An Online Coach for RoboCup

Luciano Cavalheiro da Silva[1], Daniela D. S. Bagatini[1,2], Luis Otávio Alvares[1]

[1]Universidade Federal do Rio Grande do Sul
Instituto de Informática
Av. Bento Gonçalves 9500, Porto Alegre, 91501970 – RS – Brazil
Fone: 55(51)33166168 Fax: 55(51)33167308
{lucc, bagatini, alvares}@inf.ufrgs.br

[2]Universidade de Santa Cruz do Sul
Departamento de Informática
Av. Independência 2293, Santa Cruz do Sul – RS – Brazil
Fone: 55(51)37177393

Abstract. The RoboCup, robot soccer world-wide cup, appeared with the objective to evaluate the progress of techniques used in the construction of groups of autonomous agents which collaborate for the solution of a common problem: to win a soccer game. RoboCup comprises three competition modalities: two that involve the construction of players in hardware (robots) and another one of simulation. Recently, the possibility of using an online coach agent during the games was added to the simulation modality. From the analysis of game data supplied by the simulator, such agent can, although it can not act directly on the ball like a player, infer tactical modifications in the behavior of the team and suggest them to its players. This paper presents a proposal of an online coach agent for RoboCup agent teams. A set of metrics, designed for team behavior evaluation, is then described, followed by a characterization of its use in the optimization of the agent team behavior. **Keywords**: artificial intelligence, multi-agent systems, Soccerserver, RoboCup.

1 Introduction

The study of multi-agent systems (MAS) is a wide developing field in Artificial Intelligence. In order to evaluate the progress and the innovations of techniques used in the construction of groups of collaborative autonomous agents, came out the RoboCup [1]: a robot soccer world-wide cup. RoboCup comprises three competition modalities: two that involve the construction of players in hardware (robots), specific for the tournament, and another of simulation, which is the main interest of this paper.

In the simulation modality, participants construct teams of autonomous agents in software, where each agent corresponds to a player, that are confronted with other teams with the aid of a simulator called Soccerserver [2]. The simulator is responsible for applying physical restrictions, like wind and ball friction onto the ground, and also

controls the energy that is expended by the players in response to their actions such as running and kicking. Besides, a Soccerserver module is in charge of applying soccer rules to the matches (throw-in, corner kick, offside, goal, etc). The interaction between agents and simulator happens via messages, previously defined in the protocol used by the simulator and exchanged through UDP.

Recently, the possibility of using an online coach agent for players during the games was added to the Soccerserver. Such an agent has a set of restrictions in order not to break the main rule that says the players are autonomous. Thus, during the game, it has only the permission to receive visual information about both teams' players and ball positions, besides "hearing" messages sent by the referee or by players using the **`say`** command. From the analysis of data supplied by the simulator, the coach can make tactical decisions about the behavior of the team as a whole, and notify the players through the **`say`** command. The players will "listen" to the coach's suggestions during the match and will be able to optimize their internal behavior heuristics. Respecting the autonomy principle, it is not possible for the coach agent to interfere in the very behavior of the players or even to oblige them to follow its suggestions.

This paper presents a proposal of an online coach agent for RoboCup agent teams. At first, issues related to the interaction between the coach agent and the Soccerserver are pointed out. In the agent modeling section, the heuristics for team behavior evaluation are then described, followed by a brief characterization of their use, and so are the details about agent codification in the implementation section. Finally, the conclusions are presented.

2 Online Coach x Soccerserver

The commands made available by the Soccerserver to the coach agent are [3]:

<**`eye`** `[on/off]`>: Enables/disables automatic sending of visual information by the server.

<**`look`**>: Request of visual information to the server.

<**`say`** `"message"`>: The coach screams the message at the field. All the players can potentially hear the message. As a restriction, every two cycles of simulation a player may hear at most one message of each team. In other words, if a player decides to speak in the same cycle as the coach, one of the messages will get lost. Moreover, in official games, the coach can send at most 128 messages during one game. Each message may have 512 bytes, with the possibility of using letters, digits and the symbols **( ) . + * / ? < > _**.

The coach receives information from the server through two kinds of messages:

<**`hear`** `time_of_game origin "message"`>: Aural information. Origin equal to `referee` indicates message sent by the referee.

`<see time_of_game list_of_objects>:` Visual information about the positions of the ball and players.

3 Modeling

The main functions designed for the coach are the detection of playing system patterns and the collection of statistics about each player's efficiency, both constructed upon the kicking and passing detection heuristics. Besides these, a team positioning assessment mechanism evaluates congestion and marking coverage. An important characteristic of the system is its high factor in recovering lost messages, a constant risk when the chosen communication protocol is based on UDP datagrams.

