Interactive Machine Learning Applied to Dribble a Ball in Soccer with Biped Robots

Carlos Celemin [1][0000-0001-8880-0966], Rodrigo Perez[1][0000-0002-9503-7783], Javier Ruiz-del-Solar[1][0000-0003-2965-633X], and Manuela Veloso[2][0000-0001-6738-238X]

[1] Advanced Mining Technology Center and Department of Electrical Engineering, Universidad de Chile, Santiago, Chile
{carlos.celemin,rodrigo.perez.d,jruiz}@ing.uchile.cl
[2] Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
veloso@cmu.edu

Abstract. An Interactive Machine Learning (IML) approach for training a dribbling engine for humanoid biped robots in RoboCup competitions (Standard Platform League) is presented. The proposed dribbling approach solves two decision problems: the determination of the dribbling direction and the calculation of the walking velocities required for pushing the ball toward the desired direction. Moreover, the prediction of the position of moving balls is used for improving the dribbling performance, when it is needed to intercept a moving ball. A combination of batch and incremental learning is used for shaping the policies of the dribbling controller. Results obtained from previous RoboCup competitions, and also from specific experiments, validate the proposed methods.

Keywords: Robot soccer, learning from demonstration, robot behavior, human feedback.

1 Introduction

In RoboCup soccer competitions, dribbling a ball is an important behavior needed for either defensive or offensive playing. The challenge for a dribbling engine of a robot is to navigate through the field and to avoid opponents, while keeping the ball possession; i.e. the robot needs to push the ball to a target, but the path of the ball is also controlled.

Most of the works reporting dribbling mechanisms are in the context of the wheeled robots [1][2], and simulation leagues [3][4]. For humanoid biped robots, most of the decision making systems developed for dribbling are a combination of walking and kicking behaviors, like the Finite State Machine (FSM) presented in [5] that is refined with human feedback. The same strategy of dribbling is used in [6], where a complex system of kicks is developed. Therefore, the robot walks toward the ball and according to the situation, the most convenient kick is selected, i.e. the system is mostly supported on the potential of the kicks. In contrast, in [7] is presented a system of In-walk kicks that is combined with a very efficient gait. However, there are some other approaches that do not divide the dribbling process between walking and kicking at all. For instance, [8] introduced a method for a gait that plans the steps of the walk for performing kicks while walking. A RL and prior knowledge based strategy that pushes the ball without kicks is presented in [9], it simply executes the task by walking against the ball. Our first work related to dribbling with humanoid robots was based on the same concept for pushing the ball [10]; a controller computes the velocity requests for the walking engine; in that case the agent was trained with RL and a Genetic Algorithm. In [10] is proposed to split the behavior into two simpler subtasks: A*lignment* for positioning the robot in front of the ball and the target; and *Ball-pushing* for walking against the ball with the appropriate velocity in order to keep its possession. In [11], the learning process of our system was improved using Interactive Machine Learning (IML). We proposed COACH, an interactive framework that allows users to provide corrective feedback to a policy, while this is being executed. In that work, COACH was used for training the sub-task *Ball-Pushing*. The results showed that the human advice during execution lets the learning process to attain best performances in less time than the RL scheme proposed in [10].

IML methods in general, and COACH [11] in particular, allow non-expert users without deep knowledge of the system properties, to train and adapt systems to specific requirements. This characteristic is important in situations where final users need to adapt a system to new conditions. For instance, in the context of RoboCup teams, wherein students join the project for short time periods, they work and focus on specific problems, but sometimes it is required to change the behavior of external systems developed by members who are no longer in the team. With this concern, we focused on applying IML strategies that allow tuning the dribbling system by users who do not know exactly how the decision-making system internally works.

In this work the methodology applied for training our last version of the dribbling engine, based mostly on IML, is introduced. This new system does not split the dribbling behavior in two sub-tasks, but faces the dribbling problem by addressing two main decisions: “how to push the ball? and “where to push the ball?”. This last aspect was not considered in our former works [10][11], because in those works the dribbling direction was always towards the opponent’s goal. The proposed dribbling system controls the walking velocities for pushing the ball to a specific target, but the dribbling direction while navigating considers the ball’s and opponents’ relative positions. The system is also able to take into account the fact that a ball is moving, and to predict its position for better approaching it.

