Detecting Opponent Roles in a Robot Soccer Domain

Jennifer Lin
Advisor: Dr. Brett Browning
April 2004

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, 15289

Abstract

Robot soccer provides a good domain for role-based opponent modeling because roles are well-defined and integral to the play system of the opponent. Specifically, I detect roles and role transitions in the adversary team. This information can then be used to exploit the opponent's playbook in order to defeat them. The research can also be applied in other formation detection domains. It can rely on a variety of variables that have not been extensively explored before. Role detection is achieved with a combination of hand-coded rules and Markov Models. These methods were tested on the CMDragons02. Results show promise that almost every role can be detected with at least 70% accuracy given a complete world model. While role detection can indeed be accomplished, it was found that Markov Models were not the method of choice.

Table of Contents

Chapter 1

Introduction

This thesis work is on role detection in the small-size league of robot soccer. Real time observations are used to make predictions about the role of a given opponent robot.

1.1 Motivation

Plenty of work has been done in role detection in many fields. This thesis will focus on looking at role detection for robot soccer. Role detection can be useful for automated announcing of sporting events, recognition of patterns and formations in a military domain, and air traffic control as well as many others. Also, as more robots are integrated into the human environment, the ability to model other agents is going to become more essential.

Specifically for robot soccer, role information can be used to stop the opponent team. Knowing the role of a robot can allow us to exploit opponent plays by preventing them from getting to the ball, or by finding holes in their defense. For instance, if it can be detected that the role of an opponent will make it likely to try to get open for a pass, another robot can be sent in the path to intercept a pass.

1.2 Problem

The objective of this thesis is to successfully detect the role of each opponent robot on the field at any given time during a game situation. Specifically, it will answer the question:

> **How can an autonomous robot team build models of an unknown opponent team in real time in hopes of eventually using those models to exploit the actions of the opponent?**

The roles that will be detected are the goalie, the active player, the defense, the supporting offense, and the mark. Since the rules of robot soccer include a description of what a goalie can or cannot do, every team has a goalie. The active player simply means a robot going for the ball. If no robots ever went to the ball, the game would not be very interesting. A common role is a defensive player. Teams have designated robots to stop the opponent and to score. Those designed to stop the opponent via positioning themselves in a zone are called defensive players. Those designed to stop the opponent by staying next to them or "marking" them are marks. Robots that try to get open for a pass in order to score a goal or advance the ball are designated as supporting offense. This thesis is meant to be generalized for all teams, however specific team roles differ greatly. These categories of roles make it more feasible to generalize the research across more uses.

1.3 Approach

To detect the roles of an opponent robot, two different methods were tested. First, a series of hand-coded rules were written to discern between different roles. Then, a series of Markov models were found based on the distance of the robot from the goal and the robot to the ball to try to improve upon role detection. Both of these methods are detailed later.

In order to train and test these models, data provided from CMDragons02 was used. While it would be nice to use the logged data from the other teams, there is no information on what roles they are actually running, which makes it difficult to tell whether or not the system is doing the right thing.

1.4 Contributions

The main contribution of this thesis is providing a method of identifying roles in real-time based on observations. This research work will allow the game of robot soccer to move away from the current common state of static playbooks. Also, role detection techniques can be applied to other fields where formations and path patterns are of importance by providing a reliable method of classifying agents into roles.

This thesis also presents a new way of using Markov Models. Specifically, a Hidden Markov model of roles is used, and the probability of an observation is based off of a Markov model of observations for that role.

1.5 Domain

Robot soccer provides a fast-paced dynamic autonomous game. Started in 1997, RoboCup consists of five distinct leagues. [RoboCup] These leagues are the legged, small-size, mid-size, humanoid, and simulation leagues. Each league has its own characteristics. Specifically, as of 2003, the small-size league consists of robots of up to 180mm diameter and up to 150 mm tall that move up to 2 meters per second and a ball that can reach velocities of over 5 meters per second. An orange golf ball is used on a flat green carpeted field that measures 2.8 meters by 2.3 meters. An overhead camera allows the position of robots from both teams, as well as the position of the ball to be tracked by frames. Specifically, a camera that achieves thirty frames a second was used for this the purposes of this thesis.

The positional information is then fed into an off-field computer that runs software and sends commands back to the robots by radio. This positional information is represented as an X-Y coordinate system. The center of the field is at (0, 0), and the X-values decrease towards the defending goal and increases towards the attacking goal while the Y-values increase to the left when facing towards the attacking goal and decrease to the right. Once the game starts, there is no human intervention allowed; the robots must play and set-up by themselves. Teams consist of up to five robots including the goalie. Since robots are built by individual teams, the capabilities of robots differ from team to team and sometimes even differ from robot to robot within a team.

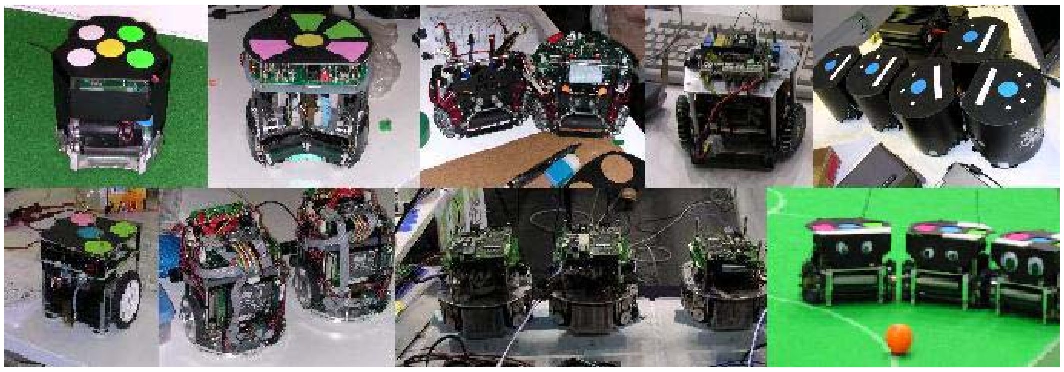

Figure 2.1: Various Robots Used by Different Teams in RoboCup 2003

Nearly all teams have a static set of roles and a static set of plays made up of different combinations of these roles. Therefore roles are well-defined enough to detect. Generally, there is a multiple-layer software system running the robots. The bottom layer defines how each robot moves in order to maneuver with the ball and around the field to perform tasks such as shooting, dribbling, and positioning. The next layer defines how a specific robot would play an attacker or play defense or another role based on the tasks in the lower level. Finally, there's a top layer that defines how each of these robots in these roles interact with each other for teamplay. This research would allow work to be done in order to dynamically write this top layer, since it would be useful to detect roles for the purposes of dynamically creating new plays to defeat an opponent, real-time without human intervention.

