# Retrieving and Reusing Game Plays for Robot Soccer

Raquel Ros[1], Manuela Veloso[2], Ramon López de Màntaras[1], Carles Sierra[1], Josep Lluís Arcos[1]

[1] Artificial Intelligence Research Institute, Campus UAB 08193 Barcelona, Spain
[2] Computer Science Dept., Carnegie Mellon University Pittsburgh, PA 15213, USA
veloso@cs.cmu.edu,{ros,mantaras,sierra,arcos}@iiia.csic.es

**Abstract.** The problem of defining robot behaviors to completely address a large and complex set of situations is very challenging. We present an approach for robot's action selection in the robot soccer domain using Case-Based Reasoning techniques. A case represents a snapshot of the game at time $t$ and the actions the robot should perform in that situation. We basically focus our work on the retrieval and reuse steps of the system, presenting the similarity functions and a planning process to adapt the current problem to a case. We present first results of the performance of the system under simulation and the analysis of the parameters used in the approach.

## 1 Introduction

The problem of defining robot behaviors in environments represented as a large state space is very challenging. The behavior of a robot results from the execution of actions for different states, if we define acting as the execution of a policy $\pi : s \to a$ (where $s$ is the current state and $a$, the action to execute in the given state). Defining each possible state and the actions to perform at each state, i.e. defining the policy, is challenging, tedious and impossible to be done completely manually. Furthermore, we have to deal with a second issue: the nature of the environment. We are working with real robots that interact with non controllable elements of the environment, which are constantly moving.

We illustrate our work in the robot soccer domain (Robocup)[2]. In this domain, we do not deal with an independent action (e.g. turn 30 degrees, kick, walk forward 100cm), but with a sequence of actions that the robots execute to accomplish their goals (e.g. dribble and shoot). We call this sequence a *game play*. Hence, the problem we address is to find out which game plays the robots should execute during a match. We focus our work on the application of Case-Based Reasoning techniques to define the actions the robots should perform in this environment, i.e. we use the CBR approach to generate the $\pi$ function. We believe that reproducing game plays from similar past situations (similar environment's description) solves the robot behavior definition problem in an easy and fast way. The approach followed in this work is to define action cases for robots to provide them with a set of cases and then have them autonomously select which case to replay.

The work we present in this paper is centered on modelling the main steps of a Case-Based Reasoning system [1]: the retrieval step and the reuse step. For this purpose, we first analyze the environment to choose the main features that better describe it and then we define an appropriate similarity function. We use different functions to model the similarity for each feature domain and then an aggregation function to compute the overall similarity.

The organization of the paper is as follows. Section 2 presents related work. Section 3 describes the robot soccer domain. Section 4 introduces the features of the environment and the formal representation of a case. Section 5 and 6 detail the retrieval and reuse steps respectively. Section 7 shows the analysis and first results of the performance of the system. Section 8 discusses the extension of the current case representation in order to model the dynamics of the game plays. Finally, Section 9 concludes the work and describes future work.

## 2 Related work

Some researchers have already focused their work on using Case-Based Reasoning techniques for deciding the best actions a player should execute during a game. Karol et al. [5] present a model to build high level planning strategies for AIBO robots. For any game situation, game plays are chosen based on the similarity between the current state of the play and the cases in the case base. The paper only presents the general model without any experiment and does not describe the different steps of the CBR approach. Wendler et al. [14] describe an approach to select soccer players' actions based on previously collected experiences encoded as cases. The work is restricted to simulation. Thus, many parameters they take into account are not considered in our domain, and also they do not have to deal with the major problems involved when working with real robots. Regarding the retrieval step, they apply a *Case Retrieval Net* model to improve the retrieval of cases in terms of efficiency.

We can also find some bibliography dedicated to solve the action selection problem, but applying other learning techniques. Riedmiller et al. [10] focus their work on Reinforcement Learning techniques applied to two different levels: moving level and tactical level. The former refers to learning a specific move, for example, learning to kick. While the latter refers to which move should be applied at a certain point, as *pass the ball*. The work is restricted to simulation, and they only used the moving level during a competition. With respect to the tactical level, they experimented with two attackers against one or two defenders. The attackers used the approach presented, while the defenders used a fixed policy. Similarly, Sarge et al. [12] present a RL approach to learn low-level skills. These skills can later be put together and used to emulate the expertise of experienced players. More precisely, they work on the *intercepting the ball* skill. They performed experiments with hand-coded players vs. learning players. They obtained positive results after one hour of learning. Finally, Lattner et al. [7] present an approach that applies unsupervised symbolic learning off-line to a qualitative abstraction in order to create frequent patterns in dynamic scenes. The quantitative data is represented by time series. In a first abstraction step, each time series is segmented into time intervals which satisfy certain monotonicity or threshold conditions. In the second step the attribute values describing the intervals are mapped onto qualitative classes for direction, speed or distance. The idea then is to create patterns based on the qualitative information of the environment (input). The result of learning is a set of prediction rules that give information about what (future) actions or situations might occur with some probability if certain preconditions satisfy. Patterns can be generalized, as well as specialized. As in the previous papers, this is only used in simulation.