3.1 Kicking detection heuristic

Since the coach does not have knowledge about the exact internal state of its players and neither of its opponents, in order to find out when the "kick" command was executed, such information needs to be constructed using the visual information made available by the Soccerserver. It was used a kicking detection heuristic based on the ball velocity variation.

First of all, the use of the gradient velocity notion was tried, and so to define a kick as a variation in the direction of this gradient or its growth in module. In tests this approach has proved to be ineffective and that is related to the noise model the Soccerserver uses. Fig. 3.1 illustrates the effect of this noise on velocity components (Vx, Vy, tangent, module) and on the derivatives of these components and their modules (dVx, dVy, dVxm, dVym, dModule).

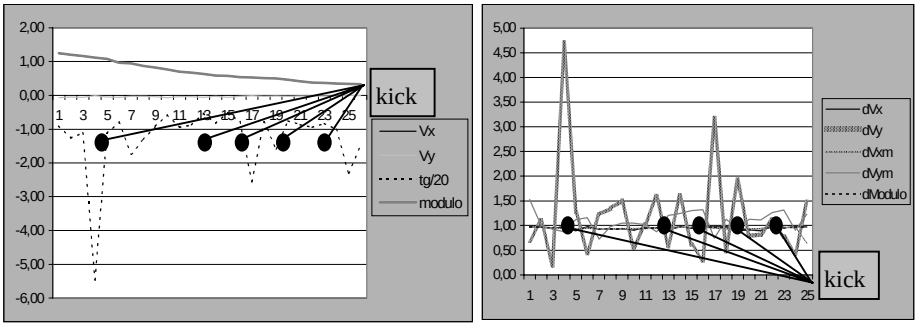

Fig. 3.1. Effect of error injection on the velocity components, modules and derivatives.

A different heuristic (Fig. 3.2) had then to be used. The main idea of this heuristic is: using the same ball deceleration model (including noise) employed by the server, to calculate value intervals for the velocity's components (*vx*, *vy*) starting from the

ball's velocity in the previous cycle. Thus, if any of the velocity's components shows a higher variation than expected, it's deduced that the ball was kicked. In tests carried out during a real game, this heuristic was able to detect more than 95% of the kicks, or even a 100% if not taken into account the weakest kicks. This heuristic may lead to false detections when the ball hits a player, though such case has proved to be very uncommon in the tests. Anyway, it's virtually impossible to detect the difference between such a case and a regular kick just with the visual information supplied by the server. The heuristic also filters jumps in the visual information, caused by the loss of messages, decreasing even more the number of false detections.

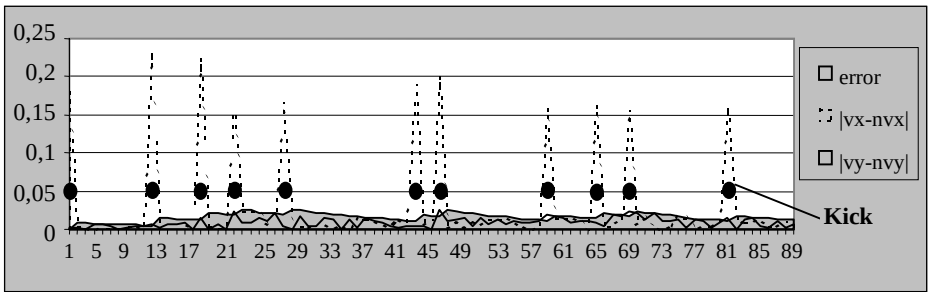

Fig. 3.2. Kicking detection heuristic at work. The gray area in the graphic indicates the maximum expected error, the other two lines represent the actual measured error for the components *Vx* and *Vy*.

3.2 Passing detection heuristic

3.2.1 Active player

After detecting that the ball was kicked, the problem is to find out who kicked it. For that, the `active_player` heuristic was defined. The active player is simply the one that:

1) can kick the ball, that means, the ball is within its kickable area
2) is the nearest to the ball

The active player information is used in the passing detection heuristic. Since generally the kicking detection occurs one cycle late, as the effect (change of ball velocity) cannot be perceived before the cause (kick), the active player of the cycle previous to the kicking detection is the one that kicked the ball. This rule has not been verified valid in dead-ball situations, in which the active player of the present cycle is used as the source of the kick.

3.2.2 Passing detection

In general, a pass is a change of the player that possesses the ball. It is quite difficult to define ball possession. In this work it has been adopted that one player possesses the ball since the moment he kicks it. Therefore, pass detection is done by the assessment of two consecutive kicks. The algorithm used is presented in Fig. 3.3.

