Classifying Adversarial Behaviors in a Dynamic Inaccessible Multi-Agent Environment

Patrick Riley

November 1999
CMU-CS-99-170

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Developing intelligent agents for multi-agent, inaccessible, adversarial environments is arguably one of the most challenging areas in artificial intelligence today. Great strides have been made in developing emergent cooperation among teammates, but less progress has been made in quickly and automatically changing overall team strategy in response to adversary actions. One way that humans do such adaptation is by noting a similarity to a past adversary. This project is a system to do that sort of classification. The system is fully implemented in the simulated robotic soccer environment as used in RoboCup. The system does the following: Each agent observes the adversary and records relevant features. Based on these observations, each agent then classifies the adversary with regards to a set of predefined behavioral classes. The agents record their classification, and the team classification is decided by a simple majority. The effectiveness of this system on some simple behavior classes is shown. Future directions can include machine learning of behavior classes and strategy changes for those behavior classes, as well as developing more complicated classes.

This report was produced as an undergraduate senior honors thesis during the 1998-1999 academic year. Many thanks go to my advisor Prof. Manuela Veloso. I would also like to thank Dr. Peter Stone for helping me first get acquainted with the Soccer Server System and for the very productive collaboration we had.

Keywords: Artificial Intelligence, Expert Systems, Games, Adversary Adaptation and Classification

1 Introduction

Developing intelligent agents for multi-agent, inaccessible, adversarial environments is arguably one of the most challenging areas of research in artificial intelligence today. Great strides have been made in developing emergent cooperation among agents. Less progress has been made in automatically changing overall team strategy in response to adversary actions. Individual decisions are often affected by the adversaries' behavior, but strategy decisions at the level of all of the cooperating agents are often fixed or can only be changed by human intervention.

Ideally, agents should effectively adjust their group behavior in response to adversary actions. One way of doing this is suggested by one way that human sports teams adjust. As they play, each player consciously or subconsciously compares the current opponent to previously played opponents. Once they have identified the particular style of play, they generally try whatever strategy was effective against the previous adversary.

The test environment for this research is simulated robotic soccer as used in RoboCup (Section 2). Some attempts have been made to do online adaption in this domain. At the level of individual skills, there has been some success. One example is the ISIS team, which changes ball interception strategies based on successes and failures during a game[4]. At a team level, the Andhill'98 team uses observationally based reinforcement learning to try and adjust the team formation[1], but this has not been very effective. Peter Stone has developed a novel reinforcement learning paradigm called TPOT-RL[7] which has been shown to be effective in learning team passing strategies. However, the learning time is such that it is not useful during the course of a normal game.

A first step in the endeavor to make artificial agents exhibit the same high-level adaptive behavior as humans is to be able to effectively *classify* adversary behaviors to determine what strategy change is appropriate. Because the environment is multi-agent and inaccessible, this step is non-trivial. It is a significant challenge to use low-bandwidth, unreliable communication to effectively combine the partial information from all the agents into a consistent whole which can be used to create an effective change of strategy.

This is the step which is addressed in this research. The algorithms are implemented in the test environment of simulated robotic soccer. The agents observe the adversary behaviors, using continuous, low bandwidth communication to keep a consistent view of the world. Then, by a global trigger they classify the adversary and record that information. Deciding on a strategy change based on that classification is not addressed in this research.

2 The Test Environment

2.1 Overview

The test environment for the algorithms and data structures developed here is the Soccer Server System used by the international artificial intelligence research initiative RoboCup[3]. The Soccer Server System was developed by Noda Itsuki[5]. It is undergoing constant revision, though an attempt is made to keep a manual[2], which describes in full the features outlined below.

The Soccer Server System is a server-client system which simulates soccer in a discrete fashion. Clients communicate using a standard network protocol with well-defined actions. The server keeps track of the current state of the world, executes the actions that the clients request, and periodically sends each agent noisy, incomplete information about the world. Agents get noisy information about the direction and distance of objects on the field (the ball, players, goals, etc.), but only for objects in the direction that the agent is facing.

2.2 Communication

Besides the natural soccer actions such as running, kicking, and turning around, the agents are able to "shout" to each other. During every cycle in the server, each agent can say a message of up to 512 characters. The reception (or "hearing") of the messages is somewhat restricted in the following ways:

- **Limited Range**: Agents only hear other agents shouting if they are within 50m of each other (the full field is 105m $\times$ 68m). This generally means that players in the middle can hear most of what is shouted, but those closer to the ends can not hear each other.

- **Limited Sender Info:** The only information about the sender that the hearing agent gets is the time and direction from which the message came. This can easily be remedied by including that information in the message itself. However, that requires that the agents have a shared frame of reference, such as global coordinates, which can only be noisily calculated.

