The CMUnited-99 Champion Simulator Team

Peter Stone[1], Patrick Riley[2], and Manuela Veloso[2]

[1]AT&T Labs — Research
180 Park Ave., room A273
Florham Park, NJ 07932
pstone@research.att.com
http://www.research.att.com/~pstone

[2]Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
{pfr,veloso}@cs.cmu.edu
http://www.cs.cmu.edu/{~pfr,~mmv}

Abstract. The CMUnited-99 simulator team became the 1999 RoboCup simulator league champion by winning all 8 of its games, outscoring opponents by a combined score of 110–0. CMUnited-99 builds upon the successful CMUnited-98 implementation, but also improves upon it in many ways. This paper gives a detailed presentation of CMUnited-99's improvements over CMUnited-98.

1 Introduction

The CMUnited robotic soccer project is an ongoing effort concerned with the creation of collaborative and adversarial intelligent agents operating in real-time, dynamic environments. CMUnited teams have been active and successful participants in the international RoboCup (robot soccer world cup) competitions [1, 2, 15]. In particular, the CMUnited-97 simulator team made it to the semi-finals of the first RoboCup competition in Nagoya, Japan [9], the CMUnited-98 simulator team won the second RoboCup competition in Paris, France [13], and the latest CMUnited-99 simulator team won the third RoboCup competition in Stockholm, Sweden[1].

The CMUnited-99 simulator team is modeled closely after its two predecessors. Like CMUnited-97 and CMUnited-98, it uses layered learning [12] and a flexible team structure [11]. In addition, many of the CMUnited-99 agent skills, such as goaltending, dribbling, kicking, and defending, are closely based upon the CMUnited-98 agent skills. However, CMUnited-99 improves upon CMUnited-98 in many ways. This paper focuses on the research innovations that contribute to CMUnited-99's improvements.

Coupled with the publicly-available CMUnited-99 source code [8], this article is designed to help researchers involved in the RoboCup software challenge [3] build upon our success. Throughout the article, we assume that the reader is familiar with the RoboCup simulator, or "soccer server" [5]. A detailed overview of the soccer server, including agent perception and actuator capabilities, is given in [7].

The paper begins by briefly summarizing the main features of the CMUnited-98 simulator team (Section 2). Section 3 describes the improvements in CMUnited-99's

[1] The CMUnited small-robot team is also a two-time RoboCup champion [14, 16].

low-level skills, including the introduction of teammate and opponent modeling capabilities. Section 4 presents the improvements in CMUnited-99's team-level coordination methods. Section 5 focuses on the process by which the low-level skills were improved. Section 6 introduces the concept of layered extrospection, a key advance in our development methodology. Section 7 summarizes CMUnited-99's successful performance at RoboCup-99 and concludes.

2 Background

This section summarizes the features of the CMUnited-97 and CMUnited-98 teams which have been carried over into the CMUnited-99 implementation. Subsequent sections emphasize the research innovations unique to CMUnited-99.

2.1 Main Loop

CMUnited agents are capable of perception, cognition, and action. By perceiving the world, they build a model of its current state. Then, based on a complex set of behaviors, they choose an action appropriate for the current world state.

A driving factor in the design of the agent architecture is the fact that the simulator operates in 100msec cycles. The simulator accepts commands from clients throughout a cycle and then updates the world state all at once at the end of the cycle. Only one action command (dash, kick, or turn) is executed for a given client during a given cycle. Meanwhile, perceptions arrive asynchronously and unpredictably: agents can receive anywhere from zero to six perceptions in a given cycle.

Therefore, CMUnited agents store perceptions as they arrive and then update their internal state from these stored perceptions and the predicted effects of past actions only when it is time to take an action. A detailed description of the agents' world model and how it is updated is given in [13].

2.2 Agent Skills

Once the client has determined the server's world state as accurately as possible, it can choose and send an action to be executed. It can choose from among several low-level skills which provide it with basic capabilities. The output of the skills are primitive movement commands. Examples of low-level skills include kicking, dribbling, ball-interception, goaltending, defending, and clearing. CMUnited-98's versions of these skills are described in detail in [13]. Except where specifically noted in subsequent sections, CMUnited-99's low-level skills are the same.

The common thread among these skills is that they are all *predictive, locally optimal skills* (PLOS). They take into account predicted world states as well as predicted effects of future actions in order to determine the optimal primitive action from a local perspective, both in time and in space. Even though the skills are predictive, the agent *commits* to only one action during each cycle. When the time comes to act again, the situation is completely reevaluated. If the world is close to the anticipated configuration, then the agent will act similarly to the way it predicted on previous cycles. However, if the world is significantly different, the agent will arrive at a new sequence of actions rather than being committed to a previous plan. Again, it will only execute the first step in the new sequence.

2.3 Formations

Given all of the individual skills available to the CMUnited clients, it becomes a significant challenge to coordinate the team so that the players are not all trying to do the same thing at the same time. At the core of the CMUnited-98 coordination mechanism is the *locker-room agreement* [11]. Based on the premise that agents can periodically meet in safe, full-communication environments, the locker-room agreement specifies how they should act when in low-communication, time-critical, adversarial environments.

A good example of the use of the locker-room agreement is CMUnited's ability to execute pre-compiled multi-agent plans after dead-ball situations. While it is often difficult to clear the ball from the defensive zone after goal kicks, CMUnited-98 players move to pre-specified locations and execute a series of passes that successfully move the ball out of their half of the field. Such "set plays" exist in the locker-room agreement for all dead-ball situations.

Another central part of the CMUnited locker-room agreement is the concept of flexible formations consisting of flexible roles. Roles are defined independently of the agents that fill them: homogeneous agents (all except the goalie) can freely switch roles as time progresses. Each role specifies the behavior of the agent filling the role, both in terms of positioning on the field and in terms of the behavior modes that should be considered. For example, an agent filling a forward role will never go into active defense mode. However, agents can switch roles during the course of play.

A formation is a collection of roles, again defined independently from the agents. Just as agents can dynamically switch roles within a formation, the entire team can dynamically switch formations. Formations also include sub-formations, or units, for dealing with issues of local importance. For example, the defensive unit can be concerned with marking opponents while not involving the midfielders or forwards. A player can be a part of more than one unit.

For a detailed presentation of roles, formations, and units, see [11].

2.4 Communication

Communication is another important coordination tool for CMUnited agents. The soccer server provides a challenging communication environment for teams of agents. With a single, low-bandwidth, unreliable communication channel for all 22 agents and limited communication range and capacity, agents must not rely on any particular message reaching any particular teammate. Nonetheless, when a message does get through, it can help distribute information about the state of the world as well as helping to facilitate team coordination.

All CMUnited messages include a certain amount of state information from the speaker's perspective. Information regarding object position and teammate roles are all given along with the confidence values associated with this data. All teammates hearing the message can then use the information to augment their visual state information.

The principle functional uses of communication in CMUnited-98 are

– To ensure that all participants in a set play are ready to execute the multi-step plan. In this case, since the ball is out of play, time is not a critical issue.
– To assign defensive marks. The captain of the defensive unit (the goaltender in most formations) determines which defenders should mark or track which opponent forwards. The captain then communicates this information periodically until receiving a confirmation message.

