The CMUnited-98 Champion Simulator Team *

Peter Stone, Manuela Veloso, and Patrick Riley

Computer Science Department,Carnegie Mellon University
Pittsburgh, PA 15213
{pstone,veloso}@cs.cmu.edu, priley@andrew.cmu.edu
http://www.cs.cmu.edu/{~pstone,~mmv}, http://www.andrew.cmu.edu/~priley

A shortened version of this paper appears in
"RoboCup-98: Robot Soccer World Cup II." M. Asada and H. Kitano eds.
Springer Verlag, Berlin, 1999.

Abstract. The CMUnited-98 simulator team became the 1998 RoboCup simulator league champion by winning all 8 of its games, outscoring opponents by a total of 66–0. CMUnited-98 builds upon the successful CMUnited-97 implementation, but also improves upon it in many ways. This article describes the complete CMUnited-98 software, emphasizing the recent improvements. Coupled with the publicly-available CMUnited-98 source code, it is designed to help other RoboCup and multi-agent systems researchers build upon our success.

1 Introduction

The CMUnited-98 simulator team became the 1998 RoboCup [4] simulator league champion by winning all 8 of its games, outscoring opponents by a total of 66–0. CMUnited-98 builds upon the successful CMUnited-97 implementation [8], but also improves upon it in many ways.

The most notable improvements are the individual agent skills and the strategic agent positioning in anticipation of passes from teammates. While the success of CMUnited-98 also depended on our previous research innovations including layered learning [9], a flexible teamwork structure [10], and a novel communication paradigm [10], these techniques are all described elsewhere. The purpose of this article is to clearly and fully describe the low-level CMUnited-98 agent architecture as well as the key improvements over the previous implementation.

Coupled with the publicly-available CMUnited-98 source code [11], this article is designed to help researchers involved in the RoboCup software challenge [5] build upon our success. Throughout the article, we assume that the reader is familiar with the soccer server [1].

* This research is sponsored in part by the DARPA/RL Knowledge Based Planning and Scheduling Initiative under grant number F30602-95-1-0018. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the U. S. Government.

The rest of the article is organized as follows. Section 2 gives an overview of the entire agent architecture. Section 3 describes the agents' method of keeping an accurate and precise world model. Section 4 details the low-level skills available to the agents. Section 5 presents the CMUnited-98 collaborative coordination mechanisms. Section 6 summarizes the RoboCup-98 results and Section 7 concludes.

2 Agent Architecture Overview

CMUnited-98 agents are capable of perception, cognition, and action. By perceiving the world, they build a model of its current state. Then, based on a 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 fixed cycles of length 100 msec. As presented in Section [1], 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, turn, or catch) is executed for a given client during a given cycle.

Therefore, agents (simulator clients) should send exactly one action command to the simulator in every simulator cycle. If more than one command is sent in the same cycle, a random one is executed, possibly leading to undesired behavior. If no command is sent during a simulator cycle, an action opportunity has been lost: opponent agents who have acted during that cycle may gain an advantage.

In addition, since the simulator updates the world at the end of every cycle, it is advantageous to try to determine the state of the world at the end of the previous cycle when choosing an action for the current cycle. As such, the basic agent loop during a given cycle t is as follows:

- Assume the agent has consistent information about the state of the world at the end of cycle $t-2$ and has sent an action during cycle $t-1$.
- While the server is still in cycle $t-1$, upon receipt of a sensation (see, hear, or sense_body), store the new information in temporary structures. Do not update the current state.
- When the server enters cycle t (determined either by a running clock or by the receipt of a sensation with time stamp t), use all of the information available (temporary information from sensations and predicted effects of past actions) to **update the world model** to match the server's world state (the "real world state") at the end of cycle $t-1$. Then **choose and send an action** to the server for cycle t.
- Repeat for cycle $t+1$.

While the above algorithm defines the overall agent loop, much of the challenge is involved in updating the world model effectively and choosing an appropriate action. The remainder of this section goes into these processes in detail.

3 World Modeling

When acting based on a world model, it is important to have as accurate and precise a model of the world as possible at the time that an action is taken. In order to achieve this goal, CMUnited-98 agents gather sensory information over time, and process the information by incorporating it into the world model immediately prior to acting.

3.1 Object Representation

There are several objects in the world, such as the goals and the field markers which remain stationary and can be used for self-localization. Mobile objects are the agent itself, the ball, and 21 other players (10 teammates and 11 opponents). These objects are represented in a type hierarchy as illustrated in Figure 1.

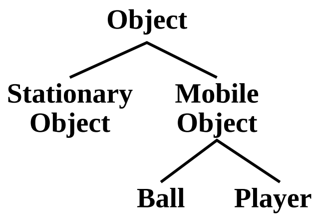

Fig. 1. The agent's object type hierarchy.

Each agent's world model stores an instantiation of a stationary object for each goal, sideline, and field marker; a ball object for the ball; and 21 player objects. Since players can be seen without their associated team and/or uniform number, the player objects are not identified with particular individual players. Instead, the variables for team and uniform number can be filled in as they become known.

