Acquiring a Robust Case Base for the Robot Soccer Domain

Raquel Ros, Josep Lluís Arcos
IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Barcelona, Spain *
{ros,arcos}@iiia.csic.es

Abstract

This paper presents a mechanism for acquiring a case base for a CBR system that has to deal with a limited perception of the environment. The construction of case bases in these domains is very complex and requires mechanisms for autonomously adjusting the scope of the existing cases and for acquiring new cases. The work presented in this paper addresses these two goals: to find out the "right" scope of existing cases and to introduce new cases when no appropriate solution is found. We have tested the mechanism in the robot soccer domain performing experiments, both under simulation and with real robots.

1 Introduction

Working with robots that interact with their environment is a complex task due to the degree of uncertainty in the robot's perception of the world. But even if we achieve a very accurate perception mechanism, and as consequence, have a very low degree of uncertainty, we still have to consider that the environment is not fully controllable and unpredictable situations can occur. For these reasons, the reasoning system has to introduce some mechanism that, on the one hand, allows to adapt the a priori knowledge given by an expert to the real perception of the robot and, on the other hand, automatically incorporates new knowledge when an unexpected situation occurs.

Researchers address the case base acquisition in different ways based on their domains. In the textual case-based reasoning field, cases are extracted from textual sources as for example e-mails, manuals and scientific papers [Minor and Biermann, 2005], or medical records [Abidi and Manickam, 2002]. In robotics, log files created during the execution of the tasks are used as inputs for case generation [Gabel and Veloso, 2001]. Interactive approaches have been studied to complete the missing knowledge of the system with a user through concept mapping [Leake and Wilson, 1999] or even

*Partial funding by the Spanish Ministry of Education and Science projects QUALNAVEX (DPI2003-05193-C02-02) and MID-CBR (TIN2006-15140-C03-01). Raquel Ros holds a scholarship from the Generalitat de Catalunya Government.

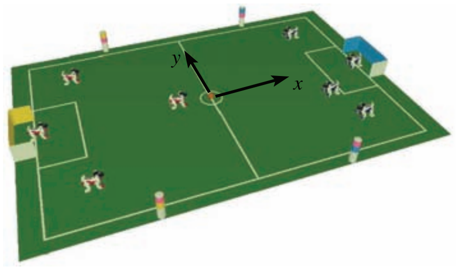


Figure 1: Snapshot of the Four-Legged League (image extracted from the Official Robocup Rule Book).

acquiring case adaptation knowledge combining rule-based methods with user guidance [Leake *et al.*, 1996]. In the AQUA system, [Ram, 1993] presents an approach for incremental learning based on the revision of existing cases and the incorporation of new ones when no solution was found. His main challenges were: (i) novel situations; (ii) mis-indexed cases; and (iii) incorrect or incomplete cases. In our approach, we attempt to cover (i) and (iii) in our domain.

This paper is the continuation of previous work [Ros *et al.*, 2006], where we designed a Case-Based Reasoning system for action selection in the robot soccer domain. Before introducing the problem we address, we briefly describe the domain and the most relevant features of the system.

We focus our work on the Four-Legged League of the RoboCup Soccer Competition. In this 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 is 6 m long and 4 m 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. For details on the official rules refer to the Official Robocup Rule Book.

System Description

The goal of the CBR system is to determine the actions the robots should execute given a state of the game (a snapshot of the game at time t). Instead of considering single actions for each state, we define sequences of actions which we call game plays. Some examples are *get near the ball and shoot* or *get the ball, grab it, turn and shoot*. Hence, a case consists of the description of the environment (problem description) and the game play the robots performed for that state (solution description). The problem description features are: robots positions, ball position, opponents positions, defending goal, time of the game and score difference. The solution description is defined as an ordered list of actions.

Henceforward in the paper we will focus on a simplified version of the system: We consider only one robot, the ball and the defending goal as the description of the problem (although the ideas presented are extendable to the other features described above). The solution description remains the same:

$$case = ((R, B, G), A)$$

where $R = (x, y, \theta)$, $B = (x, y)$, $G = \{cyan, yellow\}$, and $A = [a_1, a_2, ..., a_n]$ (θ corresponds to the robot's angle with respect to the x axis of the field).