The work is organized as follows: In Section 2, the modules that compose the proposed dribbling engine are described. In Section 3, the IML approaches used for training the dribbling modules are introduced. The proposed methodology is validated in Section 4, using results collected during official RoboCup competition, but also with experiments that test each proposed module. Finally, in Section 5 conclusions of this work are drawn.

2 Dribbling Strategy

As already mentioned, the dribbling task is split in two decision problems: *where to dribble?*, which is related to decide the direction where the ball is pushed; and *how to dribble?*, which is concerned to the computation of velocity requests necessary for walking against the ball properly, pushing it towards the desired direction.

2.1 Dribbling Controller

This module is the responsible of solving the second decision problem mentioned in the previous paragraph. The controller computes the velocity requests to the walking engine in order to push the ball in the direction computed by the module that decides where to dribble, i.e. the linear velocities v_x, v_y and the rotation velocity v_θ. The state space is represented with the angles and distances shown in Fig. 1, where ρ is the robot-ball distance, γ is the robot-ball angle, and φ is the dribbling direction-robot complementary angle. In Fig. 1 the dribbling direction is the angle between the ball and the opponent goal, but in other cases it could be a different direction (e.g. due to obstacles). The angles γ and φ have to be zero for the robot to be aligned to the ball and target, and the distance ρ has to be low in order to keep possession of the ball.

The assumption taken in this controller is very intuitive. It is supposed that each velocity axis is in charge of decreasing one of the variables γ, φ, and ρ. For instance, if the robot rotates with v_θ proportionally to the angle γ, the robot would be aligned to the ball. If simultaneously the sideward velocity v_y is proportional to the angle φ, the robot would walk by a circle centered in the ball position until it is aligned to the ball and the target; when the robot is aligned, it can walk against the ball with velocity v_x proportional to the distance ρ in order to walk faster when it is far, and slower when it is near for pushing to a close distance. In [10] was discussed a simple linear proportional controller based on the previous description, but in order to walk more effective paths, that work proposed to use a Takagi-Sugeno Fuzzy System as non-linear approximation for mapping from states to velocities, keeping the same assumption but with state dependent gains for the proportional controller.

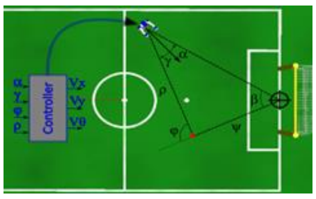

Fig. 1. Variables observed by the robot.

2.2 Dribbling Direction Computation

The system described before had the target fixed in the center of the opponent's goal, and the direction to dribble was always towards that target. Another system is required to compute different directions in order to dribble towards the goal, but avoiding collisions and decreasing the risk of pushing the ball out from the field. Then, this system is related with the problem *where to dribble?* Since the robot can detect the opponents with its camera, it is possible to have a map of obstacles. However, differently to other leagues, in the SPL it is a very challenging problem to model and predict the behavior of the opponent team in this adversarial environment. Therefore, in our UChile Robotics Team, the opponents' positions are tracked but are assumed static for navigation purposes. Due to this assumption, and that there is no prediction of the future states, the navigation problem with the dribbling is not tackled with long term path planning approaches, rather, the strategy is to compute the dribbling direction in every time step, based on the updated state of the environment (i.e. the positions of the other robots and the ball).

The dribbling is not a typical navigation problem, since it has to be controlled the robot and ball positions, taking into account that the ball is moved with a very constrained strategy. Moreover, in our approach, the trajectory to go and push the ball is defined by the controller; and a high-level system computes the direction that the ball has to follow. Our proposal is inspired by the potential fields [12][13] methods in which there are attracting and repulsive objects, whose forces depend on the distances. Nevertheless, this application has some specific characteristics that make necessary two considerations:

1. The attractive force of the opponent goal is constant, and does not decrease with the distance.
2. The opponents and posts are obstacles that need to be avoided. The repulsive force works when the robot starts dribbling far from any obstacle. However, it is possible that a dribbling episode starts with the ball in the possession of an opponent, i.e. the ball can be just beside the feet of the opponent. In those cases, the direction that minimizes the possibility of collision is an angle orthogonal to the line between the opponent and the ball.

