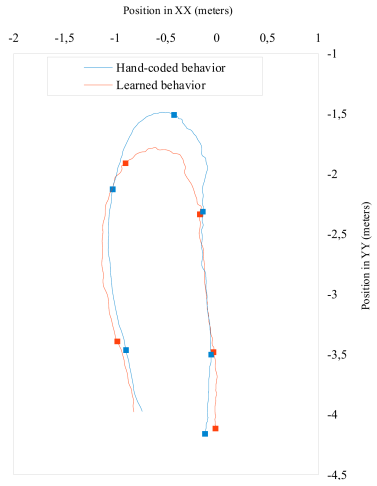

Fig. 5. Performance comparison between learning and hand-coded controllers in a real environment on the dribbling learning task. Squares are placed on each trajectory every second, allowing for a temporal comparison. For each trial, the robot started close to the (0, -4) position. It then grabbed the ball and moved forward. The dribbling behavior started after the robot reached an YY position of greater than -2.5 meters.

C. Receiving a pass

Passes are a very important part of a soccer game. For robots to succed in performing this task, receiving the ball efficiently must be achieved. While humans do it naturally, for a robot this is a very challenging task [12].

Note that the task is not to gain possession of the ball as quickly as possible, in other words to intercept the ball. To receive a pass, the robot has to gain control of the ball with the minimum *movement* possible. This way a pass can be used to gain a strategic advantage during a game. The objective of this behavior is for the robot to position itself in the ball path and absorb the ball speed upon contact, with the minimum required movement, so as to not forfeit its current position. This greatly increases the difficulty of the task. Since what we are trying to optimize is only the linear movement of the agent, we chose to delegate the task of facing the ball to a PID, thus simplifying the problem.

After careful analysis of the problem, it becomes obvious that we can reduce the state space if instead of considering a global coordinate frame, we consider a coordinate frame centered on the ball with the XX axis pointing towards the movement of the ball. If the ball stops the XX axis points towards the robot. The final formulation of the state vector includes 6 features: the ball speed, robot position (x,y), robot velocity (x,y) and the number of control cycles the ball is engaged. We also take advantage of simmetry along the XX axis of the coordinate frame, allowing to represent the component YY of the robot position in absolute values and the same component of the robot velocity to indicate if it is pointing towards the ball path or not.

The fact that we model the problem by considering coordinates as being in the axis of movement of the ball allows us to simplify actions and reduce the size of the action set. Actions are defined to be two dimensional vectors of the form $\langle v_x, v_y \rangle$, which correspond to target velocity values in the specified coordinate frame. This means that actions should have the same effect regardless of the orientation of the robot, thus taking advantage of the CAMBADA holonomic motion. The action set includes 6 actions: $\langle 0, 0 \rangle$, $\langle 0, -1 \rangle$, $\langle 1, 0 \rangle$, $\langle 1, -1 \rangle$, $\langle 0, -0.5 \rangle$ and $\langle 1, -0.5 \rangle$. Positive v_x actions move the robot away from the ball along its path to absorb the ball incoming velocity, and negative v_y actions move the robot towards the ball path.

The reward function encodes a series of penalties that are combined to give rise to the reception of the ball. We provide a penalty according to YY component of the robot position, p_y, to force the robot onto the ball path as soon as possible. To prevent the movement of the robot along the axis of the ball movement, a penalty is added according to the robot velocity in XX component, v_x. A constant term is added to force the robot to move if the ball stops. Additionally to ensure that the robot attemps to get the ball, a large penalty is given if the ball moves too far away from the robot, $p_x < -0.5$, where we consider that the pass has failed. If the ball is engaged for more than 5 cycles no penalty is given. The overall reward function is presented in (6).

$$
\begin{aligned}
r(s,a,s') &= \begin{cases} 0 & \text{ball received} \\ 1 & \text{if ball lost} \\ 0.01 + r_y + r_x & \text{otherwise} \end{cases} \\
r_y &= 0.01 \times \tanh^2 \left(|p_y| \times \frac{\tanh^{-1}(\sqrt{0.95})}{0.1} \right) \\
r_x &= 0.01 \times \tanh^2 \left(|v_x| \times \frac{\tanh^{-1}(\sqrt{0.95})}{0.1} \right)
\end{aligned}
\tag{6}
$$

1) Learning procedure: To learn this behaviour, the robot collects experiences in sets of 10 episodes, each with a maximum of 100 control cycles. An episode ends sooner if the ball is lost or caught. After the interaction phase ended, 10 iterations of policy evaluation were performed. The Q-Batch update rule was used to generate pattern sets, with a discount factor of $\gamma = 1.0$. The pattern set was approximated over 300 RPROP epochs. The neural network used to approximate the Q-function has 8 input nodes, 2 hidden layers of 20 nodes each and an output layer with a single node.

Learning in the simulator allowed for a controlled repeatability of the experiments. During learning the robot positioned in the center of the field and with the ball placed randomly, being the initial distance between them no less than 4 meters and no greater than 6 meters. The ball starts moving to a target location, which is determined randomly around the robot, being at most 1 meter away from it. This way, we try to reproduce situations where the passer produces a somewhat inaccurate kick. We allow for some control cycles to pass before starting the actual learning episode, to reduce noisy measurements of the ball velocity.

Learning in the real platform presented more challenges. To allow for some repeatability, the ball was rolled down a ramp at different heights to ensure different initial ball velocities. Similarly to the simulated environment, the ball was thrown

at different directions around the robot to ensure a robust ball reception controller.