Finally, CBR techniques have been also used for purposes other than action selection. Wendler et al. [13] present a case-based approach for self-localization of robots based on local visual information of landmarks. The approach is used in robot soccer, and once again, they use the *Case Retrieval Net* model. Gabel and Veloso [3] model an online coach to determine the team line-up. Based on previous soccer matches the coach reasons about the current state of the match and decides which player of his team line-up is assigned to which of the available players type. Zita and Shewchuk [4] solve a path planning problem with a system that plans a route using a city map. The global path is created using different cases from the case base. Kruusmaa [6] develops a system to choose routes in a grid-based map that are less risky to follow and lead faster to the goal based on previous experience. Ros et al [11] present an approach for robot navigation in semistructured unknown environments. Cases represent landmarks configurations that the robot should avoid in order to reach its target. Ram and Santamaría [9] and Likhachev and Arkin [8] focus their work on a CBR approach to dynamically select and modify the robot's behaviors as the environment changes during navigation.

## 3 Robot Soccer Description

The Robocup Soccer competition involves several leagues. One of them is the one we focus our work on: the Four-Legged League. Teams consist of four Sony AIBO robots. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers. The field dimensions are 6m long and 4m wide. There are two goals (cyan and yellow) and four colored markers the robots use to localize themselves in the field. There are two teams in a game: a red team and a blue team. Figure 1 shows a snapshot of the field. The robots can communicate with each other by wireless or even using the speakers and microphones (although this is not common).

A game consists of three parts, i. e. the first half, a half-time break, and the second half. Each half is 10 minutes. The teams change the goal defended and color of the team markers during the half-time break. At any point of the game, if the score difference is greater than 10 points the game ends. For more details on the official rules of the game refer to [2].

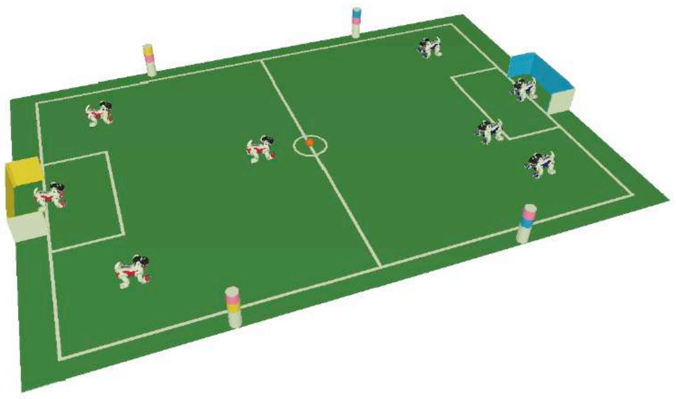


**Fig. 1.** Snapshot of the Four-Legged League (image extracted from [2]).

## 4 Case Definition

In order to define a case, we first must choose the main features of the environment (from a single robot's point of view) that better describe the different situations the robot can encounter through a game. Given the domain, we differentiate between two features' types, common in most games:

**Environment-based features** They represent the spatial features of a game. In robot soccer we consider the positions of the robots and the ball as the basic features to compare different situations, which represent the dynamics of the environment. These positions are in global coordinates with respect to the field (the origin corresponds to the center of the field). Regarding the robots, we consider the heading as a third parameter to describe their positions. It corresponds to the angle of the robot with respect to the x axis of the field, i.e. which direction the robot is facing to.

**Game-based features** They represent the *strategy* applied in the game. We use the time and the score as the main features. As time passes and depending on the current score, the strategy should be more offensive if we are losing, or a more defensive if we are winning. These features are beyond robot soccer and are applicable to other games.