Some important points to be considered:

- To define a pass, the consecutive kicks must be near in time, indicating that there was a playing sequence, an intention of passing. In the algorithm this is indicated by the clause "`if tracking pass`". In practice it is implemented with a timeout counter;
- There is a difference between two situations in case of error in passing the ball: the interception and the bad pass. If the time between the original kick and an opponent taking the ball is short, it is said there was an interception, which means a high-risk pass was tried and failed. In the other case, the ball was kicked but it did not reach its destiny properly due to an error in force or in directing it. This situation was simply called a bad pass;
- In case the timeout for tracking is reached, characterizing that the ball was kicked away (e.g. cleared without control), two situations are predicted: the ball is retrieved by a teammate or an opponent retrieves it.

```
-- if ball kicked
   -- if tracking pass
      -- if same team (previous kick)
         -- if same player
            <! controlling the ball !>
         -- if different player
            <! pass !>
      -- if different team
         -- if tracking interception
            <! ball intercepted !>
         -- if not tracking interception
            <! bad pass !>
   -- if not tracking pass
      -- if ball out of play (kick-off, throw-in, free kick,...)
         <! ignore, there is no previous kick !>
      -- if same team (previous kick)
         <! lost ball retrieved !>
      -- if different team
         <! bad long pass (bad lead pass) !>
```

Fig. 3.3. Passing detection algorithm.

Some external events influence the detection causing the algorithm to restart, especially messages sent by the referee such as goal, offside, half-time, corner kick, etc. It is clearly noticed that in these situations, ball movement, e.g. from the goal area to the midfield, is not a regular kick, neither a potential pass, justifying the need for restarting.

3.3 Passing detection heuristic

From the analysis of the most meaningful events of a RoboCup match, a set of metrics was selected as a basis for assessing the effectiveness of the team and the specific characteristics of each player agent. Counters for the following events are maintained for each player:

- well-succeeded pass;
- bad pass;
- received pass;
- ball controlling;
- lost ball (away kick);
- ball interceptions;
- goals.

Most of the time the update of such statistics comes from the passing detection mechanism, except for the number of goals, which is updated through the messages sent by the referee.

Observing the possibility that player behavior may vary during the game, in a certain way invalidating the previously accumulated statistics, it is expected that the set of statistics also reflect this alteration over time, “forgetting” past data. This effect is got via the periodic decrement of all of the player’s statistics, working as a periodical partial resetting of data.

3.4 Play patterns detection mechanism

As an attempt to learn the opponents’ play pattern, it was included in the coach a mechanism for the detection of playing system patterns, that is, of patterns in the passing sequence made by the opponent which frequently constitute risky plays. A risky play is known as a shot that ends up scoring a goal or that leads the opponent team to a position near to the goal area, for instance, allowing a player to enter the area with the ball.

This technique presents greater efficacy the higher the behavioral organization level of the opponent team is, i.e. more scattered positioning along the field and players that value the pass rather than controlling the ball, performing well defined roles as forward, defender, etc.

There are two approaches for the detection of this kind of play pattern: one based on the identification of each player (number of the shirt) and another one based on its position. The former’s advantage is in its simplicity, although it does not meet the case in which the players constantly switch roles in the game, in other words: players that in a moment play the role of defender, in another moment are attacking. In order to overcome this deficiency, the technique based on positions was designed. This technique records passes in the form:

$$\text{“pass region } x \Rightarrow \text{region } y\text{”}$$

The question is to define adequately these regions inside the field, and how to treat cases in which the pass is made from the position next to the border of the region, for which a slight variation in the position of the player may invalidate the pattern detection in later steps. The choice used in this work divides the field with a matrix NxM, being a region defined by four adjacent rectangles overlap naturally, solving the problem of passing in the borders. Fig. 3.4 illustrates this idea.

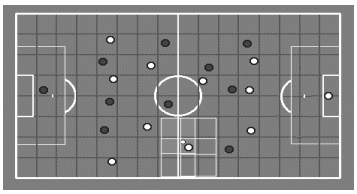

Fig. 3.4. Detection of play patterns per region.

Yet, this second approach inserts a high processing overhead, because for a unique pass, 4 regions of origin and 4 of destiny must be taken into consideration, leading to a total of 16 possibilities. For a play consisting in a sequence of 2 passes this number rises to 64, and for a sequence of 3 passes to 256 potential patterns to be analyzed. So there is an exponential increase with the length of the sequence to be analyzed. Due to the difficulty in dealing with this case in an acceptable computational time, this option of detection was not implemented.