- **Limited Bandwidth/Frequency**: As mentioned before, the messages are of limited length. More importantly though, each player can only hear 1 message from each team every two cycles. Therefore, if several agents shout at the same time, some will not be heard.

- **No Guaranteed Delivery**: No communication with the server has guaranteed network delivery, and as mentioned above, some messages will be dropped by the server if too many agents shout at the same time.

In spite of these restrictions, the agents can communicate effectively quite frequently. The 512 characters allowed in each message can carry a large amount of data.

2.3 Inaccessibility

The environment is inaccessible in two major mays. First of all, the agents visual information reflects only objects that are in the direction the agent is facing (see Figure 1). The agent can trade off size of the view cone for frequency of information, but for the purposes of this work, that trade off was generally held constant.

Secondly, and more importantly for this work, the amount of information received decreases as the distance to the object increases. When the seen object is very close, the agent receives information about its relative position and velocity. As the object is moved farther

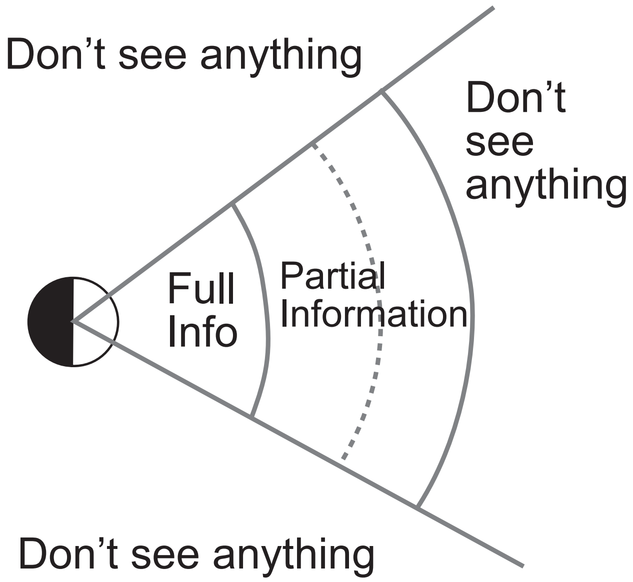


Figure 1: The Incompleteness of Visual Information

away, there is a decreasing probability that the agent will receive information about the object's velocity. Also, for seen players, there is a decreasing probability of seeing the player's number, and then further away, a decreasing probability of even seeing which team the player is on. This is shown in Figure 1. This means that if an agent watches a player moving away, it will gradually receive less and less information. This has important implications for observing the opponents, which is discussed further in Section 4.1.

2.4 Development Base

The agents used here are based upon the CMUnited '98 simulator team developed by Peter Stone, Manuela Veloso, and Patrick Riley [8]. The team won the simulator league at RoboCup'98, outscoring opponents by a combined score of 66-0. The team is based partially on the CMUnited '97 simulator team developed by Peter Stone and Manuela Veloso [6].

3 Classification

3.1 Overview

The classification of adversary behavior is done in several steps:

1. Observe the opponents' behavior and record relevant features (Section 3.2).

2. Each agent classifies the adversary based on its observations (Section 3.4).

3. Record the information (Section 3.5).

This items are performed on a fixed time schedule, which is easy since the game clock is a globally accessible feature of the world. In an environment without such global triggers, one can imagine each agent keeping track of their own time perceptions and then using communication to synchronize. This cycle of observe/classify/record is known as an *epoch*.

3.2 Observations

In general, the observations are domain-dependent. They should record some feature and define a similarity metric over different epochs of observations.

The observations are basically just feature extractions over all the visual and auditory information which the agents get. At every action opportunity (every cycle of the server), each agent updates its world model to be as accurate as possible, then records some piece of the world model for each observation. For all of the observations described here, the time internal to the epoch is discarded. For example, if we have an observation that records events of a particular type, and one happens at time 50 of a 500-cycle epoch, it will be recorded exactly the same as if it had occurred at time 400. This allows significant simplification of the data recorded, but is not an essential feature of observations.

3.2.1 RectGrid and RectGridMulti

Two data structures are used to record the observation information.