For a detailed specification of the communication paradigm as it was first developed for CMUnited-97, see [11].

3 Agent Skills

CMUnited-99's basic skills are built mostly on CMUnited-98's skills. This section focusses on CMUnited-99's improvements in low-level skills.

3.1 Ball Velocity Estimation

One of the most important part of good ball handling skills is an accurate estimation of the ball's velocity. When a player is facing the ball, an estimate of the ball's velocity is "visible" via the player's sensory perceptions. However, in both CMUnited-98 and CMUnited-99, when an agent is handling the ball, it uses *position based velocity estimation*. That is, if the agent observes the ball on two successive cycles, it knows the actual path which the ball traveled, and therefore its current velocity. Position based velocity estimation is useful for several reasons:

– Even when trying to watch the ball by turning its neck or body, the agent does not see the ball's velocity every cycle.
– As long as the ball is within 3m of the player, the agent will "feel" the ball, and get position information. It only receives velocity information if the player is facing the ball. If we can rely on just "feel" information, then the can look around for strategic information, such as the location of teammates and opponents.
– Position based velocity estimation allows us to detect unexpected ball movements, which occur most often when an opponent kicks the ball or the ball collides with a player.
– We found experimentally that position based velocity estimation is more accurate than visual info while the ball is within the kickable area.

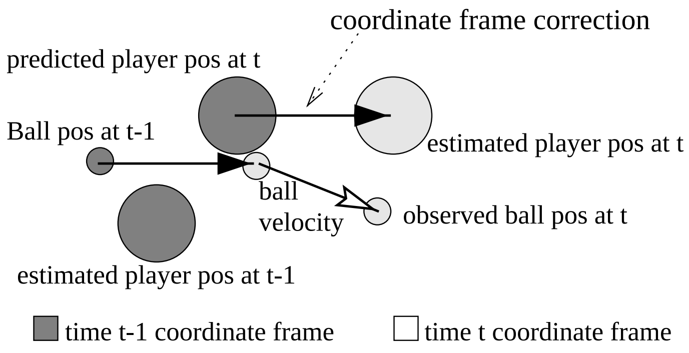

coordinate frame correction
predicted player pos at t
Ball pos at t-1
estimated player pos at t
ball
velocity
observed ball pos at t
estimated player pos at t-1
time t-1 coordinate frame
time t coordinate frame

Fig. 1: Correcting for position error in ball velocity estimation. The dark colored circles represent positions as estimated in the agent's coordinate frame at time $t-1$, and the light colored circles represent positions estimated in the agent's coordinate frame at time t. The difference between the players predicted position at time t and observed (estimated) position can be used to translate one coordinate frame into another.

While this is intuitively a fairly simple idea, there are several complications. First, each agent needs to keep track of the kicks it performed in order to accurately estimate the ball velocity. This is for cases where the agent is not receiving sensations every cycle. The server gives information about kicks that it received, and it is important to note when requested kicks are not executed by the server.

The second is somewhat of an artifact of our world model. Our agents store the current position of all objects in global coordinates by converting the objects' sensed relative positions to global coordinates based on the agent's estimated current position. When they get new visual information, our agents re-estimate their current position. Both of the estimates are quite noisy since they are based on usually distant flags. This means that the ball's old position and new position are in essentially different coordinate frames. In Figure 1, objects in the old coordinate frame are represented in dark grey and objects in the new coordinate frame are lighter. As shown, our agents can calculate the disparity between the coordinate frames by taking the difference of the player's predicted position at time t (judged in the coordinate frame from time $t-1$) and the player's observed position at time t. The ball's position at time $t-1$ can then be moved to the new coordinate frame. The ball velocity is then simply the difference between it's position at time t and position at time $t-1$. This gives a good estimate of the ball's velocity because the only error left is the error in the reported ball positions. When the ball is close to the player, this error is quite small.

3.2 Ball Interception

The basic structure of our ball interception strategy is the same as inCMUnited-98. We successively simulate the ball's positions on future cycles and then determine if the agent can reach a spot that the ball will occupy before the ball does.

Several important improvements were made to this scheme. In CMUnited-98, the agents included an estimate of the current ball position and velocity in every communication message. Sometimes, the agent currently running after the ball would hear such a message from a teammate further away and change its interception strategy. This was problematic for two reasons. First, there is a difference in "coordinate frames" as discussed in Section 3.1. Second, the teammate further away would usually have less accurate information because, in the simulator, visual noise increases with distance. Therefore, the agents in CMUnited-99 will sometimes not listen to heard ball information. If the ball is within the feel distance or if the confidence that the agents have in the ball's position is approximately equal and the communicating teammate is further away, the listener will ignore the information.

A second important change is an emphasis on dashing instead of turning. By careful analysis of the agents' behaviors (see Section 6), we discovered that our agents would frequently make small turns while in pursuit of a moving ball, largely because of noisy information about the ball's velocity. Since the agents plan on doing at most one turn while in pursuit of the ball, this behavior was interfering with the effectiveness of the ball interception skill.

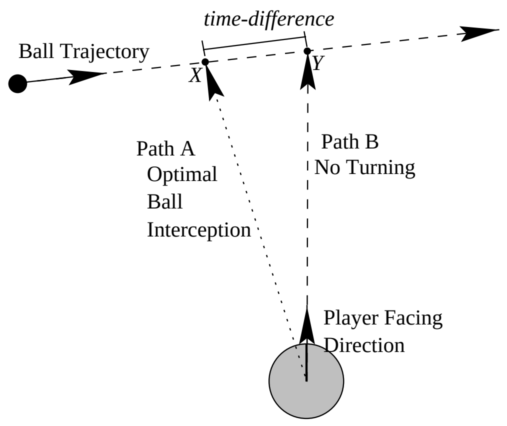


Fig. 2: Deciding whether to turn in ball interception. Point Y is where the agent would intersect the ball if it ran straight without turning first. Point X represents the agent's estimation of the point closest to the ball at which it can intersect the ball's path by turning and then running straight.

To improve this situation, the agents now calculate how long it would take to intercept the ball with no turning at all. This is just a simple ray-ray intersection as shown in Figure 2, path B. *time-difference* is the distance between the calculated optimal (point X) and the path with no turning (point Y), judged by how long it would take the ball to get from X to Y. If *time-difference* is below a threshold of a few cycles, then the agent will proceed along path B instead of path A. Proceeding along path B will always result in a dash instead of a turn.

3.3 Modeling of Opponents and Teammates

In CMUnited-98, decisions about when to shoot were made in one of three ways:

- Based on the distance to the goal
- Based on the number of opponents between the ball and the goal
- Based on a decision tree.

Each of these methods is imperfect. Distance to goal completely ignores how the opponents are positioned. The number of opponents between the ball and the goal does not accurately reflect in how *good* of a position the defenders are. Lastly, the decision tree was trained for passing[7], so its performance on the related but different behavior of shooting is questionable.

CMUnited-99 makes this decision in a more principled way by using a model of an "optimal" goalie. That is, we use a model of a goalie that reacts instantaneously to a kick, moves to exactly the right position to stop the ball, and catches with perfect accuracy.