In the work we present in this paper we always refer to a main robot (we could think of it as the team's *captain*; hereafter we will refer to it either as the robot or the captain) who is responsible for retrieving a case and informing the rest of the players the actions each of them should perform (including himself). We divide the description of a case in two parts: the problem description and the solution description. The former refers to the description of the environment and the game features at time $t$ from the captain's point of view (we can talk about a snapshot of the game), while the latter refers to the solution to solve that problem. Thus, within the soccer domain a case is a 2-tuple:

$$case = ((R, B, G, Tm, Opp, t, S), A)$$

where:

1. $R$: robot's position $(x_R, y_R)$ and heading $\theta$.
$$x_R \in [-2700..2700]\text{mm.} \quad y_R \in [-1800..1800]\text{mm} \quad \theta \in [0..360)\text{degrees}$$
2. $B$: ball's position $(x_B, y_B)$.
$$x_B \in [-2700..2700]\text{mm.} \quad y_B \in [-1800..1800]\text{mm}$$
3. $G$: defending goal
$$G \in \{\text{cyan}, \text{yellow}\}$$
4. $Tm$: teammates' positions.
$$Tm = \{(id_1, R_1), (id_2, R_2), (id_3, R_3)\}$$
where $id_i$ corresponds to the teammate identification for teams of 4 robots.
5. $Opp$: opponents' positions.
$$Opp = \{opp_1, opp_2, ..., opp_n\}$$
where $opp_i$ is a point $(x, y)$ and $n \in \{1, 2, 3, 4\}$ for teams of 4 robots.
6. $t$: timing of the match. Two halves parts of 10 min.
$$t \in [0..20]\text{min}, t \in \mathbb{N}$$
7. $S$: difference between the goals scored by our team and the opponent's team. The maximum difference allowed is 10. The sign indicates if the team is losing or winning.
$$S \in [-10..10]$$
8. $A$: sequence of actions (also seen as behaviors) to perform. Some examples of individual actions are $Turn(\phi)$, *Kick*(right), *Dribble*, etc. The combination of these actions result in different sequences.

## 4.1 Case properties

We can observe two symmetric properties of the ball's and robot's positions and the defending goal: one with respect to the x axis, and the other one, with respect to the y axis and the defending goal. That is, a robot at point $(x, y)$ and defending the yellow goal describes *situation 1*, which is symmetric to *situation 2* ($(x, -y)$, defending the yellow goal), *situation 3* ($(-x, y)$, defending the cyan goal) and *situation 4* ($(-x, -y)$, defending the cyan goal) (Figure 2(a)).

Similarly, the solution of a problem has the same symmetric properties. For instance, in a situation where the solution is *kick to the left*, its symmetric solution with respect to the x axis would be *kick to the right.* Thus, for every case in the case base, we compute its symmetric descriptions, obtaining three more cases. Figure 2(b) shows an example of the case previously described.

Because of the inevitable spatial nature of robots domains, interestingly a particular case can be mapped into multiple ones through different spatial transformations. Thus, from a small set of cases, we easily generate a larger set.

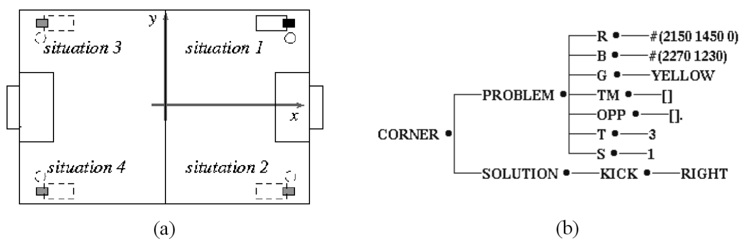


**Fig. 2.** (a) Situation 1 corresponds to the original description of the case. While situation 2, 3 and 4 correspond to the symmetric descriptions. (b) Example of a case.

## 5 Retrieval Step

To retrieve a case we must define a similarity function that computes the similarity degree between the current problem $P_c = ((R_c, B_c, G_c, Opp_c, t_c, S_c), \langle\rangle)$ and the cases in the case base $C_i = ((R_i, B_i, G_i, Opp_i, t_i, S_i), A_i)$ in the interval [0..1] (with 0 meaning no similarity at all, and 1 meaning maximum similarity). Next we introduce the different similarity functions used to compare the features of a case. We first compute the similarities along each feature (assuming feature independence). Then we use a filtering mechanism based on these values to discard non-similar cases and finally, we use an aggregation function to compute the overall similarity obtaining a set of similar cases (if any).