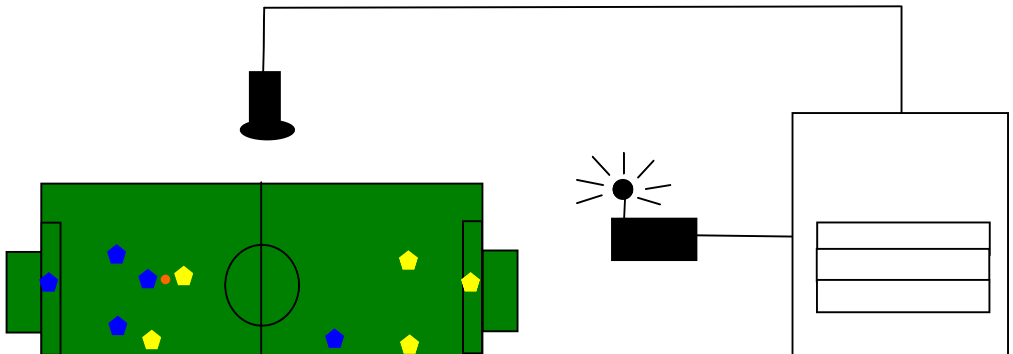

Figure 2.2: Small-Size System

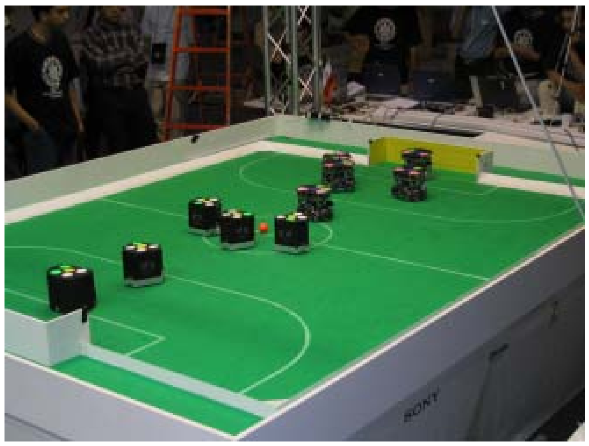

Figure 2.3: A Photograph from an Actual Small-Size Robot Soccer Game

While access to two full teams of physical robots is extremely difficult to find, there is access to a simulator of physical robots called UberSim [BroTry, 2003]. This allows the code to be tested without 10 physical robots on the field. UberSim was written to simulate the CMDragons team of small-size robots. It sends frames of data to software the same way the information from the camera is sent; the software cannot tell the difference between a simulator and the real camera. Unfortunately, there are a few problems in UberSim. Although positional data has 100% confidence since there is no occlusion or lighting problems, the physics built into UberSim is far from perfect. Often times, a ball may shoot off the screen, get stuck in a wall, or fly above/through robots on the field. In many of these cases, the ball cannot be manually reset and instead the whole system must be reset such that UberSim repositions each robot and the ball randomly somewhere else on the field and keeps running as if nothing happened. This forces the robots to take time to travel back to their positions on the field, providing noise in the data. This is also the case when robots sometimes collide into each other and disappear off the screen. All of these factors result in noise in the system that is difficult to filter out and lead to more inaccurate results later.

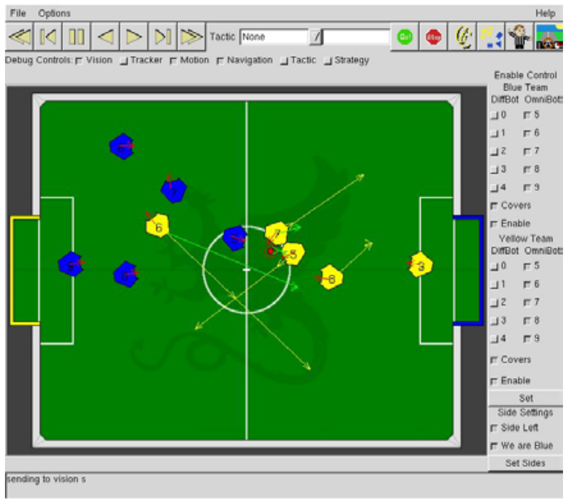

Figure 2.4: A Screenshot from a GUI running UberSim

Also, there are over twenty logged games from RoboCup 2002, American Open 2003, and RoboCup 2003. The logs can play back like an actual game and the software can detect the roles of opponents played in the past in these real situations. These log files allow the code to be tested on a variety of opponent teams who most likely run roles and plays that differ greatly from CMDragons. Logs play back all information from vision that was received at the time with the complete frames, as well as provide information as if the software was running at that time. They were recorded at thirty frames a second.

1.6 Related Work

In robotic soccer, most of the work regarding role detection has been done in the simulation league. The legged league as well as the mid-size league have a difficult enough time tracking multiple targets, that they do not do opponent modeling. The simulation league uses information mostly from past games, but also some real-time information in set-play situations [RilVel, 2002][Ste, 2001]. However, the interactions between robot and robot as well as robot and ball are also less predictable in the real game where simple physics equations are used to map this out in the simulator. For instance, a pass becomes much harder in small-size because the kicking speed if significantly faster than the combination of latency and the robot's ability to respond to changes. For half a meter, the robots do not even see the ball before it is about to hit them. If a pass comes off the kicker a little bit off, it is difficult for the recipient to adjust for this since only a fraction of a degree of error is allowed given about a 6 centimeter dribbler and passed from a half meter away.

Related work has also been done outside of robotic soccer in the form of using recorded American football games [Int2001]. Bayesian networks were used. Of the 25 known plays they used, 21 were correctly identified. This may be applicable to small-sized robotic soccer, but requires knowledge of the opponent's playbook.

As for deployed systems within small-size, CMDragons02 used basic opponent modeling in the F-180 league as far as trying to determine whether to shoot or to pass to another player based on the goalie speed and the speed of our kickers. Prior to that, CMDragons01 started the use of adaptive playbooks.

Meanwhile, the idea of using Markov models comes from the research done by [HanVel, 1999] and [Boh, 2002]. Both groups used Hidden Markov Models to automate detection of robot behaviors. Specifically for [HanVel, 1999], the robot's distance from the ball was used as an observation to determine whether or not the robot was going for the ball. Finite windows of data were used as input to the system. Every few time steps, the window would be reset. This concept seems to be similar enough to apply to role recognition since the robot's distance to the ball is an indicator of certain roles.

Using the positional information from the field also comes from [Tam, 1996]. This paper shows that tracking individual agents helps in plan recognition. Examples are shown through both RoboCup and a simulated air combat situation.

What makes this research unique is that it deals with real robots and it collects and uses the data real-time in order to distinguish between roles. As the game is going on, the opponent modeling occurs and the information is immediately used to determine the next action of the team.

Chapter 2

Objectives

For role detection to be successful and meaningful, many things must occur. If these objectives were not met, then even if roles were successfully identified, the information would not be useful.

- **The system must be capable of detecting roles in real-time.**
- **The system must have a low latency.**
- **The system should be generic enough to work with any team.**
- **The system must only rely on positional and temporal information.**

Although there are logs of games in years past, the system must work in real-time making no assumptions from data from previous games. The information from the data is not very useful for learning because teams will change drastically from year to year. Almost all teams make significant changes in hardware every year, which affects the physical capabilities of the robots themselves. A faster robot and stronger kick allow teams to also change their plays and strategies as well.

Since the role information needs to be used immediately in order to respond to opponent moves, a latency of a few seconds is far too high. The system should be capable of achieving role detection within a few frames time.

There is no standard for roles defined in each team's system. Therefore, roles cannot be too specific to a particular team. There are over fifty small-size teams around the world.

The only information available is from the camera. Therefore, the system must be able to detect the roles only based off of the position of robots and the ball through time. Specifically in this case, it must be able to detect the roles using positional information at a resolution of thirty frames a second.