In Fig.2 are depicted some examples to show why is better to push the ball perpendicularly to the opponent when the ball is in the opponent's possession. The blue arrow is the direction to dribble with minimum probability of collision with the opponent (red jersey), taking into account that the goal is in the right hand side of the picture. In Fig. 2 a) the ball is beside the opponent; the trajectories red and yellow are likely to make a collision with the opponent before and after pushing the ball respectively. The red and gray directions can be the result of combining the attraction of the goal and the repulsive force of the obstacle. The gray path could be the case when the goal is very far and its force has low incidence whereas the obstacle has a high repulsive force. The result is a dangerous movement that definitely would end up with a punishment for our robot. In Fig 2 b) the ball is very close to the opponent but behind it. In that case the repulsive force and the attraction of the goal in the right hand side of the picture would sum to the gray direction, which would end up with the robot dribbling by the path of the same color, which is also dangerous. On the other hand, the blue direction is the safest to take possession of the ball and keep dribbling to the goal in the right.

With the previous considerations it is proposed a system that computes a sum of weighted candidate angles to dribble. Each candidate angle is related to an object of interest in the field. These objects are: the center of the opponent's goal, the goal-posts, the final line of the field, and every opponent-robot in the field. The weights of each angle are obtained as the product between two factors: an importance weight k_o associated to each kind of object, and a distance measure between the object and the ball. The distance measure is calculated with the Euclidean distance evaluated in a Gaussian kernel, and then its values are in the range [0-1]; being this value close to one if the object is very near to the ball. The weight of the candidate angle to the goal depends only on its importance weight k_g, as shown in (1).

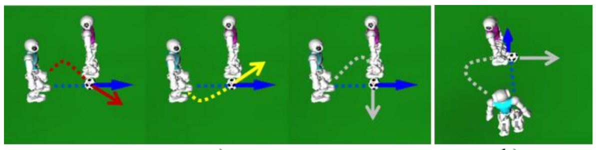

Fig. 2. Ball Dribbling paths when the ball is close to the opponent (blue path is desired).

The candidate angles to dribble depend on each object: for the final line the candidate direction is an angle orthogonal to the line with direction to the own side of the team. This is for minimizing the risk of pushing the ball out of the field when it is very close to the line. The candidate direction of the opponent goal is the angle between the ball and the goal; this is the only attracting object. As described before the obstacles (opponent robots and goalposts) have candidate directions, which are perpendicular to the obstacle-ball line. Since there are two possible orthogonal angles, the chosen candidate is the one that has lower difference with the opponent's goal direction. The linear combination of the angles is normalized by the weights, for computing the dribbling direction D:

$$D = \frac{a_g k_g + \sum_{o=1}^{No} a_o w_o}{k_g + \sum_{o=1}^{No} w_o} = \frac{a_g k_g + \sum_{o=1}^{No} a_o k_o \exp(-\frac{d_o^2}{2\sigma_o^2})}{k_g + \sum_{o=1}^{No} k_o \exp(-\frac{d_o^2}{2\sigma_o^2})} \qquad (1)$$

where a_g is the angle of the ball to the opponent goal, k_g is the constant importance weight associated to the goal, a_o is the dribbling candidate angle associated to the o-*th* interest object in the field, w_o is the distance dependent weight associated to each interest object. This is computed with the importance weight k_o and the Gaussian distance that takes the Euclidean distance d_o between the ball and the object o, and has an associated standard deviation σ_o to each type of object that defines a security distance, where the object has more influence on the dribbling direction.

With this direction D, the dribbling controller of section 2.1 computes the walking velocities based on the variables depicted in Fig. 1, but the angle φ is computed with respect to D rather than the angle between the ball and the center of the target.

2.3 Using the Dribbling Behavior for approaching a Moving Ball

The dribbling controller is always used when the robot is attacking even without having possession of the ball, e.g. when the robot is trying to “steal” the ball from an opponent or intercepting a pass done by either a teammate or an opponent. In order to address this challenge properly, the robot should move to a predicted position of the moving ball, like human players do.

Thus, the strategy applied in this work is to keep the described structure of the dribbling controller, and to track a predicted position of the ball instead of the actual one; the robot predicts the trajectory of the ball and evaluates if at certain point it is possible to intercept it. In case it is possible, then the robot moves to that position. Alternatively, the robot can move to a position that is close to the predicted final position of the ball.

3 Interactive Training of the Dribbling Engine

As it was explained in Section 1, it is necessary to make this system easily trainable by non-experts, since it can be required to change the parameters of the system during the competition before a game. For instance, when the walking engine is modified, the dribbling controller needs to be tuned again. The same happens when the team’s strategy needs to be fixed for facing the characteristics of a specific opponent team. Interactive approaches are used for letting the users to adapt the dribbling controller module and the dribbling direction computation module easily using incremental and batch learning. Although the dribbling engine works by combining the dribbling controller and the dribbling direction computation modules, both are trained independently for the ease of the process. Then, for training the dribbling controller the direction to dribble is set always toward the opponent goal. For training the dribbling direction computation module, demonstrations can be provided using static game situations, without the need of running the dribbling controller (Section 3.2).