In order to retrieve the most similar case, we model the similarity function[1] based on the ball position with a 2D Gaussian function. We consider two points in the Cartesian plane are similar if their distance is below a given threshold. Using a Gaussian allows us to model the maximum distances in the x and y axis we consider two values to be similar, as well as different degrees of similarities based on the proportional distances of the points. It is defined as follows:

$$G(\triangle_x, \triangle_y) = e^{-(\frac{(\triangle_x)^2}{2\tau_x^2} + \frac{(\triangle_y)^2}{2\tau_y^2})}$$

where $\triangle_x, \triangle_y$ are the distances between the points in the x and y axis respectively, and τ_x, τ_y, the maximum distances for each axis. These parameters represent the scope of the case, i.e. the region where the points are considered similar (represented by an ellipse on a 2D plane). Each case may have different region sizes, hence, for each case we also store this additional information as part of the system knowledge.

When a new problem is presented to the system, we first filter the cases based on the defending goal G: the problem and the case must have the same defending goal. Otherwise, they cannot be considered similar at all. Next, we compute the similarity between the position of the balls. If the similarity exceeds a given threshold then we consider the case as a potential solution to the problem. We select the case with higher similarity degree for the reusing step.

The problem we address in this paper is how to ensure the robustness of the system given the high uncertainty of the robot's perception. In a CBR system, the correctness of the case base is one of the main issues that determines the accuracy of the system performance, since it represents its knowledge. Given a new problem, its solution is obtained by comparing the description of the new problem with the cases in the case base. In our approach the position of the ball in the field (perception) and the parameters τ_x and τ_y used in the similarity function (knowledge) are crucial for the correct performance of the retrieval step. Otherwise, wrong cases might be returned, or even no case at all. In both situations, it could be either because the ball is not correctly positioned or because the parameters have high values (modelling large regions) or low values (modelling small regions).

[1]In the complete system we take into account other features to compute the similarity between cases. The robot position is not included in the retrieval step. We will not go into details, since it is not relevant for the work we present here.

We are also interested in creating case bases with "essential" cases, meaning that we expect to include cases that are general enough to cover basic situations, but also include cases that represent specific situations: the first ones are represented with larger scopes, while the second ones, with smaller scopes. Having these two types of cases allows us to reduce the number of cases to the most relevant ones, which is a desirable property in any real-time response domain.

Since we cannot improve the robot's perception (because of hardware limitations and the need of real time response) and finding by hand the right parameters for the reasoning module is very hard, we propose: first, to include in the reasoning model a mechanism to automatically compute the scope of existing cases based on the actual perception of the robot; and second, to introduce an additional mechanism in combination with the former to include new cases in the system if no case is retrieved. With the former we ensure that the regions the cases cover are adapted to the robots believes of the world, and therefore, they will respond according to its perception. And with the latter, the robot has the ability to be "creative" and to act with a new behavior when the system does not return any possible solution.

Briefly, our approach is focused on creating an initial case base with partial information so the system itself can complete it either by modifying the scope of the cases, in order to cover the problem space with the minimum number of cases, or introducing new cases when needed. Both processes are guided by a human trainer.

2 Learning the Scope of Existing Cases

The initial case base of our system is manually created and represents the a priori expert's knowledge. The idea is to provide the system a set of cases which must be adapted to the robot's actual perception. The expert knows several generic situations (prototypical cases) and their corresponding solutions, but cannot predict the real scope of the cases from the robot's point of view. Thus, we propose to create an initial case base composed of prototypical cases with default scopes, τ_x^0 and τ_y^0, and then let the robot learn them. To this end, the expert should tell the robot if the case retrieved at each time is correct or not. We refer to this process as the training step.

As mentioned in the previous section, we refer to the case's scope as the region of the field where that case should be retrieved. These regions are modelled as ellipses, which correspond to Gaussians' projections on a plane (from now on, we will refer directly to the *ellipse*). In order to learn the scope of the cases, we propose to modify the size of these ellipses varying the Gaussian parameters τ_x and τ_y. To this end, we must define some policy to determine when and how these parameters should be adjusted.