If none of these objectives were in place, then it is possible to end up with a system that could only detect roles from one specific team using past information assuming no changes and took several seconds to run. It would then be impossible to create new plays that relied on knowing the roles of the opponents, and would not work on any other team or even on the same team if the code was even slightly modified. Therefore, each one of these must be achieved in addition to the successful role detection.

Chapter 3

Hand-coded Rules

The first approach to detecting roles was to create a number of hand-coded rules based on domain knowledge. This approach should be easy to do and straightforward. Results from this approach should show whether or not the task at hand is possible while later approaches attempt for more accurate results.

3.1 Motivation

Since the robots the behaviors that make up robot roles are hand-coded, it seems logical that they can be detected by hand-coded rules are well. For instance, a robot running mark is told to stay in close proximity an particular opponent robot, staying between it and the ball. Therefore, they follow a very particular movement pattern and positioning sequence. Simple rules should be able to pick up on these. Here are some histograms diagramming this fact:

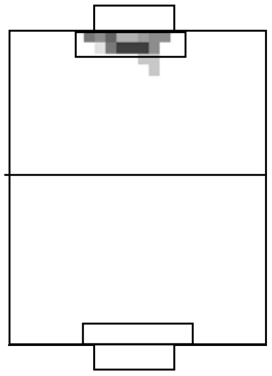
Figure 3.1: Histogram of Goalie Positions

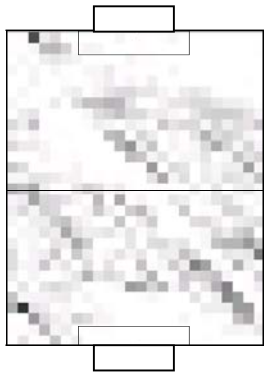
Figure 3.2: Histogram of Positions as Active Player

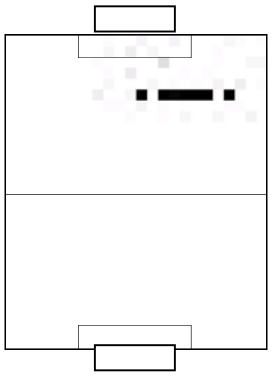
Figure 3.3: Histogram of Robot Positions as Defense

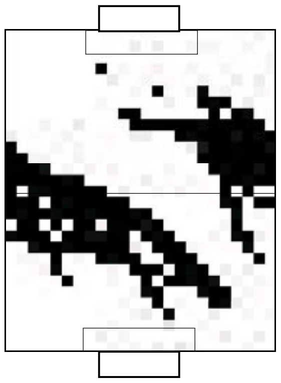
Figure 3.4: Histogram of Robot Positions as Supporting Offense

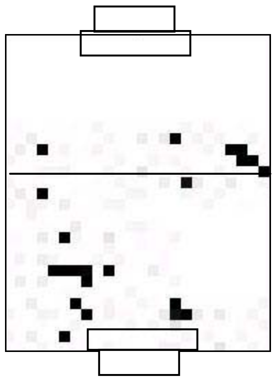
Figure 3.5: Histogram of Robot Positions as Mark

The darker splotches on the plot represent areas of the field that the robot is at more often given that role. White areas represent areas where the robot is never at. As is evident by the plot, the most common locations of each robot when their positions are plotted on a histogram is very specific for each role. For instance, even though the active player is the one that should be following the ball, it still forms patterns with its positioning on the field. Other roles like goalie and defense are more evident. Based on these locations, rules were written to discern between the roles.

3.2 Hand-coded Rules Approach

Of the five roles, the first role to pick out is the goalie. This is the robot designated to defend the goal from shots on it. Only one robot plays the position of goalie throughout the whole game. The goalie has special privileges. For instance, it is the only robot allowed in the goal box. Also, when in the goal box, an opponent robot cannot touch the goalie. Therefore, as long as the goalie is playing his position properly, this robot is usually the closest to the goal. Therefore, a simple minimum function was applied to the distance of each robot to the goal to find which robot was the goalie. While hysterisis may sound like a good idea, it was found to be unnecessary, especially since no other robot but the goalie is allowed in the goal box according to the rules, and goalies generally stay in the goalie box since they have the advantage that no opponent robot may legally touch the goalie in the goal box. These factors all amount to the goalie always being the nearest to the relevant goal.

The next role to pick out is the active player. The active player is defined as the one directly interacting with, or about to directly interact with the ball. Therefore, it is easy to say that robot closest to the ball is the active player. In the case where the goalie is the one with the ball, the robot is identified as a goalie and the role of active player is not assigned to any robot

The role with the next more specific rule is the defensive player. The defense is a player guarding off a certain zone of the field, preventing an opponent robot from getting the ball or shooting in that area. A player is identified as being on defense if its center dot is in the triangle created by the ball and each goal post. A confidence level was determined for each robot by taking the difference in y-positions of the robot and triangle, where the Y-position represents how far to the right or left of the triangle the robot is, when facing the goal. This confidence level was based on a Gaussian distribution. The confidence was then thresholded, causing additional robots to be declared as defenders.

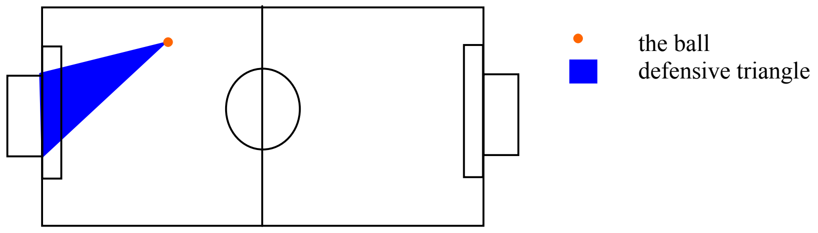

Figure 3.6: The Defensive Triangle used for Recognizing Robots Running Defense

After coding the rules for the defensive player, the rules for supporting offense go into effect. This role represents robots that would try to drive to the open space to receive a pass or a loose ball in order to advance the ball up the field or to take a shot on goal. The supporting offense is always on the offensive side of the field, but its position relative to the ball or other robot could be a variety of things. Therefore, it was simply anything on the offensive side that wasn’t on defense.

The final position is a special position called mark. The mark is a robot assigned to preventing another particular robot from getting the ball. In this role, a robot is defending another robot by following it regardless of the ball’s position. The code actually classifies whether or not the robot is running mark after active player, since it is such a special case. In this case, the nearest robot to each robot is logged. If the nearest robot is further than a certain threshold or keeps changing, the robot is *not* running mark. Unfortunately, this is the hardest role to discern since sometimes many robots are clumped around the ball and therefore are always near each other, especially if the ball has trouble getting free from the crowding. This is still an important role because it is the distinguishing factor telling whether the opponent is running a zone defense, a robot-to-robot defense, or some combination of both strategies.

3.3 Experimental Set-up

Information to test the software was collected by having the simulator play twenty-minute five-on-five games. When a goal was scored, the ball was replaced in the middle without the robots setting up again. There were no stop plays and as little time as possible was spent with the ball in the goal. In cases where the ball went hopeless out of bounds, the simulator was quickly reset, repositioning all robots and the ball in random positions on the field. Records were kept for the score of each team and the number of resets that were required during the game.