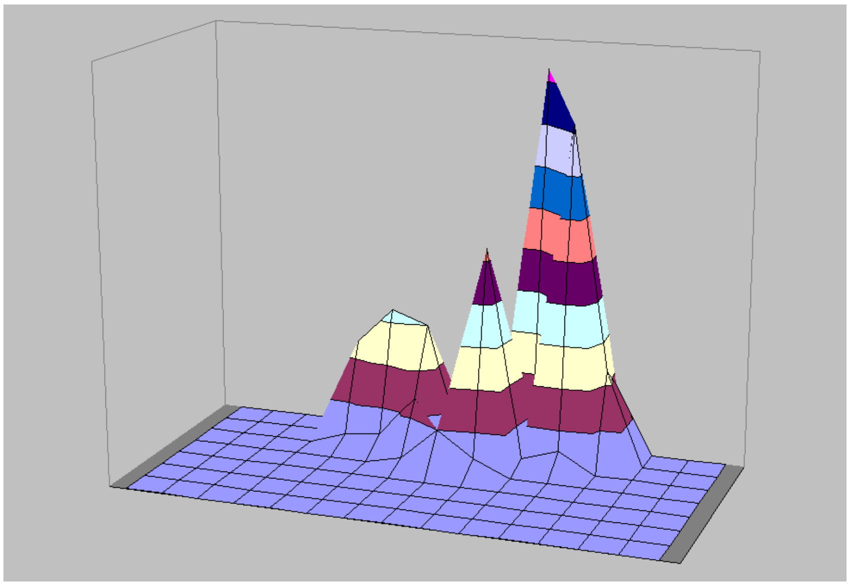


Figure 2: A example RectGrid

A RectGrid is just a division of the field into small geographic regions (or bins). The recording of an observation is just adding a count to one of the bins. The end result can be represented as a 3d-plot as in Figure 2. In all the experiments performed here, the RectGrid dimensions are 8$\times$15, with the longer dimension in the longer dimension of the field.

RectGridMulti is simply a RectGrid with an additional dimension representing direction. In other words, each bin of a RectGrid is further divided into directions. This is useful for storing information such as the direction of a pass as well as where it occurred. For the experiments here, the directions were divided into 4 bins:$[-45°, 45°]$, $[45°, 125°]$, $[135°, -135°]$, $[-135°, -45°]$.

3.2.2 Implemented Observations

- **Ball Position**: The ball's position is stored in a RectGrid
- **Opponent Position**: Each opponent's position is stored in RectGrid (so there are 11 separate RectGrids in all)
- **Opponent Passing**: This observation records all opponent passes in a RectGridMulti. A pass is defined as follows: Some opponent is in control of the ball at some cycle. Within 50 cycles, a different opponent controls the ball, with no more than 2 other players controlling the ball in the middle. It is important to allow some control of the ball by other players in the middle of a pass because if a pass goes near another player, noise in the world info can make an agent believe a player has control when he does not.
- **Opponent Dribbling**: This observation records all opponent dribbles in a RectGridMulti. A dribble is defined as follows: Some opponent is in control of the ball continuously (with no more than 4 cycles in a row where he does not control the ball) and his position changes by at least 3m.

3.3 Behavior Classes

The goal is to match the observed behavior to predefined behavior classes. The format for behavior classes is fairly simple. Each behavior class is a list of target configurations for some or all of the observations. For example, the RectGrid shown in Figure 2 represents the position of one midfielder in the normal 433 formation.

Each target configuration also has a weight that is used for matching. This way some configurations can be considered more important for a particular adversary type.

3.4 Similarity Metrics

3.4.1 Overall Matching

The procedure for matching observations to behavior classes is illustrated in Figure 3. On the left side are the different behavior classes, which include varying numbers of observation types (each shading represents a distinct observation). In the middle are the actual observations from the game in progress. As is shown, each target configuration in the behavior classes is matched to the appropriate observation from the game (a detail of this is discussed below). The similarities are then added and normalized to within $[0, 1]$.