2) Results: Using the simulator, 34 learning iterations were needed for a good control policy to be obtained. The learning agent needed less than 9 minutes of interaction time, and a total number of gathered experience tuples close to around 16000. The hand-coded behaviour is able to receive 80% of the passes performed. The learning behaviour is only able to capture 50% of the passes.

In the real platform, a suitable policy was obtained after 28 learning iterations. This amounted to close to 12 minutes of interaction time, during which 21303 experience tuples were collected. The actual time involved, including preparation and off-line Q-function approximation, was around 2 hours.

The learned solution outperforms the existing hand-coded solution, when receiving passes on the real platform. While the hand-coded solution is only able to receive 10% of the passes, the learned controller was able to receive 50% of the performed passes. This is mostly justified by the amount of noise present in the ball velocity estimation, to which the hand-coded solution seems to be more susceptible.

Figure 6 presents a comparison of the performance of the two controllers. For a better visualization, squares and circles are placed on the robot and ball trajectories, respectively, every third of a second. Also, circles with a dark outline represent the ball under the control of the robot. As it is visible, the learned behavior is faster to move its final position, where it grabs the ball safely. The hand-coded behavior, on the other hand, starts by moving sideways, but then changes the direction of its movement and starts backing up. In this trial, the hand-coded behavior failed to grab the ball, as it was not aligned with its trajectory at the moment of impact.

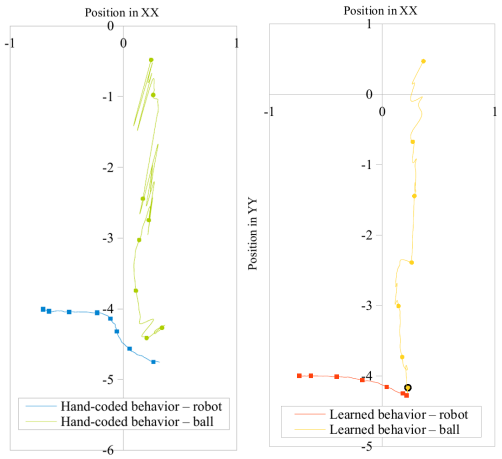

Fig. 6. Comparison of the hand-coded and the learned behavior when receiving a pass. Both the robot and the ball trajectories are marked with squares and circles every third of a second. A circle with a black outline signifies the robot has grabbed the ball. The receiving behavior was enabled shortly after the robot detected that the ball started moving.

V. Conclusion

This paper presents the application of the Q-Batch update-rule in the context of learning in robotic soccer. This represents the first application of this update-rule in real robotics. Three tasks were developed, with increasing difficulty. The design of the learning tasks is described, with a focus on hardware abstraction, allowing other teams to implement the tasks on different platforms. The tasks were applied on simulation and the real platform. However a focus was given to learning on the real platform where better results were obtained.

The successful application of the Q-Batch update-rule opens up opportunities for the development of benchmarks to evaluate the learning performance against other learning methods. Additionally, this work represents the first application of RL methods in the CAMBADA team, allowing for further research in this field on the on-going project.

Acknowledgments

This work was supported by project Cloud Thinking (funded by the QREN Mais Centro program, ref. CENTRO-07-ST24-FEDER-002031).

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge (MA): MIT Press, 1998.

[2] M. A. Wiering and M. van Otterlo, Eds., *Reinforcement Learning: State of the Art*, ser. Adaptation, Learning, and Optimization. Springer, 2012, vol. 12.

[3] C. Szepesvári, *Algorithms for Reinforcement Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning, R. J. Brachman and T. G. Dietterich, Eds. Morgan & Claypool, 2010.

[4] S. Lange, T. Gabel, and M. Riedmiller, "Batch Reinforcement Learning," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds. Springer, 2012, ch. 2, pp. 45–74.

[5] M. Riedmiller and H. Braun, "A direct adaptive method for faster backropagation learning: the RPROP algorithm," in *Proceedings of the IEEE International Conference on Neural Networks*, H. Ruspini, Ed., San Francisco, CA, 1993, pp. 586–591.

[6] M. Riedmiller, "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method," in *Proc. of the European Conference on Machine Learning*, ser. Lecture Notes in Computer Science, J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, Eds., vol. 3720. Porto, Portugal: Springer, 2005, pp. 317–328.

[7] J. Cunha, N. Lau, and A. J. R. Neves, "Q-Batch: initial results with a novel update rule for Batch Reinforcement Learning," in *Advances in Artificial Intelligence - Local Proceedings, XVI Portuguese Conference on Artificial Intelligence, Azores, Portugal*, September 2013, pp. 240–251.

[8] M. Lauer, S. Langue, and M. Riedmiller, "Motion estimation of moving objects for autonomous mobile robots," in *Kunstliche Intelligenz*, vol. 20, no. 1, 2006, pp. 11–17.

[9] J. Cunha, N. Lau, J. M. O. S. Rodrigues, B. Cunha, and J. L. Azevedo, "Predictive Control for Behavior Generation of Omni-Directional Robots," in *Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence*, ser. Lecture Notes in Artificial Intelligence, vol. 5816. Aveiro, Portugal: Springer-Verlag Berlin / Heidelberg, October 12-15 2009, pp. 275–286.

[10] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine Learning*, vol. 84, pp. 137–169, 2011.

[11] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, vol. 27, no. 1, pp. 55–73, May 2009.

[12] G. Corrente, J. Cunha, R. Sequeira, and N. Lau, "Cooperative Robotics: Passes in Robotic Soccer," in *Proceedings of 13th International Conference on Autonomous Robot Systems and Competitions*, Lisbon, Portugal, April 2013, pp. 82–87.