When deciding whether to shoot, the agent first identifies its best shot target. It generally considers two spots, just inside of the two sides of the goal. The agent then considers the lines from the ball to each of these possible shot targets. *shot-target* is the position whose line is further from the goalie's current position.

The agent then predicts, given a shot at *shot-target*, the ball's position and goalie's reaction using the optimal goalie model. We use the following predicates:

blocking-point The point on the ball's path for which an optimal goalie heads.
ball-to-goalie-cycles The number of cycles for the ball to get to the *blocking-point*
goalie-to-ball-cycles The number of cycles for the goalie to get to the *blocking-point*
shot-margin =*ball-to-goalie-cycles*−*goalie-to-ball-cycles*
better-shot(k) Whether teammate k has a better shot than the agent with the ball, as judged by *shot-margin*

The value *shot-margin* is a measure of the quality of the shot. The smaller the value of *shot-margin*, the more difficult it will be for the goalie to stop the shot. For example, for a long shot, the ball may reach the *blocking-point* in 20 cycles (*ball-to-goalie-cycles*= 20), while the goalie can get there in 5 cycles (*goalie-to-ball-cycles*= 5) This gives a *shot-margin* of 15. This is a much worse shot than if it takes the ball only 12 cycles (*ball-to-goalie-cycles*= 12) and the goalie 10 cycles to reach the *blocking-point* (*goalie-to-ball-cycles*= 10). The latter shot has a *shot-margin* of only 2. Further, if *shot-margin*< 0, then the "optimal" goalie could not reach the ball in time, and the shot should succeed.

Using a model of opponent behavior gives us a more reliable and adaptive way of making the shooting decision. We can also use it to make better passing decisions. When near the goal, the agent may often be faced with the decision about whether to pass or shoot the ball. The agent with the ball simulates the situation where its teammate is controlling the ball, using the goalie model to determine how good of a shot the teammate has. If the teammate has a much better shot, then the predicate *better-shot*(k) will be true. This will tend to make the agent pass the ball, as described in Section 4.1.

There is one complication here; it takes some time to pass the ball. In the time that elapses during a pass, the world will change, and the receiving agent may then decide the original agent has a better shot. This could potentially lead to passing loops where neither agent will shoot. CMUnited-99 does two things to avoid this loop. First, the agent will only pass to a teammate with a better shot if, given the current state of the world, the goalie cannot stop the shot. Secondly, the extra time difference between the passing and receiving agents must be greater than some threshold (5 in CMUnited-99).

Note that this analysis of shooting ignores the presence of defenders. Just because the goalie can not stop the shot (as judged by the optimal goalie model) does not mean that a nearby defender can not run in to kick the ball away.

3.4 Breakaway

An important idea in many team ball sports like soccer is the idea of a "breakaway." Intuitively, this is when some number of offensive players get the ball and themselves past the defenders, leaving only perhaps a goalie preventing them from scoring. After looking at logfiles from previous competitions, we saw many opportunities for breakaways which were not taken advantage of.

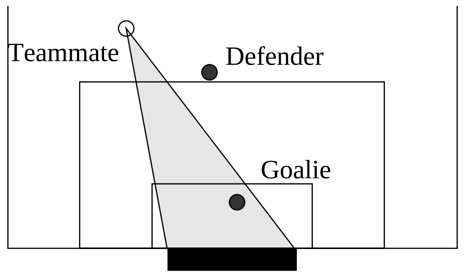


Fig. 3: The Breakaway Cone

The first question which has to be answered is "What exactly is a breakaway?" This is built upon several predicates:

controlling-teammate Which teammate (if any) is currently controlling the ball. "Control" is judged by whether the ball is within the kickable area of a player.

controlling-opponent Which opponent (if any) is currently controlling the ball

opponents-in-breakaway-cone The breakaway cone is shown in Figure 3. The cone has its vertex at the player with the ball and extends to the opponents goal posts.

teammates-in-breakaway-cone The same as the previous definition, but for the other side of the field. This is used when judging whether the opponents currently have a breakaway.

our-breakaway $= (\textit{controlling-teammate} \neq \text{None}) \wedge (\textit{controlling-opponent} = \text{None}) \wedge (\textit{opponents-in-breakaway-cone} \leq 1)$

their-breakaway $= (\textit{controlling-opponent} \neq \text{None}) \wedge (\textit{controlling-teammate} = \text{None}) \wedge (\textit{teammates-in-breakaway-cone} \leq 1)$

Offensive Breakaways. The first new skill we use in breakaways is a generalization of dribbling called "kick and run." When executing the normal dribbling skill, the agent aims to do one kick and then one dash and have the ball end up in its kickable area again. However, this causes a player dribbling the ball to move only half as quickly as a player without the ball since half of its action opportunities are spent kicking rather than moving. Therefore, defenders are able to easily catch up to dribbling players.

For kick and run, the agents aim for one kick and n dashes before being in control of the ball again. In effect, they kick the ball harder to allow them to spend more of their time running. However when doing so, they risk losing the ball if an opponent is able to catch up to the ball before they are back in control of it. Therefore, there is a trade-off involved in the decision of how hard to kick the ball. For use in breakaways, n varies with the proximity of opponents. The closer the opponents are, the smaller n will be.

When to shoot is one of the most important decisions the agents make, especially in breakaways. If the agent shoots too early, the goalie will have plenty of time to stop the ball. If the agent shoots too late, then the goalie may have time to get the ball before the kick is complete.

We use the optimal model described in Section 3.3 to help make this decision. During a breakaway, the agent shoots when either one of the following is true:

- *shot-margin* (defined in Section 3.3) gets below a certain threshold (1 cycle in CMUnited-99)
- The time that it would take for the goalie to proceed directly to the ball and steal it gets below a certain threshold (6 cycles in CMUnited-99).

This skill was extremely effective in the competition, with the vast majority of our goals being scored using the specialized breakaway code.

Defensive Breakaways. Given an effective offense on breakaways, it is a significant challenge to respond effectively on defense. Running straight for the ball is ineffective since the striker keeps moving with the ball. Before trying to get the ball, the defenders must "run past" both the striker and the ball in order to get in better position. This should usually be possible because the striker will be spending time controlling the ball, where the defenders should be running constantly. Therefore, we defined a special "running-past" mode of playing defense. However, problems occur for several reasons:

– It is easy to lose sight of the striker and the ball while running past.
– Noise may cause the decision about whether an opponent has a breakaway to oscillate.
– Ordinarily, the fastest defender to the ball uses regular ball interception and the second fastest runs past the player. However, during a breakaway, these roles switch between the two players often, causing oscillation. Any sort of oscillation is usually bad because the agents will waste time by turning constantly or dashing towards the wrong spot.

We solved these problems in two ways. First, each agent remembers if it was in running-past mode on the last cycle. If it was, then the agent biases its behavior decision towards staying in running-past mode. It also remembers the point for which it was aiming, so that it can continue to run at that point. Otherwise, the target point for which the agent is aiming can oscillate, causing the agent to turn more often, and therefore take longer to get to the correct point. Secondly, when opponents have a breakaway, both the fastest and second fastest player to the ball run past the player, rather than either one going for the ball directly. When the players have run sufficiently past the opponent, they will get into the breakaway cone (see Figure 3) and the opponent will no longer have a breakaway. At that point, the players revert to the normal defensive modes.