3.1 Incremental Learning for the Dribbling Controller

In [11] COACH was used for training policies with corrective feedback provided by human teachers during execution time. This feedback is a binary advice of how the executed continuous action has to be modified (increased or decreased). Then, using Stochastic Gradient Descent (SGD) the framework updates the policy; therefore the next time step the policy has a new set of parameters. In [11] the framework was presented for shaping approximations based on linear models of radial features. However, those principles can be applied to different types of approximation like the fuzzy model used in this problem. In Algorithm 1 is described how COACH works. More details can be seen in [11].

The users may have an insight of how an action can be modified for obtaining a better performance, but he/she hardly knows precisely the magnitude of the required correction. With the previous assumption, COACH updates the policy setting a constant error magnitude e, whereas its sign is given from the feedback binary signal.

For training a policy, lines 4-11 of Algorithm 1 are executed every time step during execution: first the agent observes the world (line 4), then the policy is computed and the action executed (lines 5-6), it checks if there is human advice: h has a value of 1 if the teacher says “increase the action”, -1 for “decrease the action” and 0 if there is not feedback. If there is feedback, the error assumption is computed (line 10) and the SGD update of the policy is carried out taking into account specific considerations of the used approximation model. This framework is applied while the user sets the environment with several varied conditions of ball and robot initial positions, then provide feedback in order to shape the policy for achieving a desired behavior.

Algorithm 1: Basic Structure of COACH (simple framework).

```
1:  e ← constant   // error magnitude
2:  α ← constant   // learning parameters
3:  while true do
4:     $\vec{s}$ ← getStateVec()
5:     $a = P(\vec{s})$
6:     TakeAction( $P(\vec{s})$ )
7:     wait for next time step
8:     h ← gethumanCorrectiveAdvice()
9:     if h=! 0
10:       $error \leftarrow h \cdot e$
11:       updatePolicyModel($\alpha, error, \vec{s}$)
12:    end if
13: end while
```

3.2 Training of the Dribbling Direction Computation Module

In the dribbling direction model expressed by (1) it is necessary to tune the parameters k_o for all the interest objects mentioned in 2.2, and their deviations σ_o, which define the distance where each object o-*th* has more influence. The procedure for training the module consists in two alternative stages (the second one is optional): first, a batch learning process based on a dataset of instances demonstrated by the user is executed. Then, if the user considers necessary a local update, a second stage of tuning is carried out based on corrective feedback provided incrementally by the user with COACH (Algorithm 1 is applied). The second stage of tuning is not required when the user is satisfied by the resulting model of the batch learning process.

Batch stage. It is collected a dataset of several static game situations wherein the user selects the desired direction to dribble. An interface shows an inactive scenario and allows the user to change the positions of the ball, and the opponent robots, but also to set the desired dribbling direction. Then, the positions (input) and direction (output) are attached as an instance of the demonstrations dataset. In other words, for gathering the direction demonstrations it is not necessary to have the dribbling controller running, and this can be done without active robots. With the dataset of situations and decisions, it is performed an optimization process for obtaining a set of parameters that fits the model to the data based on a Genetic Algorithm; in this case a set of parameters is represented by an individual that is evaluated by the fitness function selected for this problem and discussed below.

For selecting an appropriate fitness functions it must be taken into account that human teacher demonstrations can include instances that are noisy, inconsistent or ambiguous [14, 15]. For instance, in this context users can provide instances with different directions to dribble for similar states. The inconsistent data can be considered as outliers in the demonstrations set that do not need to be fitted with the model. However, with the typical mean square error (MSE) that takes into account only second order statistics and that is used in regressions, it is assumed Gaussianity in the error distribution. Then, the shortcoming of this is that, the outliers can have a large impact in the process of matching the model to the demonstrations.

Some works have used information-theoretic measures to face similar problems. For example [16] applies the minimization of the error entropy (MEE), which is equivalent to minimize the distance between the PDF of the desired data and the model output. This helps to fit the model to most of the data, and drops the importance of outliers that are far from the PDF, which contribute to have a high MSE. For this reason we selected as fitness function the MEE criterion described in [16].