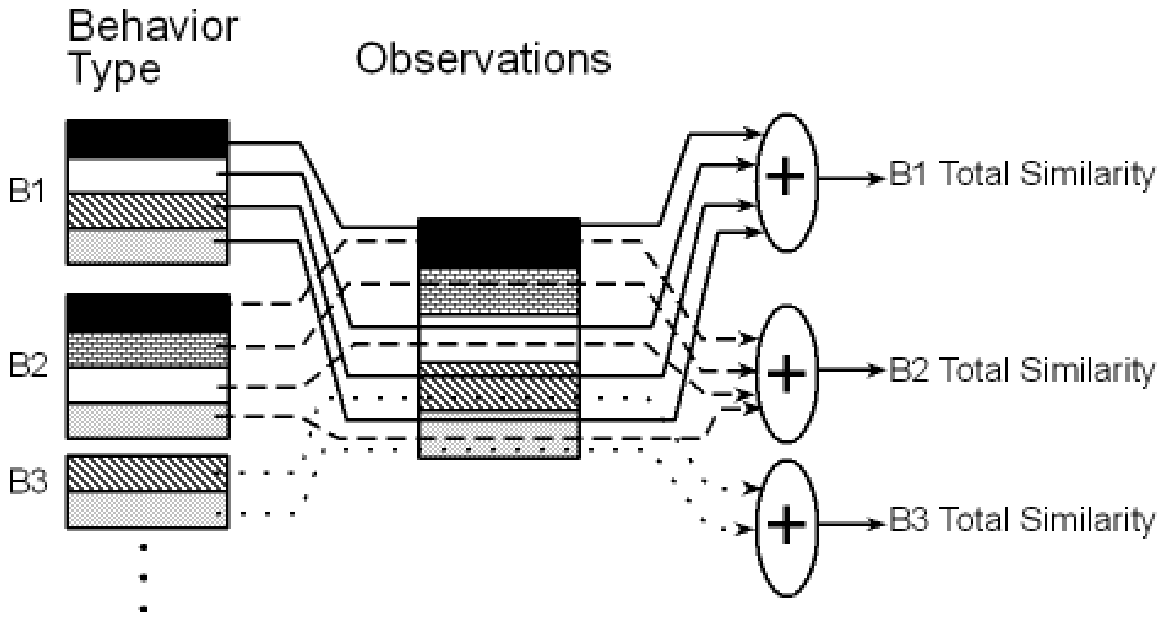


Figure 3: Matching Adversary Classes to Observations

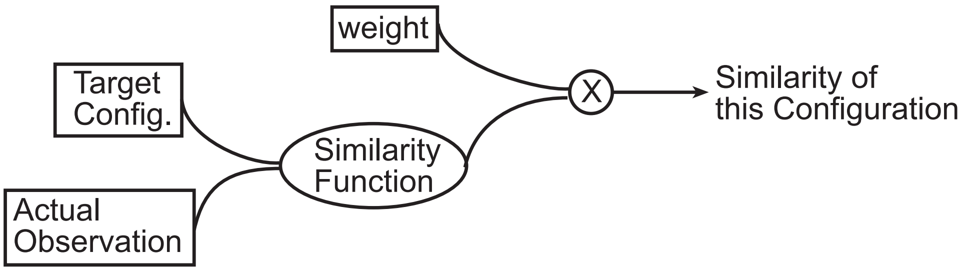


Figure 4: Matching a Target Configuration to an Observations

The procedure for matching a target configuration to an observation is also straightforward. The target and actual observed configurations are put into a similarity function (see Section 3.4.2), whose output is multiplied by the weight of this target configuration in the behavior class.

3.4.2 Observation Similarity

The similarity of the other observations is based on the metric of similarity defined for RectGrid and RectGridMulti. The only other question is for the opponent position observation: How is it decided which numbers match up? It should not be required that the same number players play the same position in order to be considered similar. Since the players can generally be distinguished by how far forward they play, the average forward/backward position of each player is calculated. The RectGrids are ordered that way and each observed player is compared to the five most closely ranked corresponding adversary class RectGrids. The most similar of those is used in the total similarity count.

3.4.3 RectGrid: A Discrete Spatial Similarity Metric

Qualitatively, the desired properties of a RectGrid similarity function S are:

- For all RectGrids a, b, $0 \leq S(a,b) \leq 1$, with 0 representing no similarity and 1 occurring if and only if $a = b$.
- If RectGrid a has the same topology as RectGrid b, just shifted slightly, then $S(a,b)$ should be large. If RectGrid c is shifted even more, then $S(a,b) > S(c,b)$.
- If RectGrid a has the same topology as RectGrid b, just scaled down, then $S(a,b)$ should also be large, but not as good as if they had the same size.
- Symmetry: $S(a,b) = S(b,a)$