It was used a detection mechanism that employs a tree structure like that one used in mining of association rules by the well-known Apriori algorithm [4]. In this structure, each node represents a player, and its sons, the potential pass receivers. At each detected play pattern, the counting of the frequency of nodes identified in that pattern is updated from the root node. Thus, following the branch {1, 4, 7, 9} in node (9) we will have an idea of the frequency in which the play $1 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9$ was used. In practice, the stored value consists in an absolute counting of the number of times this pattern has been repeated throughout the game, and not by a measure of frequency in itself (i.e. related to a repetition period).

The integration of this mechanism with the coach is made by a history coupled to the passing detection mechanism. For each consecutive pass inside the same team, the detection mechanism updates the pass history. External events like referee's messages indicating foul or ball out of play cause the restarting of the history. Besides, loss of the ball to the other team or too big leaps in visual information also produce the same effect. For every simulation cycle, the position of the ball is reevaluated and if it has reached a region considered critical (next to the goal area) the passes pattern stored in the history is used to update the pattern tree. Fig. 3.5 illustrates the structure used by the play patterns detection mechanism.

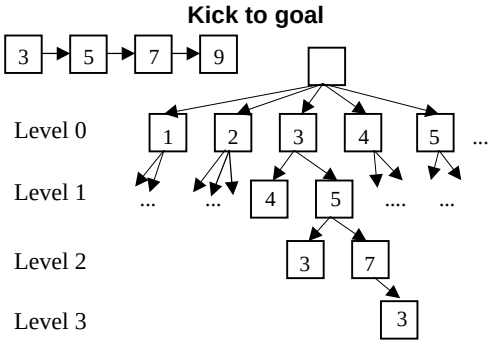

Fig. 3.5. Structure used by the mechanism for detection of play patterns.

Again, as in the case of statistics collecting, the behavior of each player, and consequently of the whole opponent team, may vary during the game. It is necessary that the detection of patterns incorporate this characteristic, getting adapted to the latest play pattern of the opposing team and "forgetting" previous behavior. This effect is reached through the periodical decrement of the players' statistics in the counting of tree nodes. Nodes whose counting stay below a chosen minimum are removed, as well as their sub-trees. However, in order to prevent more recent patterns of being eliminated right after having been inserted in the tree, an adaptation in the counting method had to be done: the nodes which are present in a pattern are incremented by *k* units, assuring that they will remain in the tree for the next (`k- removal_threshold`) cycles. In this way, older patterns will gradually lose their weight until being eliminated.

The coach can extract the patterns stored in the tree or select the most frequent ones in pre-defined periods of the game (e.g. in every two minutes or when the ball is out of play) and then make tactical decisions about the positioning of its team, altering offensive or marking characteristics. More details about its utilization will be presented in the following sections.

3.5 Congestion and marking efficiency detection mechanism

The latest designed mechanism is the detection of congestion and marking efficiency (positioning). It is based on the idea of a thermal sensor and that the players "heat" the closest areas.

The basis of this mechanism is the construction of the game's thermal photograph at each cycle. This photograph can be instantaneous or accumulative, providing a history effect. Furthermore, with the inclusion of a "cooling" model, this mechanism with temperature history starts to reflect the state of a specific moment of the game, what can be more interesting due to the potential behavioral variations during the course of the match. This mechanism is the only one for which it was projected a graphic interface. Fig. 3.6 illustrates this thermal photography technique.

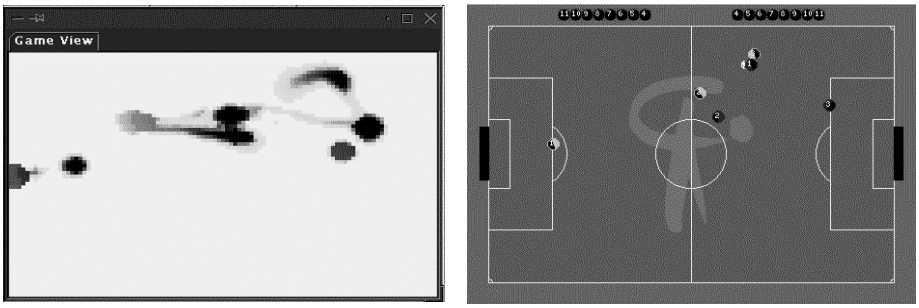

Fig. 3.6. Thermal Photograph.