2.1 When to adjust the values

The goal of increasing or decreasing the size of the ellipse is to reach the "ideal" region that a case should cover. The center of the ellipse represents the exact position of the ball on the field specified in that case. As we move towards the boundary

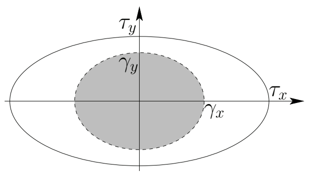


Figure 2: Example of a *security region* (gray region) and a *risk region* (white region) defined by $\gamma_x = 0.5$ and $\gamma_y = 0.75$.

of the ellipse, the uncertainty about whether the points belong to the scope of the case increases. We call the set of points next to the center of the ellipse the *security region*, and those near the boundary of the ellipse the *risk region*. We define γ_x and γ_y as the relative size of the *security region* with respect to the size of the ellipse (each value corresponds to a radius percentage). Figure 2 shows an example of these regions.

When the system proposes a solution for a new problem, we use the expert's feedback for tuning the parameters of the retrieved case. If the proposed solution succeeded, the scope of the case is increased. Otherwise, it is decreased.

We will first focus on the increasing process. If the problem is located inside the *security region* (the position of the ball is in this region) the system cannot introduce new knowledge to the case. Its current information is robust enough to determine that the problem corresponds to the scope of that case. On the contrary, if the problem is within the *risk region* the system can confirm the current scope of the case increasing the size of the ellipse. Expanding the ellipse results in expanding the *security region* as well.

Problems are incorrectly solved using a case due to scope overestimation. Hence, we have to reduce the size of the ellipse. If the ball is inside the *security region*, we do not decrease the parameters since it corresponds to a robust region. If the problem is within this region and the feedback is negative, we assume that the error is caused by something else (wrong localization) and not because of a wrong knowledge of the system. As an illustration, imagine the following situation: the robot is not well localized and as a consequence, it perceives the position of the ball incorrectly. It could happen that it retrieves the right case given its own perception. But from the external observer perception, the case used to solve that problem is not the right one. Therefore, the feedback given to the robot is negative. If the system reduces the ellipse, it could radically reduce the scope of the case, not because it was overestimated, but because of the high imprecision. However, when the problem is inside the *risk region*, the system does reduce the scope of the case, since the scope overestimation might be the cause of the negative feedback.

In summary, the system enlarges or reduces the scope of a case, i.e. modifies its knowledge, when the problem presented is correctly or incorrectly solved and it is within the case's *risk region*.

2.2 How to adjust the values

The first problem is to determine the increasing values for each parameter (τ_x, τ_y), i.e. how much should the system increase the size of the ellipse with respect to each axis. We define δ_x and δ_y as the increasing value, and $\hat{\delta}_x$ and $\hat{\delta}_y$ as the maximum increasing value for each axis. We propose three increasing policies (we only show the equations for axis x; the same equations are used for axis y):

- *fixed*: the increasing amount is a fixed value. Thus, we define a step function:

$$\delta_x = \begin{cases} \hat{\delta}_x & \text{if } \triangle_x \geq \gamma_x \tau_x \\ 0 & \text{otherwise} \end{cases}$$

- *linear*: we compute the increasing value based on a linear function:

$$\delta_x = \begin{cases} \frac{\triangle_x - \gamma_x \tau_x}{\tau_x - \gamma_x \tau_x} \cdot \hat{\delta}_x & \text{if } \triangle_x \geq \gamma_x \tau_x \\ 0 & \text{otherwise} \end{cases}$$

- *polynomial*: we compute the increasing value based on a polynomial function:

$$\delta_x = \begin{cases} \frac{(\triangle_x - \gamma_x \tau_x)^5}{(\tau_x - \gamma_x \tau_x)^5} \cdot \hat{\delta}_x & \text{if } \triangle_x \geq \gamma_x \tau_x \\ 0 & \text{otherwise} \end{cases}$$

After computing the increasing values, we update τ_x and τ_y adding the computed δ_x and δ_y respectively.

The motivation for decreasing the parameters is to reduce the ellipse size so the new problem solved is not considered inside the region anymore. To this end, we equal τ_x and τ_y to the values in the problem, only if they are higher than the radius of the *security region*:

$$\tau_x^t = \begin{cases} \triangle_x & \text{if } \triangle_x \geq \gamma_x \tau_x \\ \tau_x^{t-1} & \text{otherwise} \end{cases}$$

We update τ_y in the same way.