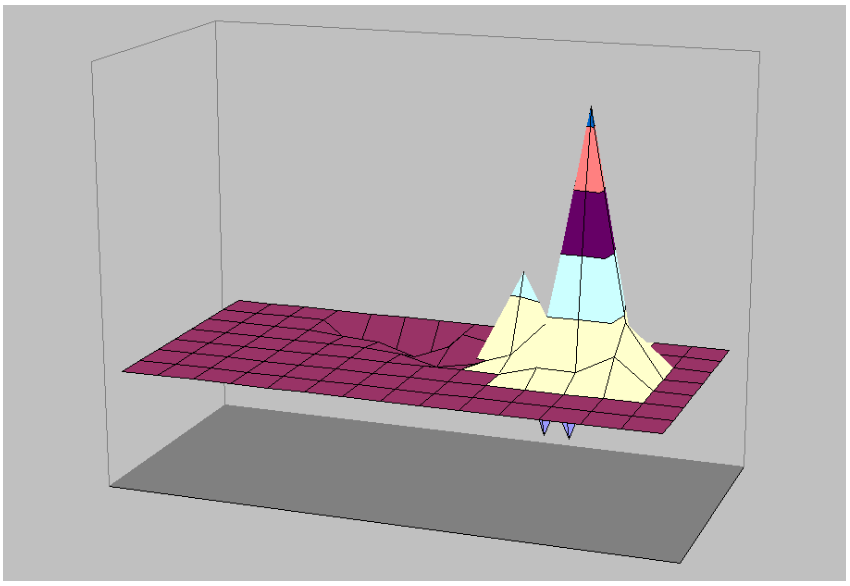


Figure 5: A example difference of RectGrids

A new metric was devised in order to achieve all of these properties. It works as follows. First, scale the RectGrids so the total in all the bins is the same. Reduce the end similarity calculated (equation (5)) by a factor according to how much scaling was required. Then, compute the difference of the RectGrids by subtracting each of the bins. This gives a new RectGrid something like what is shown in Figure 5, with some peaks and valleys. If there are no peaks and valleys, then $a = b$, so return 1. Imagine each valley v_j exerts a gravitational attraction on each peak p_i.

$$F_{ij} = \frac{|p_i||v_j|}{(\mathrm{dist}(p_i, v_j))^2} \tag{1}$$

Then divide each peak into pieces proportional to the forces acting on them.

$$\alpha_{ij} = \frac{F_{ij}}{\sum_j F_{ij}} \tag{2}$$

Next multiply the amount of each piece by the distance it has to go to get to the valley corresponding to it.

$$S_i = \sum_j (\alpha_{ij}|p_i|)\,\mathrm{dist}(p_i, v_j) \tag{3}$$

The similarity is simply

$$S = \frac{1}{\sum_i S_i} \tag{4}$$

The only problem is that this is not a symmetric relation. This is easily solved though:

$$S' = S(a,b) + S(b,a) \tag{5}$$

One can easily see that the qualitative properties desired are achieved.

3.4.4 RectGridMulti

The RectGridMulti similarity metric ignores the extra direction information and performs the RectGrid similarity metric described above. It would be possible to assign an additional attraction/repulsion for the direction info, but that would have to be weighted against the peak/valley attraction. More experimentation would be needed to establish the proper balance.

3.5 Multi-Agent Agreement

After classification, each agent records its vote, and a simple majority rule is used to determine the team opinion. At this point, the agents do not negotiate this opinion during the game through communication, though this sort of vote could easily be incorporated into the current communication system.

4 Experiments and Results

4.1 Continuous Communication

While the game is being played, the agents use the Soccer Server's communication mechanism (see Section 2.2) to exchange information quite frequently. They will always "shout" at least once in every 100 action opportunities. Included in every communication is the best guess for the location of the ball and every player, with the confidence the agent has about that information. Details of this mechanism are discussed in [8].

As discussed in Section 2.3, the visual information of far away objects is incomplete. However, the players do some basic visual tracking implemented by Peter Stone [8]. Therefore, if they see a player's number, they track that player's movement (as long as they are receiving information about the player's location), even when they no longer see the player's number. This suggests a useful procedure that the agents use. Take the simple case of two

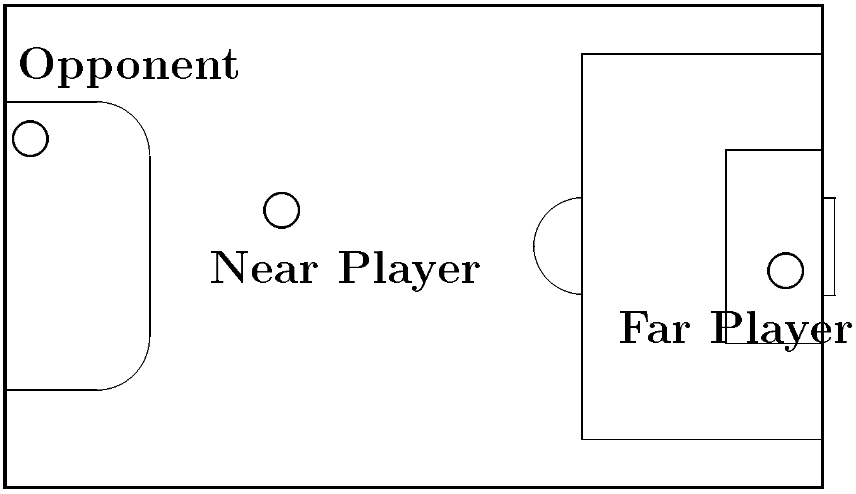


Figure 6: An Example for the Usefulness of Continuous Communication

cooperating agents looking at an opponent. As shown in Figure 6, one agent is further away, so it may not ever see the opponents number. However, the agent that is closer may have that information, which is exchanged in routine communication. Once the agent further away knows the number of the player, it can track that player as it moves around and be reasonably sure that it it is the same opponent even if it never gets visual confirmation of the number of the opponent.