4 Coordination

A great deal of CMUnited-99's coordination mechanisms have not changed significantly from those of CMUnited-98. We refer the reader to [13] for information about behavior modes, the locker-room agreement, roles and formations, strategic positioning using attraction and repulsion (SPAR), and communication.

4.1 Ball Handling Decision

One crucial improvement in CMUnited-99 is the agents' decision-making process when in control of the ball. The decisions made at these times are the most crucial in the robotic soccer domain. In general, the agent has the options of

– dribbling the ball in any direction
– passing to any teammate
– shooting the ball
– clearing the ball, or
– simply controlling the ball.

Which it chooses affects the future options of teammates and opponents.

The agent uses a complex heuristic decision mechanism, incorporating a machine learning module, to choose its action. The most significant changes from CMUnited-98 are that the agents use special-purpose code for breakaways (see Section 3.4); that the pass-evaluation decision tree [10] has been retrained during practice games to capture the agents' improved ball-interception ability (see

Section 3.2; that the agents can cross the ball (see below); and that the agents consider whether there is a teammate in a better position than they are to shoot the ball (see Section 3.3).

In presenting the agent decision-making process, we make use of the predicates defined in Section 3 as well as the following:

distance-to-their-goal The distance to the opponent's goal
distance-to-our-goal The distance to own goal
opponents-in-front-of-goal The number of opponents (including the goalie) in the breakaway cone shown in Figure 3
closest-opponent The distance to the closest opponent
closer-to-goal(k) Whether teammate k is closer to the opponent's goal than the agent with the ball
can-shoot Whether *distance-to-their-goal* < 25 and (*opponents-in-front-of-goal* ≤ 1 and *shot-margin* ≤ 6.
can-shoot(k) Same as above but from teammate k's position
congestion $= \sum_{opponents} \frac{1}{(distance-to-opponent)^2})$
congestion(k) Same as above but from teammate k's position
can-dribble-to(x) No defender is nearby or in a cone extending towards the point x

Following is a rough sketch of the decision-making process without all of the parametric details. In all cases, passes are only made to teammates that the decision tree predicts will be able to successfully receive the pass (called *potential receivers* or PR below). If there is more than one potential receiver satisfying the given criteria, then the one predicted with the highest confidence to receive the pass is chosen.

- If $\exists r \in$ PR s.t. *better-shot*(r): pass to r.
- If *our-breakaway*: execute special-purpose breakaway code (see Section 3.4).
- If (*distance-to-their-goal* < 17 and *opponents-in-front-of-goal* ≤ 1) or *shot-margin* ≤ 3: shoot on goal.
- At the other extreme, if *distance-to-our-goal* < 25 or *closest-opponent* < 10: clear the ball (kick it towards a sidelines at midfield and not towards an opponent [13]).
- If $\exists r \in$ PR s.t. *closer-to-goal*(r) and *can-shoot*(r) and *congestion*(r) $\leq$ *congestion*: pass to r.
- If *can-dribble-to*(opponent's goal): dribble towards the goal.
- If $\exists r \in$ PR s.t. *closer-to-goal*(r) and *congestion*(r) $\leq$ *congestion* (even if unable to shoot): pass to r.
- If close to a corner of the field (within a grey or black area in Figure 4) then *cross* the ball as follows.
  - if very near the base line or the corner (in the black area): kick the ball across the field (to "cross target"), even if no teammate is present ("cross it").
  - If able to dribble towards the baseline: dribble towards the baseline (for a later cross).

- If able to dribble towards the corner: dribble towards the corner.
- Otherwise, cross it.

Even though the cross doesn't depend on a teammate being present to receive the ball, we observed many goals scored shortly after crosses due to teammates being able to catch up to the ball and shoot on goal.

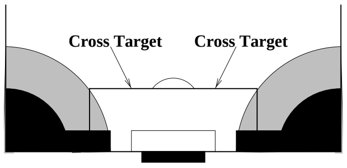


Fig. 4: The "cross" behavior. When within one of the black areas, the agent kicks the ball to the "cross target" on the opposite side of the field. When within one of the grey areas, it tries to dribble to the baseline or the closer corner. If that's not possible due to opponent positions, it kicks the ball to the "cross target" on the opposite side of the field.

– If *can-shoot*: shoot.
– *can-dribble-to*(one of the corner flags): dribble towards the corner flag.
– If approaching the line of the last opponent defender (the offsides line): send the ball (clear) past the defender.
– If $\exists r \in$ PR s.t. *closer-to-goal*(r) or *congestion*$(r) \leq$ *congestion*: pass to r.
– no opponent is nearby: hold the ball (i.e. essentially do nothing and wait for one of the above conditions to fire).
– If $\exists r \in$ PR s.t. no opponent is within 10 or r: pass to r.
– Otherwise: Kick the ball away (clear).

Notice that such a ball-handling strategy can potentially lead to players passing the ball backwards, or away from the opponent's goal. Indeed, we observed such passes several times during the course of games. However, the forward passes and shots are further up in the ball-handling decision, and therefore will generally get executed more often.

4.2 On-Line Coach

Whenever the play stops (due to a foul, an offsides call, or the ball going out of bounds), CMUnited-99 agents execute one of several possible "set-plays" as determined by the locker-room agreement. Which set-play is to be executed and which agent is to fill each set-play role depends on the ball's location. However, in general the players' knowledge of the ball's location is imperfect and may differ from player to player. CMUnited-98 agents used a communication-based negotiation protocol to resolve possible discrepancies among agents. In CMUnited-99, the on-line coach settles the issue by announcing the ball's exact location once the play has stopped. In CMUnited-99, the on-line coach is not used in any other way.

4.3 Deferring Ball Handling

Generally, two defenders go to the ball. If both players are able to kick the ball, problems often result. This happens both because the players may have different ideas of where to kick the ball and because both players assume their kicks get executed (when only one of the two kicks does). To solve this problem, anytime an agent is handling the ball, it considers whether there is a teammate who could also kick the ball this cycle. If there are multiple teammates who can kick the ball, then only the one further upfield will actually send a kick command. When the agents are close enough together to both kick the ball, they will receive position information about each other on every visual sensation. Therefore, the agents will generally have correct ideas about their relative positions, and exactly one will perform a kick.

5 Off-line Training

For the various agent skills described in Section 3 and in [13], there are many parameters affecting the details of the skill execution. For example, in the ball skill of dribbling, there are parameters which affect how quickly the agent dashes, how far ahead it aims the ball, and how opponents affect the location of the ball during dribbling.

The settings for these parameters usually involve a tradeoff, such as speed versus safety, or power versus accuracy. It is important to gain an understanding of what exactly those tradeoffs are before "correct" parameter settings can be made.