Updating both values separately and only when the problem is within the *risk region* prevents from radically reducing the scope of the case. Below we describe a simple example to illustrate the approach.

2.3 Example

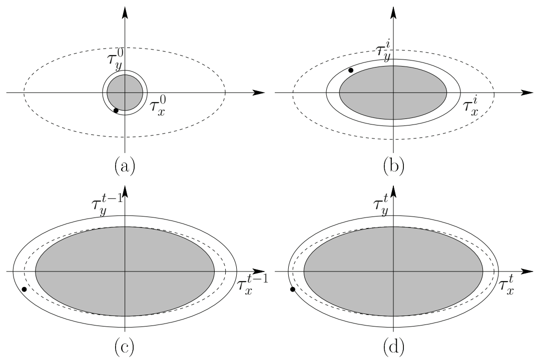


Figure 3: Case scope evolution. $\gamma_x = \gamma_y = 0.8$

Figure 3 depicts four steps of the training process. The gray region represents the *security region*, while the dashed ellipse corresponds to the "ideal" scope of the case (defined by the human trainer) we attempt to reach. Any problem located within this ideal area produces a positive feedback by the expert. The black dot represents a new solved problem (ball position with respect to the case). Figure 3 (a) shows the initial stage at time 0, where the scope of the case is minimum

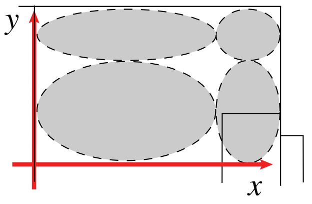


Figure 4: "Ideal" case base based on the expert knowledge.

(τ_x^0, τ_y^0). Since the new solved problem is within the *risk region* and the feedback is positive, we proceed to enlarge the size of the ellipse using one of the policies defined.

At time i, Figure 3 (b), we can observe that the ellipse has increased, but still has not reached the expected size. Hence, we continue to enlarge the scope by solving new problems as long as the expert feedback is still positive.

Figure 3 (c), time $t-1$, depicts a situation where the ellipse generated is bigger than the expected size. From now on, the feedback may be positive or negative. If a new problem is within the *risk region* and the feedback is positive, then we would proceed to increase the ellipse. But, if the feedback is negative, then the decreasing process is used to reduce the ellipse. The figure shows an example of this situation. As we can see, the new problem is located in the *risk region*, but out of the ideal scope. Thus, the current scope is reduced, but only by updating τ_x since $\triangle_y < \gamma_y \tau_y$.

Figure 3 (d) shows the updated scope, where the problem remains outside the scope of the case. As more problems are solved, the scope of the case will converge to the ideal scope.

In conclusion, we distinguish two phases in the training process: growing the scope of the case and converging to the ideal scope. During the first phase, the feedback is always positive and the scope is always being expanded. The second phase occurs once the expected scope is exceeded. Then, the feedback could either be positive or negative. The goal of the positive feedback is to enlarge the scope, while the goal of negative feedback is to converge to the ideal scope the human trainer expects.

3 Introducing New Cases in the System

Finding out all possible situations a robot can encounter during its performance is unfeasible. Even more so in a domain with high uncertainty, such as the domain we are working with. Because of the domain (a real time game), we cannot afford the robot to stop during the game just because it does not "know" what to do in that situation. Somehow, the robot must always execute an action at every time step of the game.

After the training step, the knowledge of the system might present some "gaps", i.e. the scope of the cases may not cover the whole field. Of course, this depends on the number of cases used during the training. But as we mentioned, on the one hand, the expert cannot define all possible cases. And on the other hand, we focus our approach on initially defining a set of generic situations, allowing the robot to create new ones based on its own experience afterwards.

Figure 4 shows an example of a hand coded case base (only 4 cases). For simplicity, we only show a quarter of the field (the positive-positive quadrant). Each ellipse represents a predefined case (given knowledge). As we can see, several gaps appear between them (white regions). They represent the regions where the robot will have to acquire new information to increase its knowledge.

A new case is created using the description of the environment (problem description), and a generated game play (solution of the new case). To create a game play, we provide the system a set of possible actions the robot can perform. The combination of these actions correspond to potential game plays. Given a new problem to solve, if it does not retrieve any case (either due to imprecision problems or because the problem is actually in a gap) the system generates a random game play[2]. The robot executes the suggested action and the expert evaluates the correctness of the solution proposed. Only if it succeeds, the new case is created.