It was hoped that this sort of communication would allow each agent to have a more complete view of the world. This is supported by the results. For each player, the percentage of the time that it knows where each of the opponents are was recorded. Those percentages were then averaged over all opponents. This was done when the agents were and were not communicating. In Figure 7, this difference is illustrated. Some players benefit more from communication; for example, the goalie (number 1) sees an increase of about 50% in the amount of time it knows where the opponents are.

The classification agreement (discussed in Section 4.2.3) further supports the agents having a fairly consistent view of the world.

4.2 Adversarial Classification

4.2.1 The Teams and Classes Used

In order to test the classification, three separate formations were used:

- **Normal 433**: This is the formation that was used most often for RoboCup'98, with 1 goalie, 4 defenders, 3 midfielders, and 3 attackers.
- **Left**: All of the players home positions in the formations are on one long side of the field. That is, if you draw a line from one goal to the other, all but one of the players line up on one side. There is one midfielder who lines up slightly on the other side. This formation is also a bit defensive; it has 1 goalie, 4 defenders, 4 midfielders, and 2 strikers.

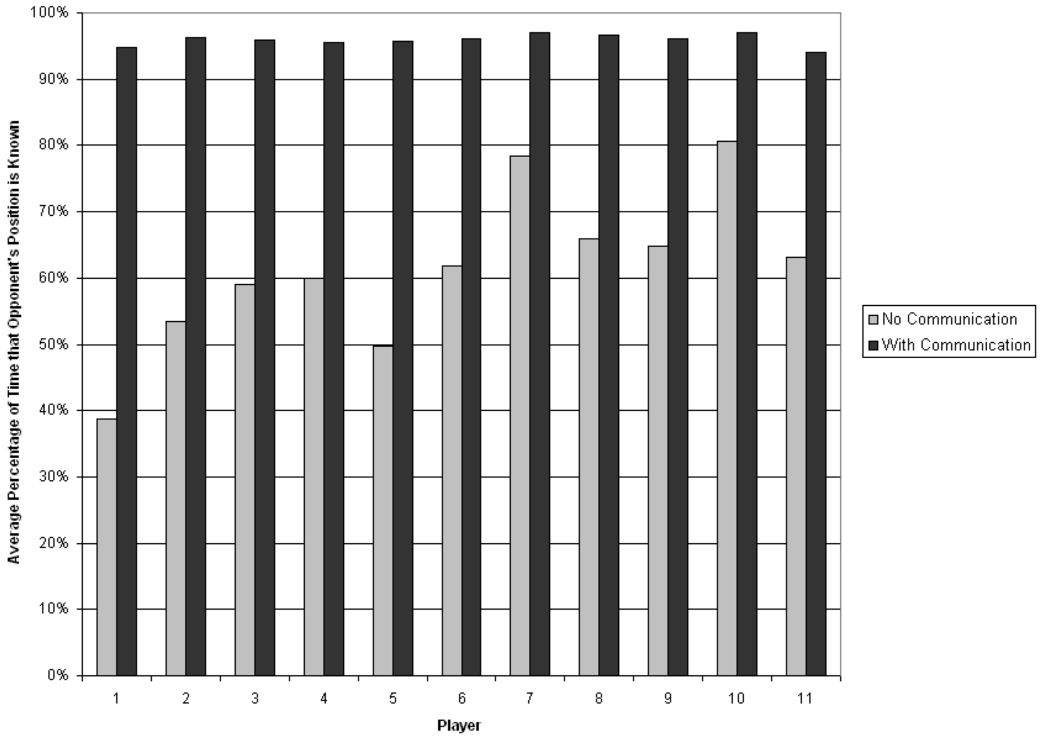


Figure 7: Results of Using Communication

- **Right**: This is the same as the left formation, but on the other side of the field.

Identification of these three teams is not as simple as it first seems. There are three behaviors that complicate this:

- **Marking**: "Marking" basically means that the defenders are assigned to opponents and position themselves nearby to intercept passes, prevent dribbling, and block shots. The players home position is completely ignored here. If the opponent strikers spread across the field, then so do the defenders who mark them.