### 5.1 Similarity functions

We next define two types of similarity functions based on the features' types described in Section 4:

**Environment-based features** We use a 2D Gaussian function to compute the degree of similarity between two points, $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in a 2D space. Unidimensional Gaussian functions are defined by two parameters: one represents the reference value $x_r$ with respect to which we compare any other value $x$, and the other, the maximum distance $\tau$ allowed between two values to consider to be similar. Hence, low values for $\tau$ model very restrictive similarities, and high values, very tolerant similarities. As we work on a 2D plane, to define the Gaussian function we have to consider four parameters instead of two: $x_r, y_r, \tau_x$ and $\tau_y$:

$$G(x, y) = Ae^{-(\frac{(x-x_r)^2}{2\tau_x^2} + \frac{(y-y_r)^2}{2\tau_y^2})}$$

where $x_r, y_r$ are the reference values, $\tau_x, \tau_y$, the maximum distance for each axis and $A$ is the maximum value of $G(x, y)$. In our case, since we model the similarities in the interval [0..1], $A = 1$. Figure 3 shows a 2D Gaussian.

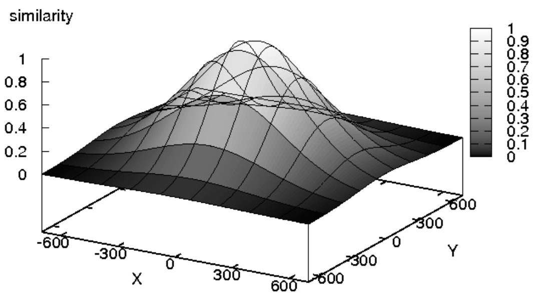


**Fig. 3.** 2D Gaussian function with $\tau_x = 300$ and $\tau_y = 250$.

We define the similarity function for two points as:

$$sim(x_1, y_1, x_2, y_2) = e^{-(\frac{(x_1-x_2)^2}{2\tau_x^2} + \frac{(y_1-y_2)^2}{2\tau_y^2})}$$

where the point $(x_1, y_1)$ refers to either the robots' or the ball's position in the problem and $(x_2, y_2)$ refers to the positions in the case. We do not use the heading of the robots to compute the similarity value, but for the reuse step.

Regarding the defending goal feature we define a simple binary function:

$$sim(G_1, G2) = \begin{cases} 1 & \text{if } G_1 = G_2 \\ 0 & \text{if } G_1 \neq G_2 \end{cases}$$

where $G_1$ is the defending goal in the problem and $G_2$, the one described in the case.

**Game-based features** We are interested in defining a function that combines time and score since they are extremely related. As time $t$ passes, depending on the score of the game, we expect a more offensive or defensive behavior. We consider as critical situations those where the scoring difference $S$ is minimum, i.e. when the chances for any of the two teams of winning or losing the game are still high, and thus the strategy (or behavior) of the team might be decisive. We model the strategy for a 20 minutes game as:

$$strat(t, S) = \begin{cases} \frac{t}{20(S-1)} & \text{if } S < 0 \\ \frac{t}{20} & \text{if } S = 0 \\ \frac{t}{20(S+1)} & \text{if } S > 0 \end{cases}$$

where $strat(t, S) \in [-1..1]$, with -1 meaning a very offensive strategy and 1 meaning a very defensive strategy.

Figure 4(a) depicts the behavior of the team at time $t$. Positive and negative scoring differences mean that the team is winning or losing respectively. The higher the absolute value of $S$ is, the lower the opportunity of changing the current score and the behavior of the team. For extreme values of $S$ (in the interval $[-10..10]$) the outcome of the function approaches zero. Otherwise, the function value indicates the degree of intensity, either for a defensive or an offensive behavior. As time passes, the intensity increases until reaching maximum values of 1 and -1, (defensive and offensive, respectively). Figure 4(b) shows the behavior of the function combining both variables.