When a new case is inserted into the system, it is created with a minimum scope (a small ellipse). From that moment on, the evolution of the new case depends on how often the robot reuses it, enlarging or reducing its scope using the mechanism presented previously. The idea is that at the beginning, the new case could seem to be a good solution for that concrete situation, but its actual effectiveness has to be evaluated when the robot reuses it. As time passes, if the scope of the case does not increase, and instead, it is reduced, we can deduce that the case is not useful for the robot's performance. On the contrary, if its scope increases, or at least, it remains stable, then we consider that the case contributes to the system's knowledge.

4 Experimentation

This section describes the experiments performed in order to test the approach introduced in the paper. We divide the experimentation in two stages: simulation and real robots.

4.1 Simulation

The goal of this first phase is to examine the behavior of the policies using different values for the parameters presented in Section 2.1. Since we had to test different combinations of values, simulation was the fastest way to obtain orientative results. The most relevant were selected for the experimentation with real robots.

We based the experiments on a single case to observe how the different values of the variables affect the evolution of its scope, i.e. the resulting size of the ellipse for the case. The initial case was defined with a small scope, $\tau_x = 100$ and $\tau_y = 100$. The expected outcome was $\tau_x = 450$ and $\tau_y = 250$. We randomly created 5000 problems. Every time the case was retrieved, we used the different policies to modify the scope of the case. The experiment was repeated combining the following values:

- *security region* size (expressed as percentage): $\gamma_x = \{0.5, 0.6, 0.7, 0.8, 0.9\}$
- maximum increasing value (expressed in mm): $\hat{\delta}_x = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$

[2]In this paper, the number of available actions is low. Hence, we considered that generating random game plays is feasible. We believe that for more complex situations other mechanisms should be used since the random generation would not be scalable.

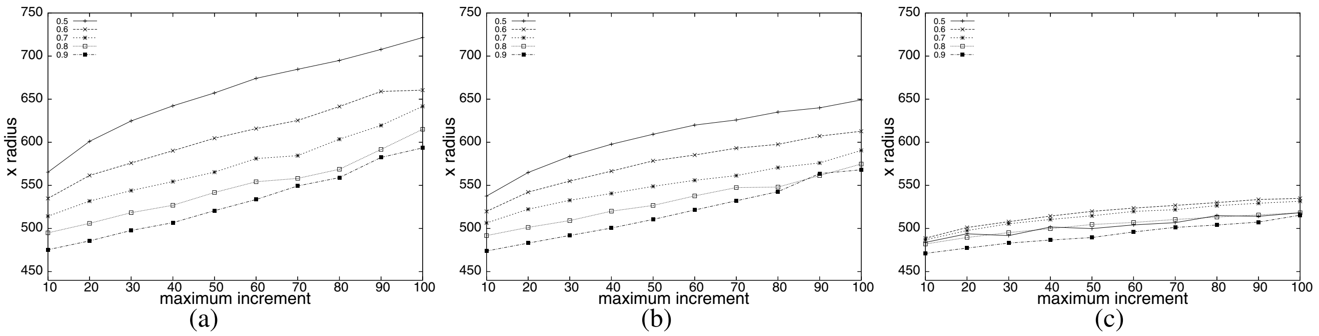


Figure 5: Resulting τ_x using the policies: (a) Fixed policy. (b) Linear policy. (c) Polynomial policy.

The same values were used for γ_y and $\hat{\delta}_y$.

For each combination of values, we ran 10 experiments. Figure 5 shows the average of the obtained results. On the one hand, $\hat{\delta}_x$ and $\hat{\delta}_y$ define how much the ellipse may increase at each time. Hence, the higher their values, the bigger the resulting scope of the case. On the other hand, γ_x and γ_y determine the size of the *security region* and the *risk region* (low values represent small *security regions* and large *risk regions*). The *risk region* determines when the scope of the case has to be modified. As this region increases, there are more chances of modifying the scope as well. Thus, for all three policies, the curves tend to increase from left to right.