We created a trainer client that connects to the server as an omniscient off-line coach client (this is separate from the on-line coach). The trainer is responsible for three things:

1. **Repeatedly setting up a particular training scenario**. In the dribbling skill, for example, the trainer would repeatedly put a single agent and the ball at a particular spot. The agent would then try to dribble the ball to a fixed target point.
2. **Recording the performance of the agent on the task**. Here we use a hand-coded performance metrics, generally with very simple intuitive ideas. In the kicking skill, for example, we record how quickly the ball is moving, how accurate the kicking direction is, and how long it took to kick the ball.
3. **Iterating through different parameter settings**. Using the server's communication mechanism, the trainer can instruct the client on which parameter settings to use. The trainer records the performance of the agent for each set of parameter values.

Once the scenario is set up, the system runs autonomously. Since most skills only involve one or two clients, we could afford to have the trainer iterate over many possible parameter values, taking several hours or days.

Once the trainer has gathered the data, we would depict the results graphically and decide which parameters to use. An example for the hard kicking skill

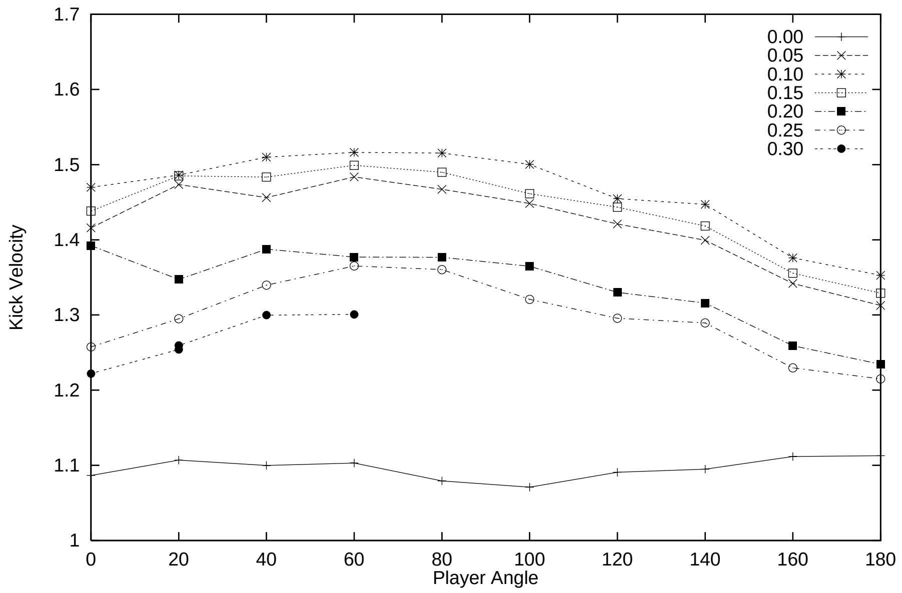


Fig. 5: An example of training data. Two parameters are varied here: the player angle (displayed on the x-axis) and the buffer around the player out of which the agent tries to keep the ball (the different lines). The goal is to maximize the kick velocity (displayed on the y-axis).

is shown in Figure 5. The two parameters varied for the test shown are the angle the agent is facing relative to the kicking angle (the x-axis), and the buffer around the player out of which the agent tries to keep the ball (the different lines).

Sometimes, the "optimal" parameter selection was fairly clear. For example, in Figure 5, we are trying to maximize the kick velocity. Therefore, we would select a player angle of approxiamtely 60 degrees and a buffer of 0.10. Other times, the data looked much noisier. In those cases we could narrow our search down somewhat and get more data over the relevant parts of the parameter space.

We were sometimes limited by processing power in the breadth or resolution of the parameter space that we could examine. A more adaptive searching strategy, such as might be given by various learning techniques like genetic programming [4], would be a useful addition.

6 Layered Extrospection

A perennial challenge in creating and using complex autonomous agents is following their choice of actions as the world changes dynamically, and understanding why they act as they do. In complex scenarios, even the human computer-agent developer is often unable to identify what exactly caused an agent to act as it did in a given situation. Adding support for human developers and observes to better follow and understand the actions of autonomous agents can be a significant enhancement to processes of development and use of agents.

To this end, we introduce the concept of *layered extrospection*[2] by which autonomous agents include in their architecture the foundations necessary to allow a person to probe into the specific reasons for an agent's action. This probing may be done at any level of detail, and either retroactively or while the agent is acting.

A key component of layered extrospection is that the relevant agent information is organized in *layers*. In general, there is far too much information available to display all of it at all times. The imposed hierarchy allows the user to select at which level of detail he or she would like to probe into the agent in question.

Layered extrospection has two main uses:

1. As a debugging tool for agent development in complex environments.
2. As a vehicle for interactive agent control.

When an agent does something unexpected or undesirable, it is particularly useful to be able to isolate precisely *why* it took such an action. Using layered extrospection, a developer can probe inside the agent at any level of detail to determine precisely what needs to be altered in order to attain the desired agent behavior. For example, if a software agent is left to perform some action over night, but fails to complete its assigned task, the developer could use layered extrospection to trace the precise reasons that the agent took each of its actions, and identify which parts of the agent need to be altered.

The fact that layered extrospection also works in real time means that a user can monitor the internals of an agent as it acts. When coupled with an interface for influencing agent behaviors, layered extrospection could be used to allow the interleaving of autonomous and manual control of agents. For example, given a set of robots autonomously cleaning the floors in a large building, a person might want to monitor agents' perceptions of which floors are in most urgent need of attention. Upon noticing that one of the agents is ignoring an area that is particularly dirty, the person could determine the cause of the agent's oversight and manually alter the internal state or goal stack within the agent in real time.

Layered extrospection was a significant part of the development of CMUnited-99, and led to many of the improvements in the team over CMUnited-98. Our development of layered extrospection was inspired in part by our own inability to trace the reasons behind the actions of CMUnited-98. For example, whenever a player kicks the ball towards its own goal, we would wonder whether the agent was mistaken about its own location in the world, whether it was mistaken about the ball's or other agents' locations, or if it "meant" to kick the ball where it did, and why. Due to the dynamic, uncertain nature of the environment, it is usually

[2] We use the term "extrospection" in order to evoke the connotations of "introspection," a reflective looking inwards to oneself. In contrast, extrospection is meant to connote a reflective looking inwards to the agent.

impossible to recreate the situation exactly in order to retroactively figure out what happened.

Our layered extrospection implementation is publicly available[8]. It can be easily adapted for use with other RoboCup simulator teams. It works as follows.

During the course of a game, our agents store detailed records of selected information in their perceived world states, their determination of their short-term goals, and their selections of which actions will achieve these goals, along with any relevant intermediate decisions that lead to their action selections.

After the game is over, it can be replayed using the standard "logplayer" program which comes with the soccer server. Our extrospection module, implemented as an extension to this logplayer, makes it possible to inspect the details of an individual player's decision-making process at any point. Figure 6 shows our robotic soccer layered extrospection interface.

The log file of the player being examined in Figure 6 contains the following lines. Notice that in Figure 6, only the lines at level 20 and below are displayed.

```
881 (35) My Pos: (15.85, -3.73)   angle: -24.00
881 (35) Ball Pos: (17.99, -4.87)conf: 0.95
881 (45) Sight at 880: Bv team:_  opp: 1
881 (5) Mode: AM_Offense_Active (I'm fastest to ball)
881 (15) get_ball: going to the moving ball (5) pow 100.0
881 (25) go_to_point: dash 100.0
881 (20) go_to_point 3 (21.1 -6.2)
881 (30) dashing 100.0
881 (45) Heard message of type 1 from 11
```

In the remainder of this section we provide two examples illustrating the usefulness of layered extrospection.