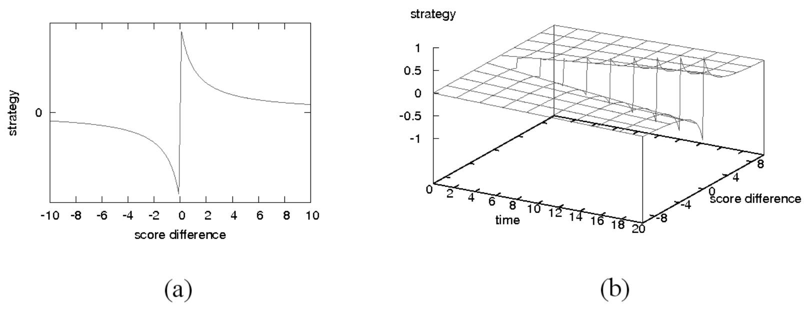


**Fig. 4.** (a) Strategy function for time $t = 5$. (b) Strategy function over time.

We define the similarity function for time and score as:

$$sim_{tS}(t_1, S_1, t_2, S_2) = 1 - |strat(t_1, S_1) - strat(t_2, S_2)|$$

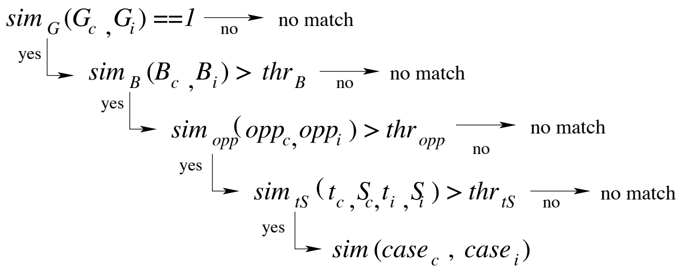
where $t_1$ and $S_1$ corresponds to the time and scoring features in the problem and $t_2$ and $S_1$, the features in the case.

## 5.2 Retrieving a case

Case retrieval is in general driven by the similarity metric between the new problem and the saved cases. We introduce a novel method to base the selection of the case to retrieve. We evaluate similarity along two important metrics: the similarity between the problem and the case, and the cost of adapting the problem to the case. Before explaining in more detail these metrics we first define two types of features: *controllable* indices and *non-controllable* indices. The former ones refer to the robot's and teammates' positions (since they can move to more appropriate positions), while the latter refers to the ball's and opponents' position, the defending goal, time and score (which we cannot directly modify).

The idea of separating the features into controllable and non-controllable is that a case can be retrieved if we can modify part of the current problem description in order to adapt it to the description of the case. Given the domain we are working on, the modification of the controllable features leads to a planning process where the system has to define how to reach the positions (or adapted positions as detailed in Section 6) of the robot and the teammates indicated in the retrieved case in order to reuse its solution.

**Similarity value** We compute the similarity between the current problem $P_c$ and a case $C_i$ using the non-controllable features. For this purpose, we filter the case based on the individual features similarities (Figure 5). If the similarities are all above the given thresholds, we then compute the overall similarity value between the case and the problem. Otherwise, we consider that the problem does not match the case. We discuss the values of these thresholds in Section 7.



$sim_G(G_c, G_i) == 1$
no
no match
yes
$sim_B(B_c, B_i) > thr_B$
no
no match
yes
$sim_{opp}(opp_c, opp_i) > thr_{opp}$
no
no match
yes
$sim_{tS}(t_c, S_c, t_i, S_i) > thr_{tS}$
no
no match
yes
$sim(case_c, case_i)$

**Fig. 5.** Filtering mechanism to compute the similarity between cases. The subindex $c$ refers to the current problem, and $i$, to a case in the case base.

In order to compute the opponents' similarity value we first must determine the correspondence between the opponents of the problem and the case, i.e. which opponent $opp_i$ from the problem description corresponds to which opponent $opp_j$ in the case description. For this purpose, we use a *Branch&Bound* search algorithm in binary tree. Each node of the tree represents either the fact of considering a match between the pair $(opp_i, opp_j)$, or the fact of not considering the match between this pair. As soon as the algorithm finds the optimal correspondence, we obtain the similarity value for each pair of opponents using the Gaussian function.

Finally, we compute the overall similarity $sim$ between the current problem and the case:

$$sim = f(sim_B, sim_{tS}, sim_{Opp_1}, \ldots, sim_{Opp_n})$$

where $n$ is the number of opponents in the case, and each argument of $f$ corresponds to the similarity value obtained for each feature. In Section 7 we discuss the most appropriate aggregation function $f$.

**Cost value** This measure defines the cost of modifying the controllable features of the problem $P_c$ to match the case $C_i$. We represent the cost of adapting the problem to a case as the maximum Euclidean distance *dist* between the players' positions in the current problem and the adapted positions in the case (after obtaining the correspondence between the players using the same method as for the opponents):

$$cost(P_c, C_i) = \max_{j \in \{R\} \cup Tm} \{dist(pos_j, pos'_j)\}$$

where $R$ corresponds to the robot, $Tm = \{tm_1, tm_2, tm_3\}$, to the teammates, $pos_j$ represents the position of $j$ in the problem description and $pos'_j$, the position of $j$ in the case description.