3.4 Results

The roles of each opponent robot was detected during the game with some success. Several trial were run using the simulator, allowing each team both output what roles they are picking for themselves as well as what roles they think the opponent is running. For each trial run, a confusion matrix was calculated in order to analyze how well each role was picked out and what roles the software could not discern at all.

	Goalie	**Active Player**	**Defense**	**Supporting Offense**	**Mark**	**Not Sure**
Goalie	**0.995**	0.000	0.002	0	0	0.003
Active Player	0.000	**0.923**	0.055	0.008	0.006	0.008
Defense	0.001	0.032	**0.714**	0.121	0.065	0.066
Supporting Offense	0.001	0.003	0.171	**0.709**	0.042	0.074
Mark	0.000	0.049	0.193	**0.719**	0.022	0.017

Table 3.1: Hand-coded Rules Confusion Matrix

Noise was expected in the matrix, due to multiple resets that occurred during the twenty minute test periods, forcing robots to end up somewhere random and take time to move back into position. There was an average of 2.3 resets per game. Each reset forced the position of the robots to be replaced randomly elsewhere in the screen. This often causes the goalie to end up on the other side of the field and the active player to be far from the ball and such. It is difficult to filter out when this happens, so it just becomes noise in the system.

In many cases when the ball is in the far corner, a robot will start marking the goalie or the active player. This is a case in the play system where the code does not work as intended. While the program is really running mark, the software tends to see that it looks like it’s a supporting offensive player when it is marking the goalie, and an active player when it is marking the other team's active player. This case appears to be a flaw in the testbed, not the software since a supporting offensive player would make more sense in such a scenario. Also, the play system also often chooses to mark the active player from the other team. This causes a lot of crowding of the ball and looks more like any of the other roles except for goalie and itself.

There was also a problem sometimes of deciding between defense and supporting offense. Part of this case is because of a tactic in the play system run called "defend_lane". Robots are often assigned to defend_lane near the center of the field. The reason for this is so that they behave more like midfielders and can easily transition into a deflection play or a pass play almost like a striker position in real soccer.

Goalie	Active Player	Defense	Supporting Offense	Mark
0.0150	0.043	0.829	0.061	0.053

Table 3.2: Real Roles During "Not Sure"

About 3.7% of the time, the software was not sure what position to identify a robot as. Of this percentage, 82.9% of the time, the robot was actually running defense. This could stem from two problems: either defend_lane was confusing it again when the robot is towards the middle but not on the offensive side of the field to be confused with supporting offense, or the threshold on the Gaussian is too high. To compensate the threshold was lowered slightly.

Another real problem encountered in the confusion matrix was that a robot would decide to switch from defense to active player or supporting offense. At some point it starts moving into position to become active player and the programming running that robot would them output that it has already become the new role where my program continued to recognize it as the old role for a few frames before catching up. To deal with this new problem, rules were added to detect transitions. This is done by comparing the robot speed to the speed of the ball. When the robot speed is significantly higher than the speed of the ball, the robot is in all likelihood speeding to its new position due to a role change. A robot detected to be transitioning between roles was not included in the data.

	Goalie	Active Player	Defense	Supporting Offense	Mark	Not Sure
Goalie	**0.984**	0.013	0.000	0.003	0.000	0.000
Active Player	0.001	**0.952**	0.027	0.016	0.002	0.002
Defense	0.034	0.095	**0.677**	0.160	0.019	0.015
Supporting Offense	0.000	0.036	0.007	**0.890**	0.005	0
Mark	0.043	0.260	0.096	**0.585**	0.014	0.002

Table 3.3: Handcoded Rules Confusion Matrix with Transitions

Goalie	Active Player	Defense	Supporting Offense	Mark
0.024	0.100	0.853	0.000	0.024

Table 3.4: Real Roles During "Not Sure" with Transitions

Making these changes seemed to have increased recognition of supporting offense significantly. The software is now never “not sure” when the robot is running a supporting offensive role, and it is no longer significantly confused with defense. Meanwhile, the detection of defense still suffers and usually when it’s “not sure” it is defense. When the software thinks that it is supporting offense, sometimes it is actually defense.

3.5 Summary

Hand-coded rules faired decently except in a few special play-specific situations, with mark, or certain defense situations. A few problems were encountered regarding transitions and boundaries of offense and defense. Solutions to these were added and the tests were reran turning the results for supporting offense from decent to accurate. In the final set of code used for this part of the thesis, the recognition of goalie, active player, and supporting offense were accurate while the recognition for defense was decent and for mark, it was poor.

Chapter 4 Markov Models

Hoping to get more accurate results, a second method of role detection was attempted. This method used distance metrics as observations for a series of Markov Models. The idea of this method developed from work from [HanVel, 1999]

4.1 Motivation

While the hand-coded rules were somewhat effective in recognizing opponent roles, but they did not really take advantage of the patterns available in the continuous temporal stream of data. Therefore, another approach was used for comparison. Markov models allow a series of state transitions from observations to be incorporated into the decision using the continuous data to determine what role the robot is running. Nine different states were defined based on observations of distance from ball and position on the field.

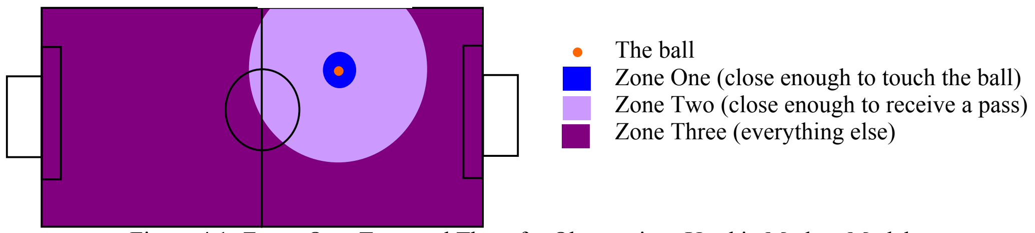
The ball
Zone One (close enough to touch the ball)
Zone Two (close enough to receive a pass)
Zone Three (everything else)

Figure 4.1: Zones One, Two, and Three for Observations Used in Markov Models

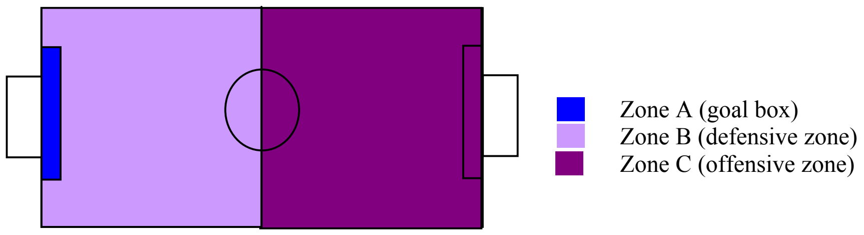

Figure 4.1: Zones A, B, and C for Observations Used in Markov Models

These observations were chosen because they are much of the same criteria that the handwritten rules were based off of. It can be predicted that the goalie would be always be in Zone A. Meanwhile, the active player would be one in Zone One. Other roles could depend on a combination of zones, such as supporting offense, which would probably be a robot both in Zone Two and Zone C at the same time. Defenders could be in Zone B and Zone Two or Three, while the mark can be almost anywhere.