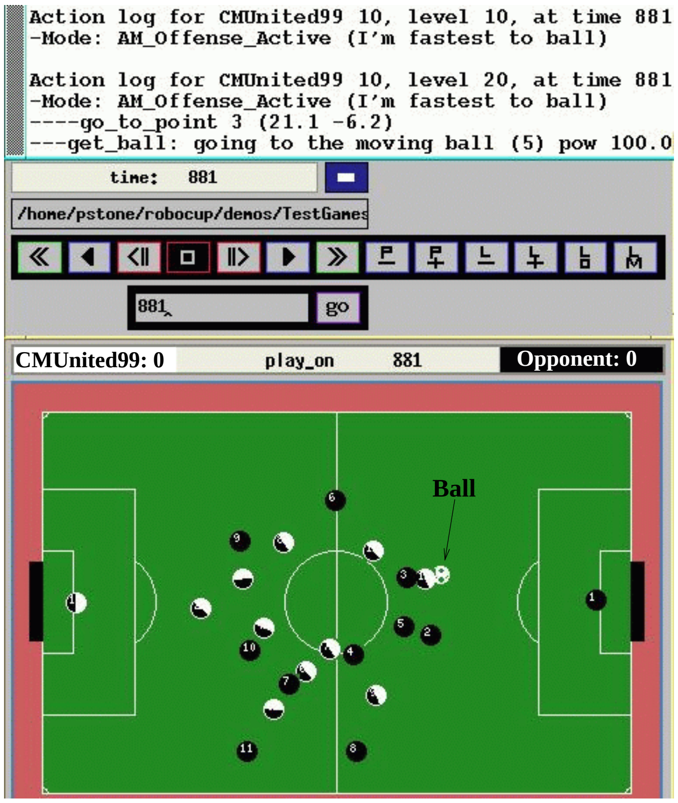
6.1 Discovering Agent Beliefs

When observing an agent team performing, it is tempting, especially for a person familiar with the agents' architectures, to infer high level beliefs and intentions from the observed actions. Sometimes, this can be helpful to describe the events in the world, but misinterpretation is a significant danger.

Consider the example in Figure 7. Here, two defenders seem to pass the ball back and forth while quite close to their own goal. In general, this sort of passing back and forth in a short time span is undesirable, and it is exceptionally dangerous near the agents' own goal. Using the layered extrospection tool, we get the information displayed in Figure 8. Note that each dash '-' represents 5 levels.

First, we see that in both cases that player number 2 was in control of the ball (time 831 and 845), it was trying to clear it (just kick it away from the goal), not pass to player number 5. Given the proximity of the goal and opponents, clearing is a reasonable behavior here. If a teammate happens to intercept a clear, then our team is still in control of the ball. Therefore, we conclude that this agent's behavior matches what we want and expect.

Next, we can see that player number 5 was trying to dribble towards the opponent's goal in both cases that he controlled the ball (time 838 and 850).



```
Action log for CMUnited99 10, level 10, at time 881
-Mode: AM_Offense_Active (I'm fastest to ball)

Action log for CMUnited99 10, level 20, at time 881
-Mode: AM_Offense_Active (I'm fastest to ball)
----go_to_point 3 (21.1 -6.2)
---get_ball: going to the moving ball (5) pow 100.0
```

Fig. 6: The hierarchical extrospection tool. The terminal window at the top is displaying the high-level information for the agent with the ball (number 10 on the light-colored team) at the instant shown in the graphical display of the game. At level 10, only the agent's active mode ("offense active") is shown. At level 20, information about its high-level actions are also included. In this case, the agent is trying to intercept the moving ball at a specific point. Details about how this point was arrived at, as well as the resulting low-level action, the agent's most recent sensory perceptions, and the agent's current world state can be displayed by moving to a lower hierarchical behavior layer. On the control bar in the middle, the buttons labeled with a "P" are used to control which agent is being examined. The buttons labeled "L" are used to control at what layer it is being examined ("L/0" and "L/M" are shortcuts for the lowest and highest layers respectively).

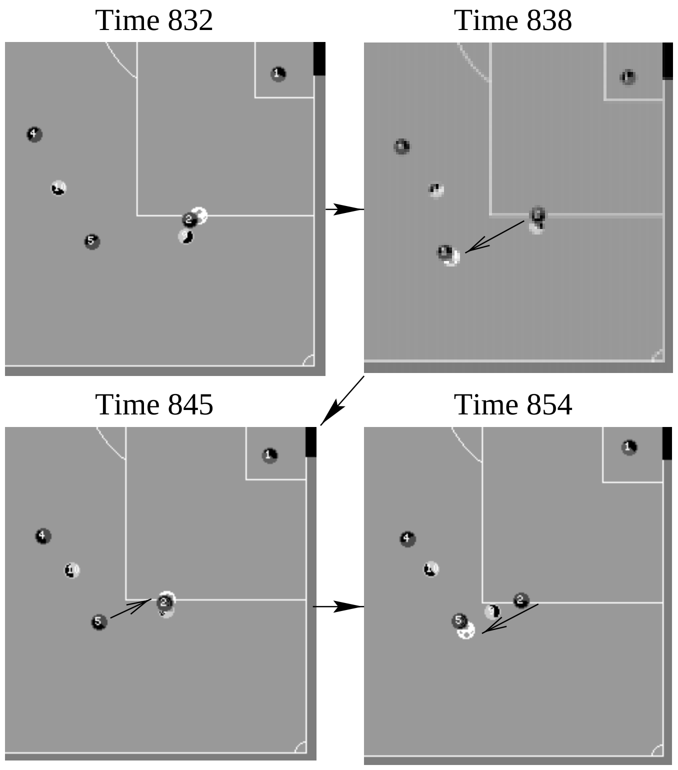


Fig. 7: Undesired passing behavior. Two defenders pass the ball back and forth while very near to their own goal. The direction of the passes is indcated by arrows.

```
Action log for CMUnited99 2, level 30, at time 831
-Mode: AM_With_Ball
--Handling the ball
----OurBreakaway() == 0
---handle_ball: need to clear
---clear_ball:  target ang == -93.0
-----starting kick to angle -93.0,  translated to point (-6.4, -34.0)
```

```
Action log for CMUnited99 5, level 50, at time 838
-------Invalidating ball vel:0.36 > 0.36, thought vel was (1.73, -0.70)
-------Position based velocity estimating:
       gpos (-32.1 -23.4), prev_seen_pos (-33.5 -23.1)
---------Sight 838.0: B_ team:___________opp:________9__
-Mode: AM_With_Ball
--Handling the ball
----OurBreakaway() == 0
-----CanDribbleTo (-22.05, -20.52): TRUE No players in cone
---handle_ball: dribbling to goal (2) -- have room
```

```
Action log for CMUnited99 2, level 20, at time 845
-Mode: AM_With_Ball
--Handling the ball
---handle_ball: need to clear
---clear_ball:  target ang == 24.0
----starting kick to angle 24.0, translated to point (-16.5, -34.0)
----kick_ball: starting kick to angle 24.0
```

```
Action log for CMUnited99 5, level 20, at time 850
-Mode: AM_With_Ball
--Handling the ball
---handle_ball: dribbling to goal (2) -- have room
```

Fig. 8: Extrospection information for the passing example (the boxes have been added for emphasis)

There are no opponents immediately around him, and the path on the way to the goal is clear. This agent's intention is certainly reasonable.