Incremental Stage. After the batch stage, or if the user considers to fix the system's behavior for a specific situation, the user sets the particular ball and opponent positions with the interface used for collecting the demonstrations, and provides corrective feedback in the angles domain. While the advice is provided, COACH updates the model's parameters of (1) using SGD, with the derivatives of the model with respect to the parameter p_o, which can be k_o or σ_o in (2), where h is the human advice for increasing or decreasing the action, and η is the step size.

$$\Delta p_o = \eta h \, \frac{\partial D}{\partial p_o} \tag{2}$$

4 Experiments and Results

In participations to RoboCup competitions, the developed dribbling engine has had an important role as our (UChile Robotics Team) main attack strategy, but also in the defensive plays. Since most of the scored goals (GF) were derived from dribbling plays (65% in RoboCup 2016), we compare in Fig. 3 the average scored and allowed goals per match in the last 3 participations respecting the performance of 2013, when the team did not have a dribbling behavior. In RoboCup 2014, we used our first dribbling approach [10], and the average number of GF increased in 50%, while the allowed (GA) was reduced by 15%. In RoboCup 2015 and 2016 the dribbling strategy used was the presented in this work, except for the prediction strategy for approaching moving balls. For those years the average of GF raised in ~120%, while the percentage of the GA decreased in about 55%. Data of these results is available online[1].

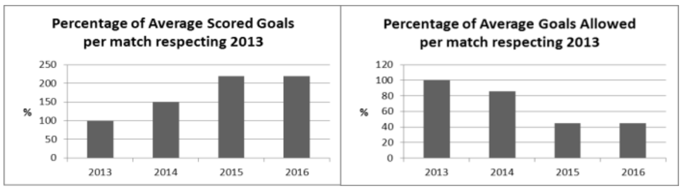

Fig. 3. Percentage of Average goals per match compared to the performance of 2013.

[1] http://www.tzi.de/spl/bin/view/Website/WebHome

Experiments were carried out for evaluating the performance of the different modules of the dribbling controller, and presented in the next sub sections. Some illustrations of the proposed system are shown in the video[2].

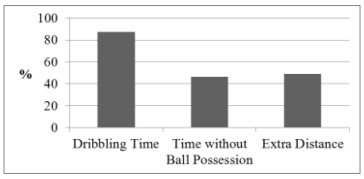

Fig. 4. Performance indices of the dribbling learned with the proposed strategy with respect to the original scheme.

4.1 Dribbling without Ball Interception

The performance of the dribbling controller trained with the approach described in [10], and used in the RoboCup 2014 competitions, is compared with the dribbling controller described in this work, and used in the RoboCup 2015 and 2016 competitions. The comparison is done in terms of time to dribble to a target, time that the robot lost ball possession while dribbling because it pushes the ball very far, and distance required to dribble to a target (the shorter the better). Several static scenarios with different initial ball and robot positions were run to dribble with the controllers obtained with both approaches. In Figure 4 is shown the increase obtained in these performance indices when the dribbling controller presented in this work is used. As it can be observed, the proposed dribbling controller and its learning method obtains a policy that reduces 12% the time for dribbling from the initial point to the target, reduces in 50% the periods of time that the ball possession is lost, and also reduces to the half the additional walked distance with respect to the length of the line given by the connection of the points initial robot position-initial ball position-target position, in order to push the ball.

4.2 Dribbling Direction Computation

This section presents experiments intended to show why it is more convenient to use an Entropy-based measure when learning using data demonstrated by human teachers. In the experiments, a user provides a set of demonstrations of the desired dribbling directions using the interface based on the simulator [17], and the MEE and the MSE are compared as fitness functions of a genetic algorithm that learns the parameters of (1) that fit to the data. The batch optimization process consists of 30 runs of the evolutionary algorithm executed for each cost function. After each optimization process, both cost functions are computed. In a second experiment, some outliers are added to the original dataset, ambiguous directions are associated to instances that represent

[2] https://youtu.be/Oc9dMag4RFw

similar situations of inputs already contained in the original set. The best performance of each experiment is presented along with the mean and standard deviation.

In Table 1, the results of the experiments with the original data demonstrated by the user show that when the optimization minimizes the MEE, it obtains even lower indices of MSE with respect to the achieved by the optimizer that minimizes the squared error. In the second experiment, the error metrics are increased as expected, but the same trend of the first experiment is obtained with wider differences, since in almost all the optimization runs using MEE as objective function, the found parameter sets computed lower MSE indices than the best of the obtained solutions from the optimization processes using the MSE cost function.