With respect to the policies, the *fixed* policy obtains the highest τ_x and τ_y values, while the *polynomial* obtains the lowest ones. The former function has a more aggressive behavior, radically increasing the size of the ellipse. The latter function has a more conservative behavior, computing first small increments and then increasing as we reach the boundary of the ellipse. As a consequence, the *fixed* policy significantly varies between the different parameters, whereas the behavior of the *polynomial* policy remains more stable although the values of the parameters change. Regarding the *linear* policy, it has an intermediate behavior with respect to the other two (tending more to the *fixed* policy).

After the experimentation, we can confirm that a more conservative policy, low increasing values and small *risk regions* is the most appropriate combination to obtain the desired scope of the cases. The conclusion is obvious since we are establishing ideal conditions in order to gradually increase the scope of the cases. But two problems arise when extending the experiments to the real world: time and uncertainty. First, the number of iterations needed to reach the expected result is unfeasible when working with real robots; and second, a noise-free environment is only available under simulation. Although we have observed different behaviors in the graphics obtained when gradually modifying the parameters, these differences are not so obvious in a real environment because other issues modify the expected result. Therefore, the next stage is to experiment in the real world with the most relevant parameters (understanding relevant as the ones that show more contrasting behaviors) to determine the effectiveness of the approach presented.

4.2 Real robots

Three types of experiments were performed with real robots. The first one aims to find out the most appropriate parameters and policy to use in the system. The second one consists in evaluating the convergence of the cases in a given case base. Finally, the goal of the third one is to observe if the system is able to acquire new knowledge when no solution is found.

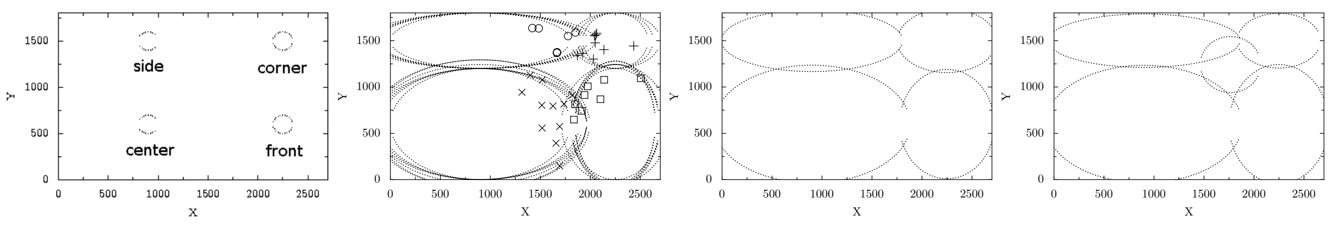
Testing the parameters and policies As we mentioned, we can divide the training process in two steps: growing the scope of the case and converging to the ideal scope. We are interested in rapidly enlarging the size of the ellipse until reaching the ideal one, and then opt for a more conservative behavior to adjust it. We switch from one strategy to the other when the size of the ellipse is decreased.

We can achieve this strategy either by combining the policies or by modifying the values of the parameters through the process. Regarding the policies, we have included two additional strategies: *fixed* or *linear* policy for the growing step, and the *polynomial* for the convergence step. With respect to the parameters, for the first step we define large *risk regions* and high increasing values, and the opposite for the second step.

The experimentation is similar to the simulation stage, where the experiments are based on a single case. The expected scope of the case is $\tau_x = 900$ and $\tau_y = 600$. We generated 100 new problems, manually positioning the ball in the field. Each experiment combined the five policies with three different sets of parameters: (i) $\gamma_x = 0.5$, $\hat{\delta_x} = 100$; (ii) $\gamma_x = 0.7$, $\hat{\delta_x} = 50$; (iii) $\gamma_x = 0.5$, $\hat{\delta_x} = 100$ and $\gamma_x = 0.7$, $\hat{\delta_x} = 50$ (the former are for the growing process, and the latter, for the convergence process). The same values were used for γ_y and $\hat{\delta}_y$. Each parameter varies separately depending on the τ altered (modifying τ_x does not imply modifying τ_y as well). We performed 10 trials per set.

Comparing the results with respect to the expected scope, we verify that: (i) the *fixed* and *linear* policies generate the highest τ_x values exceeding it; (ii) the *polynomial* policy does not even reach the ideal scope because of the low increasing speed; (iii) both *fixed-polynomial* and *linear-polynomial* strategies obtain the closest scopes to the expected ones, since they combine the advantages of both policies.