However, at time 838, player number 5 does not perform as it intended. Rather than dribbling forward with the ball, it kicked the ball backwards. This points to some problem with the dribbling behavior. As we go down in the layers, we see that the agent invalidated the ball's velocity. This means that it thought the ball's observed position was so far off of its predicted position that the agent's estimate for the ball's velocity could not possibly be right. The agent then computed a new estimate for the ball's velocity based on its past and current positions.

Given this estimation of the ball's velocity (which is crucial for accurate ball handling), we are led to look further into how this velocity is estimated. Also, we can compare the estimate of the velocity to the recorded world state. In the end, we find that the ball collided with the player. Therefore, it was invalid to estimate the ball's velocity based on position. In fact, this led us to more careful application of this velocity estimation technique.

In this case, inferring the intentions of the players was extremely challenging given their behaviors. Without extrospection, the natural place to look to correct this undesirable behavior would have been in the passing decisions of the players. It would have been difficult or impossible to determine that the problem was with the estimation of the ball's velocity.

6.2 The Use of Layers

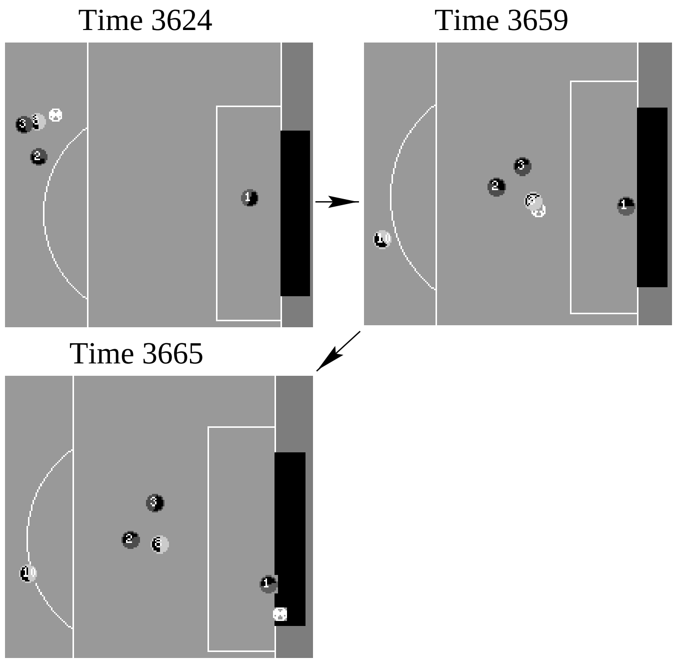


Fig. 9: Poorly performing defenders. Two defenders start close to an opponent who is about to receive the ball. However, they are unable to steal the ball and in fact do not catch up to the opponent significantly.

The fact that the agents' recordings are layered is quite important. One important effect is that the layers allow the observer to look at just higher levels, then explore each case more deeply as required.

Consider the example depicted in Figure 9. Here, two defenders are unable to catch up and stop one offensive player with the ball, even though the defenders were in a good position to begin with.

Since this is a scenario that unfolds over many time steps, we need to be able to understand what happens over that time sequence. The first pass at this

Time	Player 2	Player 3
3624	-Defense_Active(poss=?)	***-Offense_Active***
3625	-Defense_Active(poss=l)	***-Offense_Active***
3626	-Defense_Active(poss=?)	***-Offense_Active***
3627	-Defense_Active(poss=l)	***-Offense_Active***
3628	***-Offense_Active***	-Defense_Active(poss=?)
3629	***-Offense_Active***	***-Offense_Active***
3630	***-Offense_Active***	-Defense_Active(poss=l)
3631	***-Offense_Active***	-Defense_Active(poss=l)
3632	***-Offense_Active***	-Defense_Active(poss=l)
3633	***-Offense_Active***	-Defense_Active(poss=l)
3634	***-Offense_Active***	-Defense_Active(poss=l)
3635	***-Offense_Active***	-Defense_Active(poss=l)
3636	-Defense_Active(poss=l)	***-Offense_Active***
3637	-Defense_Active(poss=l)	***-Offense_Active***
3638	-Defense_Active(poss=l)	***-Offense_Active***
3639	***-Offense_Active***	-Defense_Active(poss=l)
3640	***-Offense_Active***	-Defense_Active(poss=?)
3641	-Defense_Active(poss=l)	-Defense_Active(poss=l)
3642	-Defense_Active(poss=?)	-Defense_Active(poss=l)
3643	***-Offense_Active***	***-Offense_Active***
3644	***-Offense_Active***	-Defense_Active(poss=l)
3645	***-Offense_Active***	-Defense_Active(poss=l)
3646	-Defense_Active(poss=l)	-Defense_Active(poss=l)
3647	***-Offense_Active***	***-Offense_Active***
3648	-Defense_Active(poss=?)	***-Offense_Active***
3649	***-Offense_Active***	***-Offense_Active***
3650	-Defense_Active(poss=l)	***-Offense_Active***
3651	***-Offense_Active***	-Defense_Active(poss=l)
3652	***-Offense_Active***	-Defense_Active(poss=l)
3653	***-Offense_Active***	-Defense_Active(poss=?)
3654	***-Offense_Active***	-Defense_Active(poss=?)
3655	***-Offense_Active***	-Defense_Active(poss=?)
3656	***-Offense_Active***	-Defense_Active(poss=?)
3657	***-Offense_Active***	***-Offense_Active***
3658	***-Offense_Active***	***-Offense_Active***
3659	***-Offense_Active***	***-Offense_Active***

Fig. 10: Extrospection information for the defending example (the bold italics have been added for emphasis)

is to just look at the highest level decision. The first decision our agents make is in which "action mode" they are [13]. This decision is based on which team is controlling the ball, current location, role in the team structure, etc. Usually, the player fastest to the ball will be in "Offense_Active" mode, meaning they will try to get to the ball as fast as possible. In a defensive situation, the second fastest player will be in "Defense_Active" mode, which means basically to get in the way of the player with the ball, without actively trying to steal it.

The output of just the highest level from the extrospection tool is depicted in Figure 10. There are two things to notice. First, the agents change roles many times over this sequence. Secondly, the agents' are often unsure about which side is in control of the ball (the 'poss=' field).

This constant changing of mode causes a problem for the agents. "Offense_Active" and "Defense_Active" modes tell the players to move to different spots on the field. By switching back and forth, the agents will waste a great deal of time turning to face the direction they want to go instead of actually going. Therefore, the agents do not catch up.