After computing the similarities between the problem and the cases, we obtain a list of potential cases from where we must select one for the reuse step. We consider a compromise between the similarity degree between the problem and the case and the cost of adapting the problem to the case. The properties for the best choice are to have a very similar case and to apply little adaptations to the problem to reuse the solution of the case, while the worst choice would be low similarity and high cost (the opposite situation). But we also have to avoid those situations where even though the similarity is high, the problem also needs a big adaptation (high cost) before reusing the selected case.

We then select the most similar case from the list of cases with cost lower than a threshold $thr_{cost}$:

$$C_r = \arg\max\{sim(P_c, C_i) \mid cost(P, C_i) < thr_{cost}\}, \quad \forall C_i \in \text{LS}$$

where LS is a list of cases with similarity over 0.4 and $C_r$ is the case retrieved.

## 6 Case Reuse

After selecting the best case, the next step is to reuse its solution. Before executing the actions indicated in the case, we first adapt the current problem to the description of the case. To this end we modify the controllable features (robot and teammates) to those positions where the relation between the features is the same as the one described in the case. We take the ball as the reference point in the field. From the case retrieved we obtain the relative positions of the players with respect to the ball. Hence, the adapted positions of the players for the current problem are the transformations of these relative coordinates to global coordinates, having the current position of the ball as the new reference point.

Figure 6 shows an example. The relative position of the robot with respect to the ball ($B_i = (750, 300)$) in the case retrieved is $R_i^r = (-300, 0)$. Thus, the robot's adapted global position in the current problem is $R_c = (350, 100)$ since the ball's position is $B_c = (650, 100)$. Briefly, the adaptation of the problem description is based on positioning the controllable features with respect to the ball's position, instead of maintaining the original positions indicated in the case. Once we compute these new locations, the robot retrieving the case informs the rest of the teammates about the positions they should take.

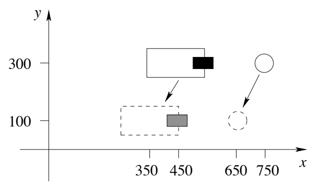


**Fig. 6.** The case description depicted in solid lines, and the problem description, in dashed lines. Adapting the position of the robot with respect to the ball's position described in the problem.

## 7 Empirical Evaluation

We discuss the different values for the thresholds and the aggregation function we have introduced in Section 5.2 and the first results of the system.

**Environment-based features** We have used a Gaussian to model the similarity function for this type of features. As we already mentioned, the function has two parameters, $\tau_x, \tau_y$, which are used to model the maximum distance between two points that we consider to be similar. These parameters define an ellipse (the projection of the Gaussian in the plane XY) with radius $\tau_x$ and $\tau_y$. All points contained in this ellipse have a $G(x, y) > 0.6$. Thus, we use this value as the threshold for the ball, $thr_B$, and opponents similarity, $thr_{opp}$. To set the $\tau$ values for the ball, we empirically observed that the maximum distance we consider the ball's position is similar to a reference point is 30cm. for the x axis, and 25cm. for the y axis (since the field has a rectangular shape). Thus, $\tau_x = 300$ and $\tau_y = 250$. Regarding the opponents' we consider a more flatter function because the imprecision of their positions is higher than the one for the ball. We then fix both $\tau_x$ and $\tau_y$ to 350.

**Game-based features.** We are specially interested in distinguishing between those situations that take place at the end of the game with score difference close to 0 from those that happen at the beginning of the game, since the strategy can be very different in each of these situations. After analyzing the values obtained by the strategy function described in Section 5.1, we observed that comparing two situations, fixing one to $t_1 = 0$ and $S_1 = 0$ and varying the other one through all the possible values, the following situations occur:

- first half of the game and no matter which score:

$$t_2 \in [0..10) \wedge S_2 \in [-10..10], sim_{tS}(t_1, S_2, t_2, S_2) > 0.7$$

- first part of the second half and equal scoring:

$$t_2 \in [10..14] \wedge S_2 = 0, sim_{tS}(t_1, S_2, t_2, S_2) < 0.7$$

– second part of the second half and 1 goal difference:

$$t_2 \in [15..18] \wedge S_2 \in [-1..1], sim_{tS}(t_1, S_2, t_2, S_2) < 0.7$$

– ending game and 2 goals difference:

$$t_2 \in [19..20] \wedge S_2 \in [-2..2], sim_{tS}(t_1, S_2, t_2, S_2) < 0.7$$

As we can see, fixing the threshold $thr_{tS}$ to 0.7 allows us to separate the situations previously mentioned.