The MSE fitness function faced more local minima than the MEE, and outliers played an important role on it. The MEE is a convenient performance metric for these cases wherein a model needs to be fitted to data provided by human users that time to time could be inconsistent or ambiguous.

In Figure 5 are shown the dribbling directions computed by the system trained with the original dataset in part a). The arrows show the angle computed to dribble if the ball position is at the beginning of the arrow; the small blue points are the positions of opponent robots.

Table 1. Results of the batch learning for the Dribbling Direction module

		Mean Square Error			Entropy Error		
Dataset: Demonstrated	Fitness Function	Min	Mean	Std	Min	Mean	Std
	MSE	0.0245	0.0275	0.0028	0.0295	0.0370	0.0072
	MEE	0.0215	0.0248	0.0024	0.0231	0.0307	0.0057
Dataset: Demonstrated + Outliers		Square Error			Entropy Error		
	Fitness Function	Min	Mean	Std	Min	Mean	Std
	MSE	0.1132	0.1144	0.0011	0.0835	0.0873	0.0033
	MEE	0.1064	0.1082	0.0013	0.0794	0.0845	0.0037

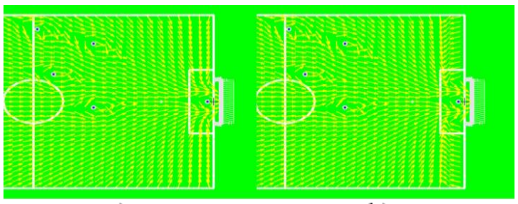

Fig. 5. Response of the Dribbling direction computation system. a) After the Batch Learning. b) After local update with incremental learning using COACH

As a proof of concept of the incremental stage, a second experiment was executed for modifying the resulting policy of the batch process that used the original database of the previous experiment. In this case, the dribbling direction strategy is modified incrementally with COACH only for the specific cases when the ball is close to the final line, specifically between the corners and the goalposts, in order to make the robot to dribble more towards the front of the goal than dribbling from the side directly to the goal. This can be more convenient because in front of the goal there are more probabilities to score a goal after a kick. In part b) of Figure 5 it is shown the response of the same system after tuning it , it is possible to see that the final line now has more influence for rejecting the ball to the own side. This change of strategy was executed for some specific situations, while the behavior keeps the same performances for the rest of the situations. Ball Interception

The strategy of intercepting the ball was validated in five different scenarios where the robot was standing in the middle of the field, and the ball was thrown from the opponent's goal to the other side of the field. Each scenario had an initial ball position and velocity, and the robot had to intercept the ball and start to dribble toward the opponent goal. The proposed strategy of going to the predicted ball position and the previous strategy of going to the current position without prediction were compared; ten executions per scenario were run for computing the average performance. The episodes were stopped when the robot got possession of the ball and was aligned to the target. The evaluation indices were the duration of the episode, and the final distance between the ball and the target, since this distance would be larger in the cases in which the robot does not intercept the ball at all. As it can be observed in Figure 6, in most of the cases the time decreased between 5 to 40% when using ball prediction. In cases like the first experiment, wherein the robot lasts almost the same time to intercept the ball with both approaches, the distance of the ball to the target was smaller at the end of the episode when the ball prediction was used; i.e. for that case, it takes the same time to intercept the ball using both systems, but with the approach based on the ball prediction, the robot intercepts the ball in a better position because the decision was: "trying to anticipate the ball movement".

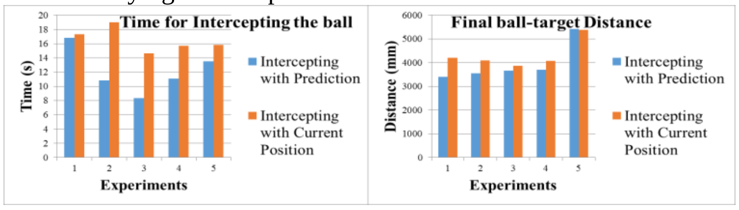

Fig. 6. Performance indices for five cases of ball interception

5 Conclusions

This work presented the dribbling strategy and its learning approach that has led to UChile Robotics Team to compete and qualify to the semifinals stage during RoboCup 2015 and 2016. The paper presents some comparative results obtained in the competitions, but also experiments that quantify the benefits of the training methods for developing the complete dribbling policy.