- **SPAR**: SPAR is an offensive positioning algorithm used in CMUnited'98[8]. When one striker has control of the ball, the others position themselves by taking into account the the position of the ball, the opponents, and the opponents goal. Once again, the home position in the formation is not taken into account.

- **Set plays**: Set plays are used in all free-kick situations. This includes any time the ball goes out of bounds (thrown-ins, goal kicks, corner kicks, etc) as well as penalty calls like offsides. A subset of the players have definite positions and roles in a set play. For example, if there is a throw-in on the right side of the field, even the left team will send several players over to that side to perform the set play. The set play can take several hundred cycles to start and execute, so this can significantly affect the results of a 500 cycle epoch.

In order to generate adversary classes for these three formations, each formation was played against a normal formation team who gathered all the observations described in Section 3.2.2. The teams played several games for a total of 75-100 epochs. The recorded observations were then averaged together to form the behavior class.

After the behavior classes were generated, each of the teams was played against a classifying team (which used the normal 433 formation). The game was run for 100 epochs, or 50000 cycles altogether, and each of the agents of the classifying team recorded their decision at the end of each epoch. The next two sections discuss those results.

4.2.2 Classification Accuracy

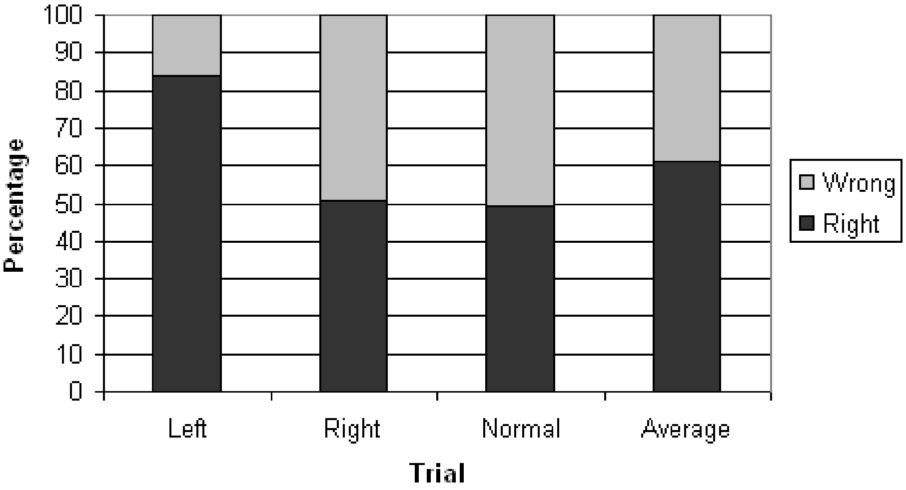
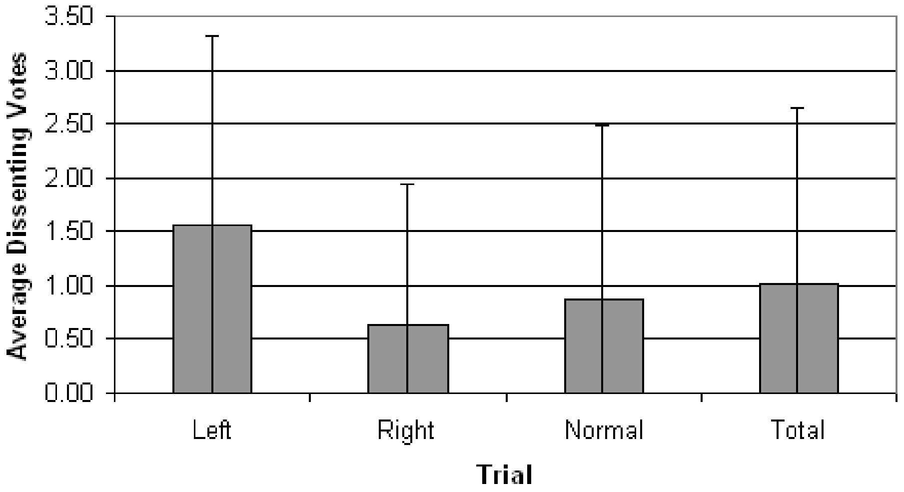


Figure 8: Classification Accuracy

The percentage of the time that the team decision (decided by majority vote) was the same as the actual formation of the observed team is shown in Figure 8. When playing against the left team, the accuracy was above 80%, but for the other two, the accuracy was only about 50%. In any case, the selection is still significantly better than random (33%). It is not known why there is such a disparity between the different trials. It is possible that the teams were stuck in a set play for a long period of time, which could push down the classification accuracy. Also, for the Right team trial, one of the observing players stopped recording votes after 21 epochs. It is possible it also stopped playing after that time. Its failing to move may have thrown off other opponent position based behaviors in the observed team.