**Aggregation function.** We tested four different functions: the mean, the weighted mean, the minimum and the harmonic mean. The minimum function results in a very restrictive aggregation function since the overall outcome is only based on the lowest value. Hence, lower values penalize high values rapidly. Regarding the harmonic mean, for similar values, its behavior is closer to the mean function. While for disparate values, the lower values are highly considered and the outcome decreases (although not as much as with the minimum function) as more lower values are taken into account. On the contrary, the mean function rapidly increases the outcome for high values, and does not give enough importance to low values. Finally, the weighted mean does not make difference between low and high values either, since the importance of each value is given by their weights. If a low value has a low weight and the rest of the values are all high, the outcome is slightly affected and results high anyway.

We are interested in obtaining an aggregation function that considers all values as much as possible but highlighting the lower ones. This is an important property as the values we are considering are similarity values. Hence, if one of the features has a low similarity, the overall similarity has to reflect this fact decreasing its value. Therefore, we use the harmonic mean as the aggregation function $f$:

$$f(x_1, ..., x_n) = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

where $x_i$ corresponds to the individual similarity values of the features.

**Cost threshold.** We consider worth adapting a problem to a case if the distances the robots have to travel from their original positions to the adapted ones are short enough so the environment changes as little as possible during this time. After observing the robots movements, we fixed the maximum distance to translate them to 1m. Their current average velocity is 350 mm per second. Hence, walking for 1m. takes around 2.8 seconds. Even though for now we are fixing this value to test the current system, we have to take into account that the threshold also depends on the opponents we are playing with. The faster they are, the lower the threshold should be.

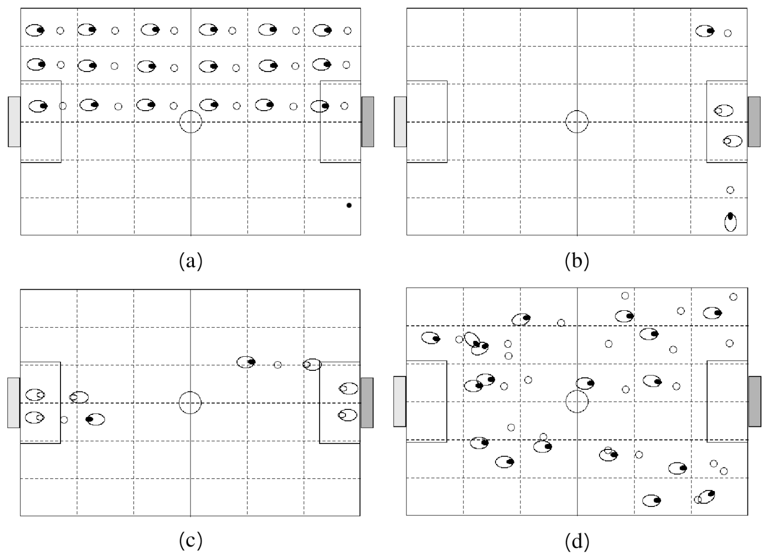


**Fig. 7.** (a) shows simple cases which allow the robot (depicted with filled head) to kick the ball towards the goal at every point of the field, (b) and (c) correspond to more complex situations where we have included one or two opponents (depicted with non-filled heads) and our robot attacking either from the front or the corners, and (d) shows some of the problems we used to test the system so far.

**Experiments** We manually defined 90 cases with one player, i.e. no teammates so far, varying the number of opponents (from 0 to 2), the time and the score difference. We also tested 50 problems created randomly to verify if the correct cases were retrieved. We indeed obtained always the right ones indicating the adapted position the robot should take and the actions to perform from that point on. Figure 7 depicts a set of the cases and problems created.

## 8 Extending the case definition

As previously mentioned, the solution of a case is a sequence of actions. So far we have been comparing snapshots of the current game with cases that describe the initial state of a game play. We believe that it would be also interesting to consider parts of a game play (the solution of a case) as part of the problem description of a case. The solution represents the (discrete) trajectory performed by the robots with their related actions. Thus, instead of comparing the current problem with the initial state of the case, we could compare it with the execution of the solution and reuse the solution from the closest point. This way, we can also avoid useless movements (e.g. going backwards to reach the initial position and then going forward again executing the solution's actions).