Noticing the 'poss=' field is also in flux allows further diagnosis of the problem. The decision about what mode to go into is sometimes affected by which team the agent believes is controlling the ball. Realizing that this value is often unknown should lead to changes in the way that value is determined, or changes in the manner in which it is used.

In this case, making use of layered extrospection to examine just the high-level reasoning decisions of a pair of agents allows us to focus on a problem that would have otherwise been easily overlooked.

We envision that layered extrospection will continue to be useful in the RoboCup simulator and in other agent development projects, particularly those with complex agents acting in complex, dynamic environments. We also plan to begin using layered extrospection in interactive semi-autonomous agent-control scenarios.

7 Results and Conclusion

The third international RoboCup championship, RoboCup-99, was held on July 28–August 4, 1999 in Stockholm, Sweden in conjunction with the IJCAI-99 conference [15]. As the defending champion team, the CMUnited-98 simulator team was entered in the competition. Its code was left unaltered from that used at RoboCup-98 except for minor changes necessary to update to version 5 of the soccer server. Server parameter changes that reduced player size, speed, and kickable area required adjustments in the CMUnited-98 code. However CMUnited-98 did not take advantage of additions to the players' capabilities such as the ability to look in a direction other than straight ahead (simulation of a neck).

The CMUnited-98 team became publicly available soon after RoboCup-98 so that other people could build upon our research. Thus, we expected there to be several teams at RoboCup-99 that could beat CMUnited-98, and indeed there were. Nonetheless, CMUnited-98 performed respectably, winning 3 games, losing 2, and tying 1 and outscoring its opponents by a combined score of 29–3. Table 1 presents the details of CMUnited-98's matches.

Meanwhile, the CMUnited-99 team was even more successful at the RoboCup-99 competition than was its predecessor at RoboCup-98. It won all 8 of its games by a combined score of 110–0, finishing 1st in a field of 37 teams. Table 2 shows CMUnited-99's game results.

Opponent	Affiliation	Score (CMU–Opp.)
Kasuga-Bitos III	Chubu University, Japan	19 – 0
Karlsruhe Brainstormers	University of Karlsruhe, Germany	1 – 0
Cyberoos	CSIRO, Australia	1 – 0
Essex Wizards	University of Essex, UK	0 – 0
Mainz Rolling Brains	University of Mainz, Germany	0 – 2
Gemini	Tokyo Institute of Technology, Japan	8 – 0
11 Monkeys	Keio University, Japan	0 – 1
TOTAL		29 – 3

Table 1: The scores of CMUnited-98's games in the simulator league of RoboCup-99. CMUnited-98 won 3, lost 2, and tied 1 game.

Note 1. The last game was lost by one goal in overtime.

Opponent	Affiliation	Score (CMU–Opp.)
Ulm Sparrows	University of Ulm, Germany	29 – 0
Zeng99	Fukui University, Japan	11 – 0
Headless Chickens III	Linköping University, Sweden	17 – 0
Oulu99	University of Oulu, Finland	25 – 0
11 Monkeys	Keio University, Japan	8 – 0
Mainz Rolling Brains	University of Mainz, Germany	9 – 0
Magma Freiburg	Freiburg University, Germany	7 – 0
Magma Freiburg	Freiburg University, Germany	4 – 0
TOTAL		110 – 0

Table 2: The scores of CMUnited-99's games in the simulator league of RoboCup-99. CMUnited-99 won all 8 games, finishing in 1st place out of 37 teams.

Qualitatively, there were other significant differences between CMUnited-98's and CMUnited-99's performances. In RoboCup-98, several of CMUnited-98's matches were quite close, with many offensive and defensive sequences for both teams. CMUnited-98's goalie performed quite well, stopping many shots. In RoboCup-99, CMUnited-99's goalie only had to touch the ball three times over all 8 games. Only two teams (Zeng99 and Mainz Rolling Brains) were able to create enough of an offense in order to get shots on our goal. Improvements in ball velocity estimation (Section 3.1), ball interception (Section 3.2), and a myriad of small improvements made possible by layered extrospection (Section 6) greatly improved CMUnited-99's midfield play over CMUnited-98.

Another qualitative accomplishment of CMUnited-99 was how closely its actions matched our ideas of what should be done. When watching games progress, we would often just be starting to say "Pass the ball!" or "Shoot it" when the agents would do exactly that. While this is certainly not a solid criterion on which to judge a team in general, it is a testament to our development techniques that we were able to refine behaviors in such a complex domain to match our high level expectations.

There are certainly many improvements to be made. For example, in CMUnited-99's game against Zeng99, our breakaway behavior (Section 3.4) was much less effective in general. This was because the Zeng99 team put an extra defender behind the goalie. CMUnited-99's agents assumed the defender closest to the goal was the goalie. Therefore, the agents applied the goalie model to that defender instead of to the real goalie. This allowed the real goalie to stop many shots which our agents did not anticipate could be stopped. Creating models of other opponents and using them more intelligently could improve this behavior.

Further, adapting models to opponents during play, as well as changing team strategy is a promising future direction. We have done some experimentation with approaches to quick adaptation in complex domains like robotic soccer[6]. Other researchers associated with RoboCup are also looking in this direction, especially with the newly introduced coach agent.

Various software from the team is available [8]. The binaries for the player and coach agents are available. Full source code for the coach agent, the trainer agent, and the layered extrospection tool are also available. Further, skeleton source code for the player agents, including the low level skills, is also available.

References

1. Minoru Asada and Hiroaki Kitano, editors. *RoboCup-98: Robot Soccer World Cup II.* Springer Verlag, Berlin, 1999.
2. Hiroaki Kitano, editor. *RoboCup-97: Robot Soccer World Cup I.* Springer Verlag, Berlin, 1998.
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 97. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 24–29, San Francisco, CA, 1997. Morgan Kaufmann.
4. John R. Koza. *Genetic Programming.* MIT Press, 1992.
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.
6. Patrick Riley and Manuela Veloso. On behavior classification in adversarial environments. (submitted to Autonomous Agents 2000), 1999.
7. Peter Stone. *Layered Learning in Multi-Agent Systems.* PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, December 1998. Available as technical report CMU-CS-98-187.
8. Peter Stone, Patrick Riley, and Manuela Veloso. CMUnited-99 source code, 1999. Accessible from http://www.cs.cmu.edu/~pstone/RoboCup/CMUnited98-sim.html.
9. Peter Stone and Manuela Veloso. The CMUnited-97 simulator team. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I.* Springer Verlag, Berlin, 1998.
10. Peter Stone and Manuela Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.

11. Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.
12. Peter Stone and Manuela Veloso. Layered learning and flexible teamwork in robocup simulation agents. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, Berlin, 2000. Springer Verlag.
13. 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*. Springer Verlag, Berlin, 1999.
14. Manuela Veloso, Michael Bowling, Sorin Achim, Kwun Han, and Peter Stone. The CMUnited-98 champion small robot team. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. Springer Verlag, Berlin, 1999.
15. Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors. *RoboCup-99: Robot Soccer World Cup III*. Springer Verlag, Berlin, 2000. To appear.
16. Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim. The CMUnited-97 small-robot team. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 242–256. Springer Verlag, Berlin, 1998.