In a variation of this mechanism, different temperatures can be associated to each team (e.g. a hot team and a cold team). The joining (sum) of thermal photos of each

team can lead to an identification of hot regions and cold regions of the field by the coach. In other words, regions controlled by its team or by the opposite team.

Moreover, an accumulated thermal coefficient per region can provide information about marking efficiency. This coefficient is built upon the sum of all the positions (a discrete model for the soccer field is used) of the region. If this value is negative, there is a predominance of players from the opposite team in that region. Another variation would be to sum only the cold positions (or the hot ones), which would provide an idea, in case of defense, of potential unmarked adversary players, indicating a need for repositioning if this condition persists for a significant number of cycles. Yet, in case of attack, a great factor of unmarking is strongly desired.

3.6 Characterization of the mechanisms' usage in the optimization of team behavior

Due to a limited interaction between the coach and the team (128 messages per match at most), a point to be studied is the best moment for the coach to send its messages, in order to minimize message loss by conflict. The suggested general strategy is to send messages in the periods when the ball is out of play (this is usually the strategy adopted by offline training agents). This technique is associated with a timeout mechanism to limit the maximum amount of time that the coach would wait to send a message, taking into account that the value of the message is highly contextual, a fact that makes too long waits impossible. Next are shown examples of the proposed algorithms' application.

Player statistics allow evaluating the efficiency of the pass decision heuristic used by the player. A high level of bad passes may suggest that a behavior that privileges ball controlling over passing would make the player more efficient. The existence of an adversary player with a great number of received passes makes clear that player has offensive strategy preparation characteristics, being able to require a specific marking strategy. On the other hand, a great number of interceptions indicate that the marking positioning heuristic is being quite efficient. Further, the occurrence of a high number of lost balls associated with a high number of ball controlling may indicate that a player is holding the ball too much, allowing a quick passing strategy to be more efficient.

The pattern-detection mechanism enables the identification of key elements in the opponent's playing system. As an example, if the pattern $4 \Rightarrow 5 \Rightarrow 6 \Rightarrow$ GOAL is frequent, the positioning of a player between 4 and 5 may annul that play.

With thermal photography, validation of the team's higher level positioning tactics can be reached. The coach may try to link different pre-defined positioning techniques (or formations, for example 4:5:2, 3:3:3:2, more by the left, more by the right, etc.) with the thermal history of the game and suggest a positioning that occupies better the empty regions in an attack or that covers the defense better. Supporting this decision-making it may even utilize the thermal coefficient of the region to infer, as an example, if the number of defenders is adequate.

4 Implementation

The prototype has been implemented in Java Language [5], using threads and message queues due to the asynchronous nature of the processing performed by the agent in relation to server's messages arrival, minimizing the loss of messages inherent to the use of the UDP protocol. In the implementation it is used a maximum sequence-length equal to 4 for pattern detection. Total memory occupation is around 10Mbytes, being half of this used by the pattern tree.

5 Conclusions

The preliminary tests show that the best results in play patterns detection are obtained as higher the organization level of the analyzed teams is, being inconclusive for teams of low organization level. But even for such cases, the other two evaluation techniques can be used successfully. In a certain way, the lack of a great variety of teams with advanced organizational behavior has raised difficulties to the complete validation of the model. However, it is expected that for future generations of agent teams the effects obtained from the use of such evaluation metrics, especially the metrics for pattern detection, will be more significant.

More effective tests on the optimization brought by the use of a coach agent could not be done because of scope and time issues, since it would be necessary to incorporate characteristics that do not exist at the moment in players, such as rehearsed positioning/set plays and the own interpretation and utilization of the intentions provided by the coach. On the other hand, the description of the proposed model makes it clear the potential uses of this agent. In future works the implementation of such teams of players and their integration with the coach will be accomplished.

References

1. KITARO, H. et al. RoboCup: The Robot World Cup Initiative. In: FIRST INT'L CONFERENCE ON AUTONOMOUS AGENTS (Agents'97). **Proceedings**. ACM Press, New York, 1997, p340-347.
2. NODA, Itsuki; MATSUBARA, Hitoshi. Soccer Server and Researches on Multi-Agent Systems. In: INT'L CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS – Workshop on RoboCup, IROS, 1996. **Proceedings**. Osaka, Japan. 1996, p1-7.
3. CORTEN, E. et al. Soccerserver manual. Technical report, RoboCup Federation, 2002. <http://sserver.sourceforge.net/home/downloads.html#documents>
4. AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. Fast Algorithms for Mining Association Rules. In: 20th INT'L CONFERENCE ON VERY LARGE DATABASES. **Proceedings**. Santiago. 1994.
5. Java Language documentation. <http://java.sun.com>