4.2.3 Classification Agreement

The other important feature of the team votes is the agreement between them. The number of votes not cast for the majority are illustrated in Figure 9. The dissenting votes are counted regardless of whether the classification with the most number of votes is the correct



Trial	Average	Standard Deviation
Left	1.56	1.77
Right	0.64	1.31
Normal	0.87	1.61
Total	1.02	1.62

Figure 9: Dissenting Votes for Team Classification

one. This gives some measure of how consistent a world picture is given to the agents through continuous communication (Section 4.1). It should be noted that since one player stopped recording votes during the right trial, one would expect a slightly lower average dissenting vote count. The average of approximately 1 out of 11 votes disagreeing is quite good, but the standard deviation is high. The players are occasionally getting into situations where the majority is decided by as slim a margin as 1 vote.

5 Conclusions and Future Work

All in all, the results are not totally conclusive. The system is able to effectively classify significantly better than average in some test cases, but performs significantly better in some cases rather than others (as demonstrated by the left results). More trials could be performed to explore why the classification accuracy varies such a great deal. Also, the multi-agent agreement needs to be worked on. In most cases it is good, but on some epochs there are wildly disagreeing opinions.

Adding more observations and more behavior classes would also help to test the system. The observations here were very simplistic. Creating observations that record more interesting and critical features would be a big step in making the system applicable to a real game situation. Also, the adversary classes demonstrated here were very simple. Identifying how to group adversaries into strategic classes is a significant challenge. Also, with more adversary classes to choose from, picking the right one would be more difficult, and would further demonstrate the effectiveness of the system.

Another thing to add to the system would be a notion of *a priori* probabilities for behavior classes. Once there is a significant library of behavior classes, having these probabilities may help choose between them.

This system could also be applied to an on-line coach. The Soccer Server System is implementing a facility for a coach as an additional research outlet. The coach has a global world view and can shout to the players only when the ball is not in play. The coach could perform the observations and classification himself, shouting to the players just the class into which the opponent falls. Each agent would then have the appropriate behavior change stored.

There are two excellent learning opportunities in this system. The first is in creating observations. The observations used here were hand coded, and automatic extraction of these features is an interesting challenge. Taylor Raines and Milind Tambe are developing the ISAAC system which does an automatic feature extraction over soccer server games[9], though the features are slightly different than those used here .

Second, the part of the process which should occur after classification is not addressed here. This part is team selection of a strategy which will improve performance against the adversary. With the system given here, the team will have a consistent classification of the adversary, so a simple mapping of behavior type to strategy is needed. This is once again an excellent opportunity for machine learning.

Overall, this system has shown some effectiveness at classification in a very small amount of time in the Soccer Server System. Further expanding the system both in the soccer environment and to other environments is an interesting research task. Developing systems which can adapt during the course of a game is an important step in the long term AI goal of developing agents with abilities more like humans.

References

[1] Tomohito Andou. Andhill-98: A robocup team which reinforces positioning observationally. In Minoru Asada, editor, *Proceedings of the second RoboCup Workshop*, 1998.

[2] Johan Kummeneje (ed). *Soccer Server Manual.* RoboCup Simulation League, `http://www.dsv.su.se/~johank/RoboCup/manual/`.

[3] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The robocup synthetic agent challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 24–49, San Francisco, CA, 1997.

[4] Stacy Marsela, Jafar Adibi, Yaser Al-Onaizan, Ali Erdem, Randall Hill, Gal A. Kaminka, Zhun Qiu, and Milind Tambe. Using an explicit teamwork model and learning in robocup: An extended abstract. In Minoru Asada, editor, *Proceedings of the second RoboCup Workshop*, 1998.

[5] Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998. up to date description can be found at `http://ci.etl.go.jp/~noda/soccer/server/index.html`.

[6] Peter Stone and Manuela Veloso. The cmunited'97 simulator team. In Hiroaki Kitano, editor, *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*, Berlin, 1998. Springer Verlag.

[7] Peter Stone and Manuela Veloso. Team-paritioned, opaque transition reinforcement leanring. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, Berlin, 1999. Springer Verlag.

[8] Peter Stone, Manuela Veloso, and Patrick Riley. The cmunited-98 champion simulator team. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, Berlin, 1999. Springer Verlag.

[9] Milind Tambe, Gal Kaminka, Stacy Marsella, Ion Muslea, and Taylor Raines. Two fielded teams and two experts: A robocup challenge response from the trenches. (will be appearing at IJCAI '99).