These metrics were chosen since many of the hand-coded rules also relied on these factors. Also, the behavior detection work [HanVel, 1999] used similar metrics leading to good results.

4.2 Markov Models Approach

Markov Models depend on observations and emitted symbols in different situations that serve as a basis of creating a model of states and state transitions. Five different Markov models were constructed in parallel, one for each role. [Vai, 2004] This was computed by taking real role data as training data to calculate the real probability of a transition between states while in each role. There are nine states that were defined based on the distance from ball and position on the field.

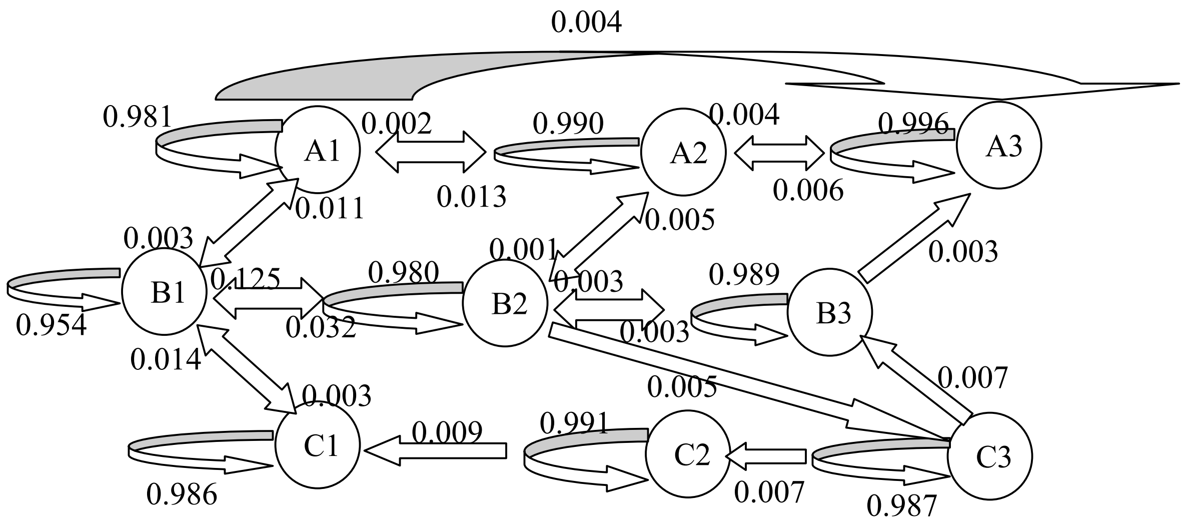

Table 4.1: Markov Model for Goalie

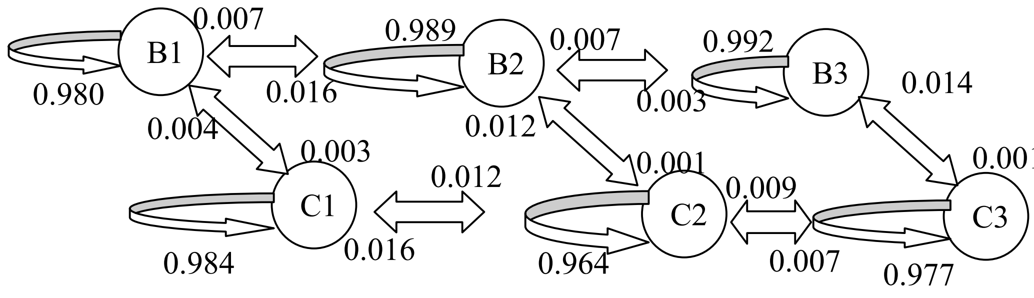

Table 4.2: Markov Model for Active Player

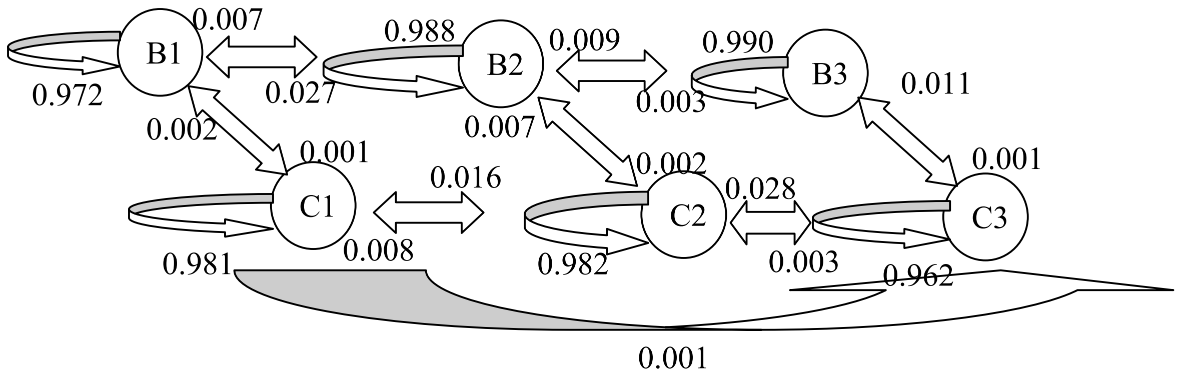

Table 4.3: Markov Model for Defense

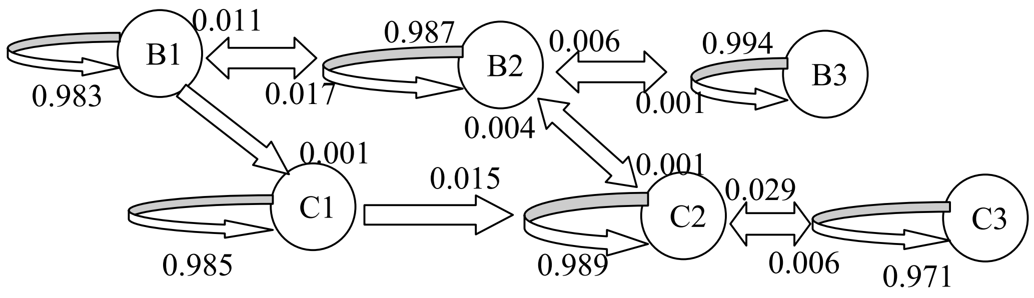

Table 4.4: Markov Model for Supporting Attacker

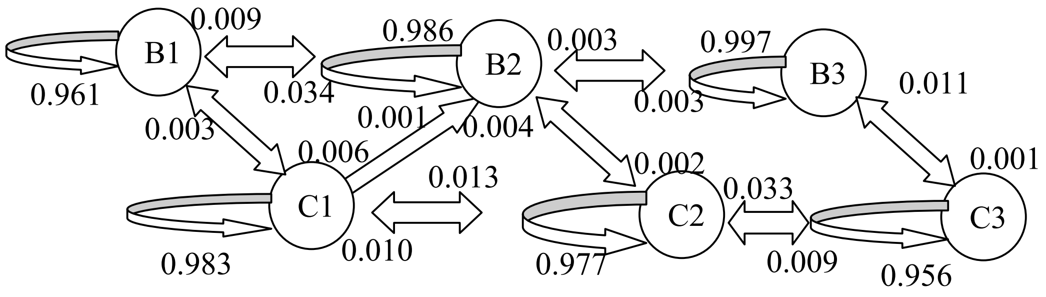

Table 4.5: Markov Model for Mark

After the Markov models are given, a test set of data is run. The test data is a temporal stream of data. Given this time series of values x_0, x_1,...,x_t, the belief state for time j for class c_i where the classes are goalie, active player, defense, supporting offense, and mark, is the probability of the class being the actual role at time j. The math for the belief states comes from [Len, 2004] and was modified slightly to work here.