The reported experiments confirm that IML approaches are an excellent alternative to solve decision problems in which there is not enough capacity and observability to model the environment, and to compute a plan to achieve a goal. IML techniques benefit from the knowledge that users have about the problem, which can be shared with the agents easily, without the need of coding that knowledge.

Moreover, the results of the process for learning the dribbling direction module showed that using the EEM metric as cost function is useful in contexts where human teachers provide demonstrations that can be contaminated with ambiguous data. Additionally, it was shown that COACH makes possible to adjust the dribbling direction module behavior for specific game situations, without the necessity of gathering a new set of demonstrations and executing the batch process.

References

1. Li, X., Zell, A.: Nonlinear predictive control of an omnidirectional robot dribbling a rolling ball. Proc. - IEEE Int. Conf. Robot. Autom. 1678–1683 (2008).
2. Riedmiller, M., Hafner, R., Lange, S., Lauer, M.: Learning to dribble on a real robot by success and failure. Proc. - IEEE Int. Conf. Robot. Autom. 2207–2208 (2008).
3. Carvalho, A., Oliveira, R.: Reinforcement learning for the soccer dribbling task. 2011 IEEE Conf. Comput. Intell. Games, CIG 2011. 95–101 (2011).
4. Macalpine, P., Depinet, M., Stone, P.: UT Austin Villa 2014 : RoboCup 3D Simulation League Champion via Overlapping Layered Learning. Proc. Twenty-Ninth AAAI Conf. Artif. Intell. 1–7 (2015).
5. Meriçli, Ç., Veloso, M., Akin, H.: Task refinement for autonomous robots using complementary corrective human feedback. Int. J. Adv. Robot. Syst. 8, 68–79 (2011).
6. Barrett, S., Genter, K., Hester, T., Quinlan, M., Stone, P.: Controlled kicking under uncertainty. In: The Fifth Workshop on Humanoid Soccer Robots at Humanoids (2010).
7. Laue, T., Röfer, T., Gillmann, K., Wenk, F.: B-Human 2011–eliminating game delays. Rob. 2011 Robot Soccer World Cup XV. 25–36 (2012).
8. Alcaraz-Jiménez, J.J., Herrero-Pérez, D., Martinez-Barberá, H., Alcaraz, J., Herrero, D., Mart, H.: A closed-loop dribbling gait for the standard platform league. In: Workshop on Humanoid Soccer Robots of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids) (2011).
9. Latzke, T., Behnke, S., Bennewitz, M.: Imitative Reinforcement Learning for Soccer Playing Robots. Rob. 2006 Robot Soccer World Cup X. 47–58 (2007).
10. Leottau, L., Celemin, C., Ruiz-del-Solar, J.: Ball Dribbling for Humanoid Biped Robots: A Reinforcement Learning and Fuzzy Control Approach. In: RoboCup 2014: Robot World Cup XVIII. pp. 549–561. Springer (2015).
11. Celemin, C., Ruiz-del-Solar, J.: Interactive learning of continuous actions from corrective advice communicated by humans. In: RoboCup 2015: Robot World Cup XIX.
12. Damas, B.D.B.B.D., Lima, P.U.U.P.P., Custódio, L., Custodio, L.L.M.: A Modified Potential Fields Method for Robot Navigation Applied to Dribbling in Robotic Soccer. Rob. 2002 Robot Soccer World Cup VI SE - 6. 2752, 65–77 (2003).
13. Tang, L., Liu, Y., Qiu, Y., Gu, G., Feng, X.: The strategy of dribbling based on artificial potential field. 2010 3rd Int. Conf. Adv. Comput. Theory Eng. 2, 307–311 (2010).
14. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Rob. Auton. Syst. 57, 469–483 (2009).
15. Chernova, S., Thomaz, A.L.: Robot Learning from Human Teachers. Synth. Lect. Artif. Intell. Mach. Learn. 8, 1–121 (2014).
16. Erdogmus, D., Principe, J.C.: An error-entropy minimization algorithm for supervised training of nonlinear adaptive systems. IEEE Trans. Signal Process. 50, 1780–1786 (2002).
17. Laue, T., Spiess, K., Rofer, T.: SimRobot-a general physical robot simulator and its application in RoboCup. Lect. Notes Comput. Sci. 173–183 (2006).