Regarding the values of the parameters, we confirmed the conclusions drawn from simulation: combining low increasing values and small *risk regions* ensures reaching the expected result. The problem once again is the number of steps for achieving this goal. Hence, combining the values of the parameters –first high increasing values and large *risk regions*



(a) (b) (c) (d)

Figure 6: Case base evolution: (a) Initial case base. (b) During training. (c) Resulting case base. (d) Acquiring a new case.

and then the opposite– results in a good alternative: we reach the ideal scope faster and then progressively adjust it.

Training the case base We created a simple case base of four cases (Figure 6 (a)):

- *center*: midfield. Action: get the ball and kick forward.
- *side*: left edge. Action: get the ball and kick right .
- *corner*: left corner. Action: grab the ball, turn until reaching 270 degrees and kick forward.
- *front*: between the *center* case and the goal. Action: grab the ball, face goal and kick forward.

All cases were initiated with the same scope ($\tau_x = \tau_y = 100$). We performed 25 trials (each with 50 random problems) for the training process. Figure 6 (b) shows the final steps of the process where the size of the ellipse is converging towards the final outcome. The solved problems are represented with crosses (*center* case), circles (*side* case), plus (*corner* case) and squares (*front* case). Finally, Figure 6 (c) shows the average of the 25 trials outcomes. As we can see, the initial case base successfully evolves into the expected case base that was designed (Figure 4).

Acquiring knowledge Once the case base is trained, the robot is ready to acquire new cases. The goal is to verify that the robot is able to fill in the gaps of the trained knowledge. We focused the experiment on learning a single case located between the four cases. The expected action was to get near the ball facing the goal and bump it (the intention is to bring the ball closer to the goal without a forward kick since it is too forceful and would push the ball out of the field).

We performed 20 trials, each composed of 50 random problems. Through all the trials the new case was created. Figure 6 (d) shows the scope (average of the 20 trials) of the new case after expanding it. As we can see, the gap is almost completely covered with the expected case.

5 Conclusions and Future Work

The aim of the this work is to obtain a robust case base (knowledge) from the robot's own observations. The scope of the cases are represented by projections of Gaussian functions on a plane (ellipses). We proposed a mechanism to adjust the size of these ellipses to their corresponding "ideal" sizes indicated by a trainer, but taking into account the robot's perception. This way we ensure that the errors in its perception are also taken into account when modelling its knowledge.

We also included an automatic mechanism to acquire new cases when no case is retrieved. In combination with the former process, the case base is not only more robust, but is enlarged as well with the most relevant cases.

Several experiments have been performed. A first stage to test the parameters and proposed policies, both under simulation and with real robots. And a second stage focused on evaluating the evolution of the case base and the incorporation of new cases in the system.

After verifying that the proposed approach works, we are ready to try more complex situations with the whole system. We also expect to implement a scalable game play generator, which will allow us to introduce more sophisticated solutions for new unresolved problems.

References

[Abidi and Manickam, 2002] S. Abidi and S. Manickam. Leveranging XML-based electronic medical records to extract experiential clinical knowledge. An automated approach to generate cases for medical case-based reasoning systems. In *Int J. Medical Informatics*, 2002.

[Gabel and Veloso, 2001] 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.

[Leake and Wilson, 1999] D. Leake and D. Wilson. Combining CBR with interactive knowledge acquisition, manipulation, and reuse. *LNCS*, 1650, 1999.

[Leake *et al.*, 1996] D. Leake, A. Kinley, and D. Wilson. Acquiring Case Adaptation Knowledge: A Hybrid Approach. In *AAAI/IAAI, Vol. 1*, 1996.

[Minor and Biermann, 2005] M. Minor and C. Biermann. Case acquisition and semantic cross-linking for case-based experience management systems. In *Int. Conf. on Information Reuse and Integration*, 2005.

[Ram, 1993] A. Ram. Indexing elaboration and refinement: Incremental learning of explanatory cases. *Machine Learning*, 10(3), 1993.

[Ros *et al.*, 2006] R. Ros, M. Veloso, R. López de Màntaras, C. Sierra, and J.L. Arcos. Retrieving and Reusing Game Plays for Robot Soccer. In *Proc. 8th European Conference on Case-Based Reasoning*, 2006.