$$\begin{aligned} B(c_{i,j}) &= P(c_{i,j}|\vec{x}_j, \vec{x}_{j-1}, \ldots, \vec{x}_0) \\ &= \frac{P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_0, c_{i,j}) * P(c_{i,j}|\vec{x}_{j-1}, \ldots, \vec{x}_0)}{P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_0)} \end{aligned}$$

Equation 4.1: Equation for the Belief State

Since the 1/ $P(x_j|x_{j-1},\ldots,x_0)$ term should be constant, it is ignored for now and dealt with later by normalizing all the belief state probabilities. Under Markov assumptions of independence, the first term, $P(x_j|x_{j-1},\ldots,x_0,c_{i,j})$ simplifies to $P(x_j|x_{j-1},c_{i,j})$. Meanwhile the second term, $P(c_{i,j}|x_{j-1},\ldots,x_0)$ can be expanded.

$$\begin{aligned} &P(c_{i,j}|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\ &\quad = \sum_l P(c_{i,j}, c_{l,j-1}|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\ &\quad = \sum_l P(c_{i,j}|c_{l,j-1}, \vec{x}_{j-1} \ldots \vec{x}_0) P(c_{l,j-1}|\vec{x}_{j-1} \ldots \vec{x}_0) \\ &\quad = \sum_l P(c_{i,j}|c_{l,j-1}) * B(c_{l,j-1}) \end{aligned}$$

Equation 4.2: Equation for the Class Probability

Here, $c_{i,j}$ is assumed to be independent of observations before time j given $c_{l,j-1}$ for all l. These assumptions simplify the problem to finding the c_i that maximizes the following equations by providing a recursive solution:

$$\begin{aligned} &B(c_{i,j}) \\ &\quad \propto P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_0, c_{i,j}) * P(c_{i,j}|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\ &\quad \approx P(\vec{x}_j|\vec{x}_{j-1}, c_{i,j}) * P(c_{i,j}|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\ &\quad = P(\vec{x}_j|\vec{x}_{j-1}, c_{i,j}) \sum_l P(c_{i,j}|c_{l,j-1}) B(c_{l,j-1}) \end{aligned}$$

Equation 4.3: Belief State Recursive Solution

The Markov models for each class provides $P(x_j|x_{j-1},c_{i,j})$ by examining the probability of the transition between x_{j-1} and x_j in the model that represents $c_{i,j}$.

$B(c_{i,j})$ was initialized to 0.20 for each value of $c_{i,j}$. Finally, $P(c_{i,j}|c_{l,j-1})$ was defined as 0.98 when $c_{l,j-1}$ was equal to $c_{i,j}$ and was $(1-0.98)/(n-1)$ otherwise, where n is 5 since that is the total number of classes. These values are chosen because it is most common for a self-transition to occur, since roles stay the same for several frames at a time before changing, but it still needs to be possible for a transition to occur. If the role changes for just a few frames, it will be viewed as noise, while after it has changed and stayed steady for multiple frames, it will stabilize again. At every time step, B was normalized to sum to 1 since the $1/P(x_j|x_{j-1},\ldots,x_0)$ term was ignored. This series of equations formulated the Hidden Markov Model placed on top of the set of Markov Models of observations. $B(c_{i,j})$ represented the probability of each state, or class in this case. The role detected was defined as the class with the greatest value of $B(c_{i,j})$.

Since many cells in the Markov models have a high probability despite little data (this happens due to the noise in the system), a Dirichlet Prior with a virtual sample value of 100 was applied on values of $P(x_j|x_{j-1},c_{i,j})$ so data cells based on low information is not taken into account as strongly.

4.3 Experimental Set-up

The experimental set-up for testing Markov Models was the same as the experimental set-up for Hand-coded Rules (Chapter 3, Page 15).

4.4 Results

After the results were collected, a confusion matrix was made again with the most probable role compared to the real role. First it was run with the test data being the same as the training data.

	Goalie	Active Player	Defense	Supporting Offense	Mark
Goalie	**0.897**	0.0380	0.0132	0.0201	0.0321
Active Player	0.0330	**0.335**	0.227	0.182	0.223
Defense	0.0231	0.282	**0.291**	0.180	0.224
Supporting Offense	0.0163	0.289	0.157	**0.348**	0.190
Mark	0.0188	0.294	0.229	0.156	**0.303**

Table 4.6: Markov Models Confusion Matrix when Test Data is the Same as Training Data

While the detection percentages were poor, unlike the hand-coded rules, this method detected the every role to be the correct role more than it detected it to be any other role when the test data was the training data. Also, it has the advantage that it is never "unsure" of what the role is.

The results were not nearly as good once other data was used for the test data.

	Goalie	Active Player	Defense	Supporting Offense	Mark
Goalie	**0.872**	0.0146	0.0305	0.0713	0.0118
Active Player	0.0670	**0.454**	0.314	0.0828	0.0830
Defense	0.0603	**0.467**	0.251	0.116	0.106
Supporting Offense	0.133	0.306	**0.341**	0.166	0.0536
Mark	0.0863	**0.363**	0.300	0.135	0.115

Table 4.7: Markov Models Confusion Matrix

While the results for active player were improved, supporting offense and mark's accuracy of recognition dropped greatly. Meanwhile, defense's accuracy of recognition stays about the same, but was confused with the active player a lot more than before. Either way though, the mark was detected with a higher accuracy rate than the hand-coded rules. It is still not accurate enough, however, since it still is about or below the percentage from a random guess (20%).

This method ended up not being as good as hoped originally for several reasons. First of all, the play system seems to change roles sometimes quicker than the robot can actually move itself to one of the other zones. For instance, there are many situations in the data where the role will have switched three times within a third of a second…not enough time for the metric to be different. For 797 role transitions, there were only 1122 metric transitions in the case of when the test data was the same as the training data. For the other data set, there were 1077 metric transitions for 826 role transitions.

Again, certain play system bugs caused problems. For instance, the mark seems to be confused a lot with the active player and defense, which happens when the mark is marking the opponent's active player. This is particularly detrimental to this method since a robot marking the opponent's active player shows up in Zone One.

Also, it is likely that the zones were too large, especially for position on the field. While the defense can be anywhere on the field, splitting it at the half field point was not necessarily optimal. This is the same for supporting offense. Active player and mark do not seem to rely on this division as much either, although active player has the distance-from-ball metric. In order to create smaller zones, clustering can be used. [RilVel, 2000] Another possible zone that could have been used is the defense triangle from the hand-coded rules.

4.5 Summary

Given that handwritten rules achieved the goal of detecting roles with relative accuracy using positional data, it had seemed that the automated process of Markov models would have yielded better results. At least with the training data, the models were able to choose the right role more often than it chose another specific wrong role, which was not the case with handwritten rules. However, either more work needs to be done for this method to work or another method needs to be used.