Mobile objects are stored with confidence values within [0,1] indicating the confidence with which their locations are known. The confidence values are needed because of the large amount of hidden state in the world: no object is seen consistently. While it would be a mistake to only remember objects that are currently in view, it is also wrong to assume that a mobile object will stay still (or continue moving with the same velocity) indefinitely. By decaying the confidence in unseen objects over time, agents can determine whether or not to rely on the position and velocity values [2].

All information is stored as global coordinates even though both sensor and actuator commands are specified in relative coordinates (angles and distances relative to the agent's position on the field). Global coordinates are easier to store and maintain as the agent moves around the field because the global coordinates of stationary objects do not change as the agent moves, while the relative coordinates do. It is a simple geometric calculation to convert the global coordinates to relative coordinates on demand as long as the agent knows its own position on the field.

The variables associated with each object type are as follows:

Object :
- Global (x, y) position coordinates
- Confidence within [0,1] of the coordinates' accuracy

  Stationary Object : nothing additional

  Mobile Object :
  - Global (dx, dy) velocity coordinates
  - Confidence within [0,1] of the coordinates' accuracy

    Ball : nothing additional

    Player :
    - Team
    - Uniform number
    - Global θ facing angle
    - Confidence within [0,1] of the angle's accuracy

3.2 Updating the World Model

Information about the world can come from

- Visual information;
- Audial information;
- Sense_body information; and
- Predicted effects of previous actions.

Visual information arrives as relative distances and angles to objects in the player's view cone. Audial information could include information about global object locations from teammates. Sense_body information pertains to the client's own status including stamina, view mode, and speed.

Whenever new information arrives, it is stored in temporary structures with time stamps and confidences (1 for visual information, possibly less for audial information). Visual information is stored as relative coordinates until the agent's exact location is determined.

When it is time to act during cycle t, all of the available information is used to best determine the server's world state at the end of cycle $t - 1$. If no new information arrived pertaining to a given object, the velocity and actions taken are used by the predictor to predict the new position of the object and the confidence in that object's position and velocity are both decayed.

When the agent's world model is updated to match the end of simulator cycle $t - 1$, first the agent's own position is updated to match the time of the last sight; then those of the ball and players are updated.

The Agent Itself: Since visual information is given in coordinates relative to the agent's position, it is important to determine the agent's exact position at the time of the sight. When updating the world model to match the end of simulator cycle $t-1$, there may have been visual information with time stamp $t-1$ and/or t (anything earlier would have been incorporated into the previous update of the world model at the end of cycle $t-1$).

If the latest visual information has time stamp $t-1$, then the agent's own position is not updated until after the other objects have been updated as their coordinates are stored relative to the old agent position. On the other hand, if the latest visual information has time stamp t, or if there has been no new visual information since the last world-state update, the agent's status can be updated immediately.

In either case, the following process is used to update the information about the agent:

- If new visual information has arrived:
  - The agent's position can be determined accurately by using the relative coordinates of one seen line and the closest stationary object.
- If no visual information has arrived:
  - Bring the velocity up to date, possibly incorporating the predicted effects of any actions (a dash) taken during the previous cycle.
  - Using the previous position and velocity, predict the new position and velocity.
- If available, reset the agent's speed as per the sense_body information. Assume velocity is in the direction the agent is facing.
- Bring the player's stamina up to date either via the sense_body information or from the predicted action effects.

The Ball: As the key focus of action initiative in the domain, the ball's position and velocity drives a large portion of the agents' decisions. As such, it is important to have as accurate and up-to-date information about the ball as possible.

The ball information is updated as follows:

- If there was new visual information, use the agent's absolute position at the time (determined above), and the ball's temporarily stored relative position to determine the ball's absolute position at the time of the sight.
- If velocity information is given as well, update the velocity. Otherwise, check if the old velocity is correct by comparing the new ball position with the expected ball position.
- If no new visual information arrived or the visual information was from cycle $t-1$, estimate its position and velocity for cycle t using the values from cycle $t-1$. If the agent kicked the ball on the previous cycle, the predicted resulting ball motion is also taken into account.
- If the ball should be in sight (i.e. its predicted position is in the player's view cone), but isn't (i.e. visual information arrived, but no ball information was included), set the confidence to 0.

  - Information about the ball may have also arrived via communication from teammates. If any heard information would increase the confidence in the ball's position or velocity at this time, then it should be used as the correct information. Confidence in teammate information can be determined by the time of the information (did the teammate see the ball more recently?) and the teammate's distance to the ball (since players closer to the ball see it more precisely).

  Ball velocity is particularly important for agents when determining whether or not (or how) to try to intercept the ball, and when kicking the ball. However, velocity information is often not given as part of the visual information string, especially when the ball is near the agent and kickable. Therefore, when necessary, the agents attempt to infer the ball's velocity indirectly from the current and previous ball positions.

Teammates and Opponents: The biggest challenge in keeping track of player positions is that the visual information often does not identify the player that is seen [1]. One might be tempted to ignore all ambiguously-specified players. However, for strategic planning it is very useful to have a complete picture of the player positions around the field.