To this end, cases should have a more complex structure. We should define them by means of a graph structure or a sequence of nodes, where each node represents a situation $S_i$ (description of the environment at time $t$) and arcs represent the associated actions to go from one node to the other. Then the retrieval step would have to consider each node $S_i$ as a potential similar case to solve the new problem.

Given the problem $P$ and the case $C$ depicted in Figure 8, instead of positioning the robot in the initial state of the case ($S1$), we could move it to the adapted position indicated in $S2$ and then continue reusing the solution from this point.

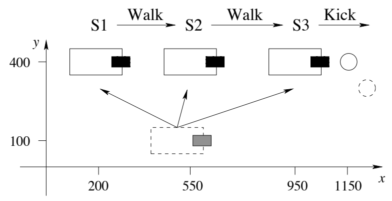


**Fig. 8.** Case description (solid lines) and problem description (dashed lines).

## 9 Conclusion and Future Work

We have presented the initial steps towards a Case-Based Reasoning system for deciding which actions a robot should execute in the robot soccer domain. More precisely, we have focused our work on the retrieval and reusing steps of the system. While we contribute concretely to robot soccer, several of the features of the approach are applicable to general game-based adversarial environments.

We have defined the concept of *case* as well as the features that describe the state of a game, dividing them in two types: the *environment-based features* and the *game-based features*. We have discussed the similarity functions for the different features and we have tested different aggregation functions to compute the overall similarity. We have introduced the a separation between the *controllable* and the *non-controllable* case indices to compute two metrics: the similarity and the cost. We select the retrieved case based on a compromise between the similarity and the cost of adapting the current problem to a case. Regarding the case reuse, we have described the adaptation of the description of the problem to the case retrieved and the reusing process of the solution. To test these first steps, we have designed a simulation interface to easily modify the different functions and parameters described.

As future work, we will continue on finishing the extension of the case description we have proposed in Section 8. After further testing the proposed approach in simulation, we will move our case-based approach to real robots.

## References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.
2. RoboCup Technical Committee. *Sony Four Legged Robot Football League Rule Book*, Dec 2004.
3. T. Gabel and M. Veloso. Selecting heterogeneous team players by case-based reasoning: A case study in robotic soccer simulation. Technical report CMU-CS-01-165, Carnegie Mellon University, 2001.
4. K.Z. Haigh and J.R. Shewchuk. Geometric similarity metrics for case-based reasoning. In *In CBR: Working Notes from the AAAI-94 Workshop*, pages 182–187, 1994.
5. A. Karol, B. Nebel, C. Stanton, and M. Williams. Case Based Game Play in the RoboCup Four-Legged League Part I The Theoretical Model. In *RoboCup*, pages 739–747, 2003.
6. M. Kruusmaa. Global navigation in dynamic environments using case-based reasoning. *Auton. Robots*, 14(1):71–91, 2003.
7. A. Lattner, A. Miene, U. Visser, and O.Herzog. Sequential Pattern Mining for Situation and Behavior Prediction in Simulated Robotic Soccer. In *9th RoboCup International Symposium*, 2005.
8. M. Likhachev and R. Arkin. Spatio-temporal case-based reasoning for behavioral selection. In *ICRA*, pages 1627–1634, 2001.
9. A. Ram and J. C. Santamaria. Continuous case-based reasoning. *Artificial Intelligence*, 90(1-2):25–77, 1997.
10. M. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, and R. Ehrmann. Karlsruhe brainstormers — A reinforcement learning approach to robotic soccer. *Lecture Notes in Computer Science*, 2019, 2001.
11. R. Ros, R. López de Màntaras C. Sierra, and J.L. Arcos. A CBR system for autonomous robot navigation. *Frontiers in Artificial Intelligence and Applications (Proceedings of CCIA'05)*, 131:299–306, 2005.
12. A. Sarje, A. Chawre, and S. Nair. Reinforcement Learning of Player Agents in RoboCup Soccer Simulation. In *Fourth International Conference on Hybrid Intelligent Systems*, pages 480–481, 2004.
13. J. Wendler, S. Brüggert, H. Burkhard, and H. Myritz. Fault-tolerant self localization by case-based reasoning. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 259–268, London, UK, 2001. Springer-Verlag.
14. J. Wendler and M. Lenz. CBR for Dynamic Situation Assessment in an Agent-Oriented Setting. In *Proc. AAAI-98 Workshop on CBR Integrations*, 1998.