Chapter 5

Discussion

The goal of this thesis was to detect roles in real-time. Two methods were tried. While both methods achieved role detection with low latency, they results in varying accuracy. The hand-coded rules was meant to be a starting place and an easy approach. The Markov Models was meant to provide better accuracy.

5.1 Meaning of Results

The results of this thesis show that detecting opponent roles with at least 70% accuracy for almost all roles is possible. The method of choice appears to be the hand-coded rules, which faired a lot better than the Markov models.

With the hand-coded rules, the ability to detect mark is totally non-existent, while the ability to detect defense could be improved on. Meanwhile, the Markov models only seemed to be able to detect the goalie with decent accuracy. It did, however, detect mark with a higher probability than the hand-coded rules.

Other possible approaches that were considered included the use of decision trees and Bayesian classifiers. A decision tree would allow for automation of the process of finding rules, however this has the same problem as the hand-coded rules were the availability of a stream of temporal data is not taken advantage of. While Bayesian classifiers was the method of choice in past work involving detecting plays in American Football, it was not explored in this thesis and maybe should be in the future. It would be interesting to use the same observations for the classifiers as was used with the Markov models.

Given the results in this thesis, it can be concluded that at least with hand-coded rules, most of the roles can be detected with at least 89% accuracy, and that all but one can be detected with 70% accuracy. Although this is still not accurate enough for use in fields such as air traffic control, it certainly shows potential and can at least be partially used for things like automated sport announcing.

5.2 Failure Cases

In the Results section for both methods, several specific failure cases were mentioned. This section is intended to address in more detail each of those cases. In order to understand this section, several symbols must first be introduced.

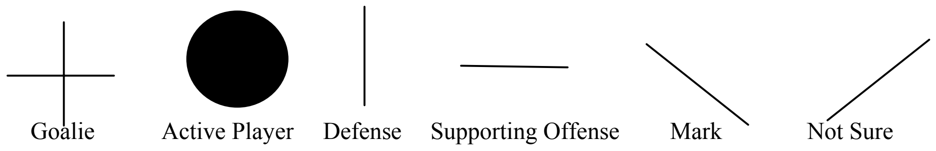

Figure 5.1: Key for Understanding Role Symbols

The actual role is shown in dark blue while the detected role is shown in cyan. There are occasionally other marks that show up on the field as either dark blue or cyan. These marks are to be ignored as they are debug outputs from other parts of the program.

The first situation to be addressed is why mark was so difficult to detect in both methods. The easiest explanation for this is that the mark code in the systems tested is broken to begin with. For some strange reason, the mark tends to chose either the goalie or the active player to mark. The point of mark is to guard an open player from receiving a pass or getting to the ball. The goalie is usually not a target for a pass, and the active player already has the ball. One of the reasons the mark code may be broken is because prior to this thesis work, the software did not know which player is the active player and which player is the goalie, so it does not realize the silliness of what it's doing. Although this goes to show why this work is necessary, it does not help us detect mark very well.

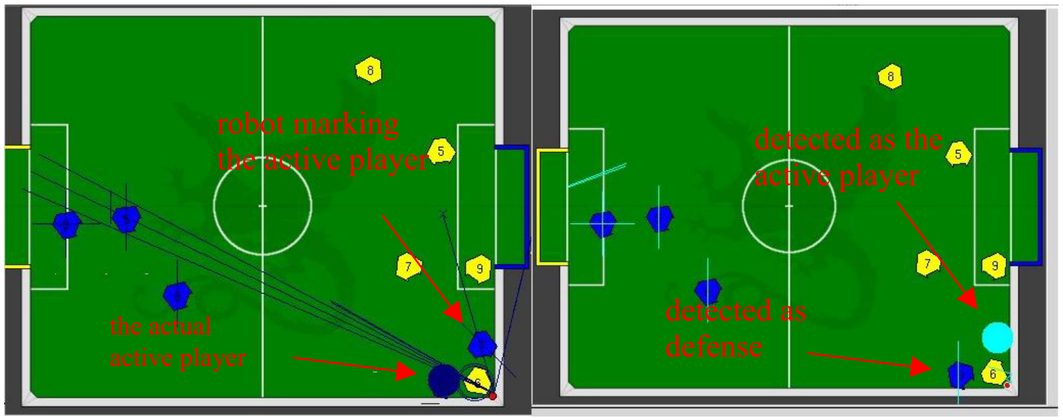
robot marking the active player
the actual active player
detected as the active player
detected as defense

Figure 5.2: A Screenshot showing a Situation Where the Active Player is Marked

When the active player is marked like in the figure above, the software isn't sure which of the two robots is the active player and which isn't. It also assumed that the robot that is not the active player is a support offensive player, not a mark, since a mark should not be marking the active player in order to be a successful mark. Sometimes there is also the same problem with marking the goalie.

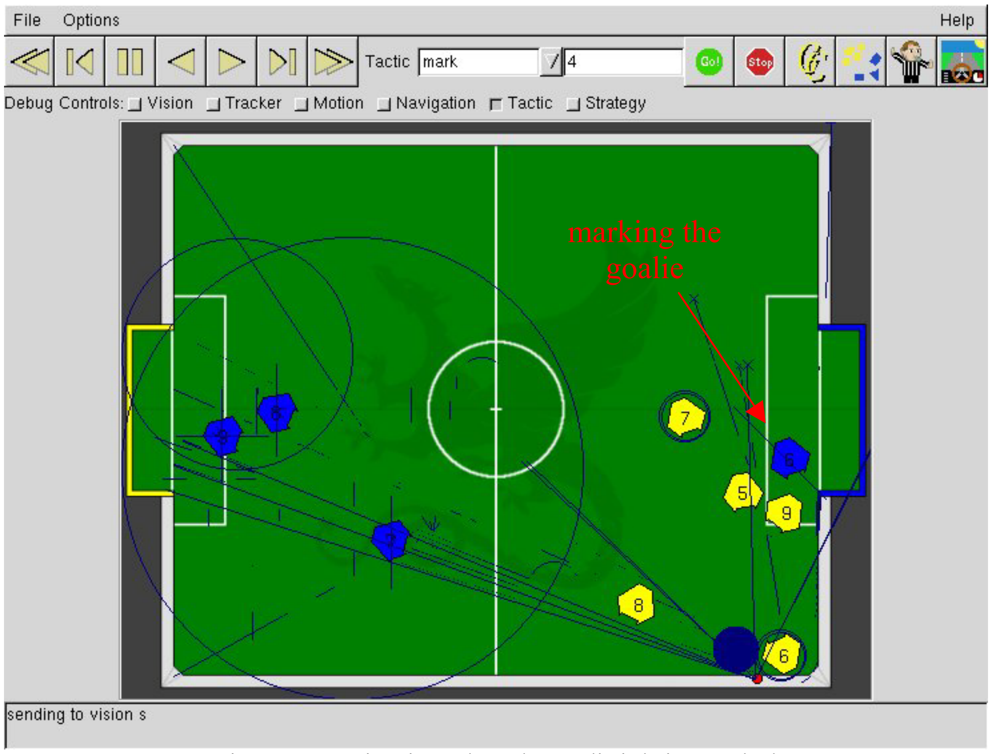

Figure 5.3: A Situation Where the Goalie is being Marked

Another problem situation was for the defense. As mentioned before, defend_line often looks and even acts like a supporting offense role. However, due to the name of the tactic, it is classified under a defensive role, causing some of the confusions between defense and supporting offense.

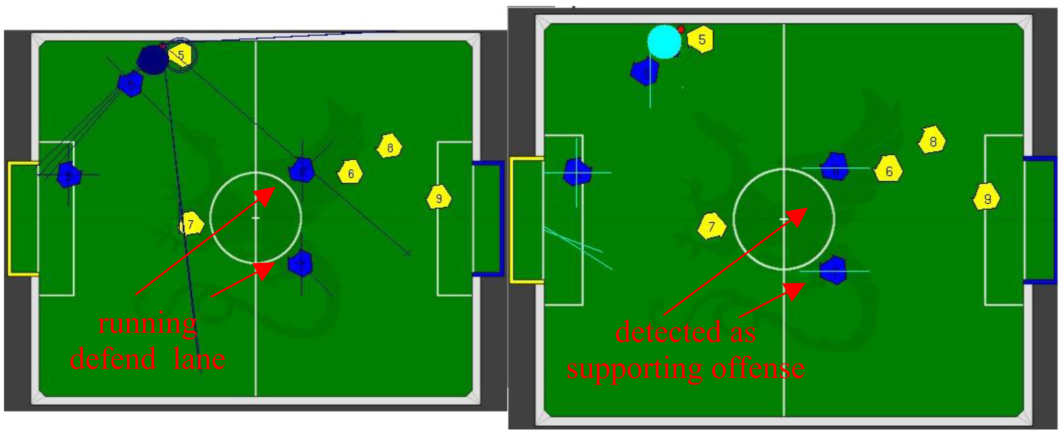

Figure 5.4: An Example Where Defend_Lane Looks/Acts like a Supporting Offensive Role

Defense also seems to have a problem of being detected when the ball is in the corner. Although the triangle-method of detecting defense is fairly effective, it does not cover this case.

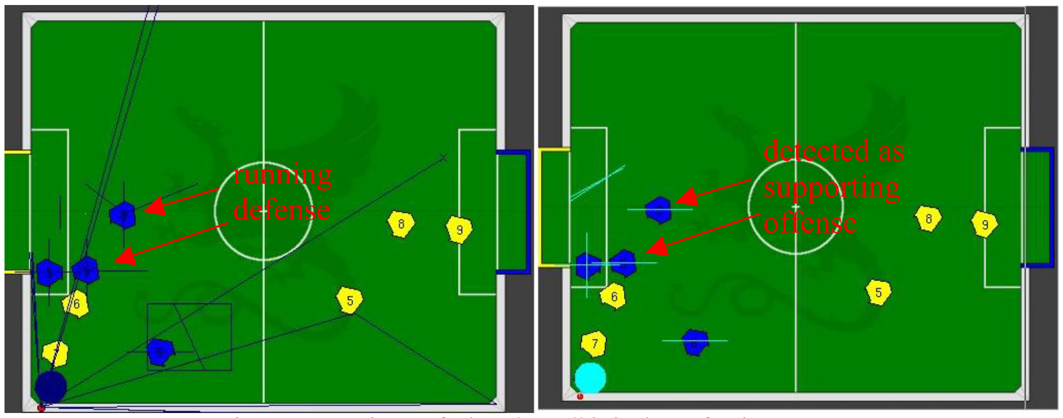

Figure 5.5: A Figure of when the Ball is in the Defensive Corner

Although frequent, these failure cases are still in the minority of all cases. In the "normal" cases, the software still works quite well.

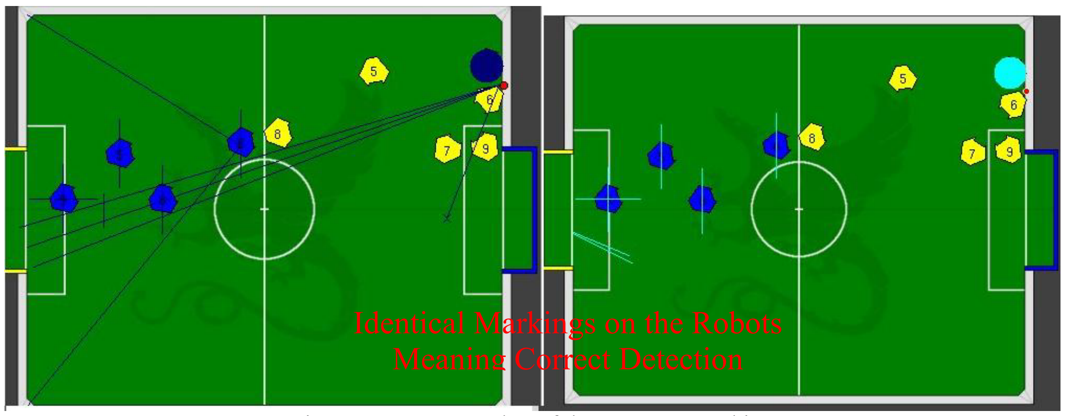

Figure 5.6: A Screenshot of the Program Working

5.3 Future Work

There is a lot of work to be done in improving the accuracy of results. With the handwritten rules method, another clause should be added to detect defense when the ball is in the corner of the field. Also, recognition could be tested on a system with working "mark" code. Both of these things should increase detection to at least 90% per role.

With the information from the role detection, the next step is to use this information to find a way to learn the best way to defeat a team given the roles and positions of the players. This will allow for a more dynamic game which will lead to interesting machine-learning questions about what happens when you have two teams learning at the same time. Also, this can also be applied to see if patterns can be found in flight patterns for air traffic control, formations in military positioning, or real sporting events.

Bibliography

[RilVel, 2002]
Patrick Riley and Manuela Veloso. Planning for Distributed Execution Through Use of Probabilistic Opponent Models. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling (AIPS-2002)*, pp. 72–81, 2002.

[RilVel, 2000]
Patrick Riley and Manuela Veloso. Towards Behavior Classification: A Case Study in Robotic Soccer. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pp. 1092, AAAI Press, 2000.

[Ste, 2001]
Timo Steffans. Feature-based Declarative Opponent-Modeling in Multi-Agent Systems. *Master's Thesis: Institute of Cognitive Science Osnabrűck*, 2001.

[Int, 2001]
Stephen S. Intille. Recognizing Planned, Multiperson Action. *Computer Vision and Image Understanding*, pp. 414-445, 2001.

[HanVel, 1999]
Kwun Han and Manuela Veloso. Automated Robot Behavior Recognition.

[Boh, 2002]
Dan Bohus. Learning to Recognize Automated Robot Behavior in the Soccer Domain.

[Rab, 1989]
Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of IEEE*, Vol. 77, No. 2, pp.257-286, February 1989.

[Robocup]

http://www.robocup.org

[Len, 2004]
Scott Lenser and Manuela Veloso. Classification of Robot Sensor Streams Using Non-Parametric Statistics. Submission 2004.

[BroTry, 2004]
Brett Browning and Eric Tryzelaar. UberSim: A Realistic Simulation Engine for Robot Soccer. *Autonomous Agents and Multi-Agent Systems (AAMAS '03).* Submission 2003.

[Tam, 1996]
Milind Tambe. Tracking Dynamic Team Activity. *National Conference on Artificial Intelligence(AAAI96)*. 1996.

[Vai, 2004]
Discussions with Douglas Vail.