  In general, player positions and velocities are determined and maintained in the same way as in the case of the ball. A minor addition is that the direction a player is facing is also available from the visual information.

  When a player is seen without full information about its identity, previous player positions can be used to help disambiguate the identity. Knowing the maximum distance a player can move in any given cycle, it is possible for the agent to determine whether a seen player could be the same as a previously identified player. If it is physically possible, the agent assumes that they are indeed the same player.

  Since different players can see different regions of the field in detail, communication can play an important role in maintaining accurate information about player locations.

  From the complete set of player locations, an agent can determine both the defensive and offensive offsides lines. It is particularly important for forwards to stay in front of the last opponent defender in order to avoid being called offsides. Forwards periodically look towards the opponent defenders in order to increase the accuracy of their location information.

4 Agent Skills

Once the agent has determined the server's world state for cycle t as accurately as possible, it can choose and send an action to be executed at the end of the cycle. In so doing, it must choose its local goal within the team's overall strategy. It can then choose from among several low-level skills which provide it with basic capabilities. The output of the skills are primitive movement commands.

The skills available to CMUnited-98 players include kicking, dribbling, ball interception, goaltending, defending, and clearing. The implementation details of these skills are described in this section.

The common thread among these skills is that they are all *predictive, locally optimal skills* (PLOS). They take into account predicted world models 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.

One simple example of PLOS is each individual agent's stamina management. The server models stamina as having a replenishable and a non-replenishable component. Each is only decremented when the current stamina goes below a fixed threshold. Each player monitors its own stamina level to make sure that it never uses up any of the non-replenishable component of its stamina. No matter how fast it should move according to the behavior the player is executing, it slows down its movement to keep itself from getting too tired. While such behavior might not be optimal in the context of the team's goal, it is locally optimal considering the agent's current tired state.

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.

4.1 Kicking

There are three points about the kick model of the server that should be understood before looking at our kicking style. First, a kick changes the ball's velocity by vector addition. That is, a kick accelerates the ball in a given direction, as opposed to setting the velocity. Second, an agent can kick the ball when it is in the "kickable area" which is a circle centered on the player (see Figure 2). Third, the ball and the player can collide. The server models a collision when the ball and player are overlapping at the end of a cycle. If there is a collision, the two bodies are separated and their velocities multiplied by -0.1.

As a first level of abstraction when dealing with the ball, all reasoning is done as a desired trajectory for the ball for the next cycle. Before a kick is actually sent to the server, the difference between the ball's current velocity and the ball's desired velocity is used to determine the kick to actually perform. If the exact trajectory can not be obtained, the ball is kicked such that the direction is correct, even if the speed is not.

In order to effectively control the ball, a player must be able to kick the ball in any direction. In order to do so, the player must be able to move the ball from one side of its body to the other without the ball colliding with the player. This behavior is called the *turnball* behavior. It was developed based on code released by the PaSo'97 team[7]. The desired trajectory of a turnball kick is calculated by getting the ray from the ball's current position that is tangent to a circle around the player (see Figure 3). Note that there are two possible such rays which correspond to the two directions that the ball can be turned around the player. Care is taken to ensure that the ball stays well within the kickable area from kick to kick so that the player keeps control of the ball.

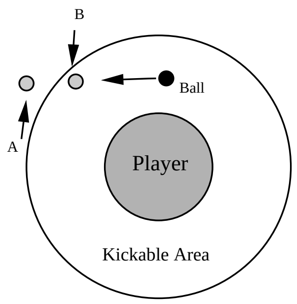

Fig. 2. Basic kicking with velocity prediction.

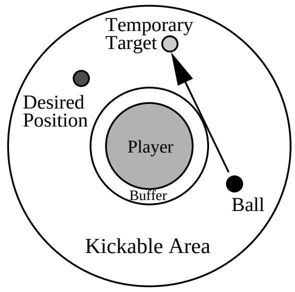

Fig. 3. The turnball skill.

The next important skill is the ability to kick the ball in a given direction, either for passing or shooting. The first step is to figure out the target speed of the ball. If the agent is shooting, the target speed is the maximum ball speed, but for a pass, it might be better to kick the ball slower so that the receiving agent can intercept the ball more easily. In this case, the agent must take into account the ball's deceleration over time when determining how hard to kick the ball.

In order to get the ball to the desired speed, several kicks in succession are usually required. By putting the ball to the side of the player (relative to the desired direction of the kick) the agent can kick the ball several times in succession. If a higher ball speed is desired, the agent can use the turnball kicks to back the ball up so that enough kicks can be performed to accelerate the ball.

This skill is very predictive in that it looks at future velocities of the ball given slightly different possible kicks. In some cases, doing a weaker kick one cycle may keep the ball in the kickable area so that another kick can be executed the following cycle. In Figure 2, the agent must choose between two possible kicks. Kicking the ball to position A will result in the ball not being kickable next cycle; if the ball is already moving quickly enough, this action may be correct. However, a kick to position B followed by a kick during the next cycle may result in a higher overall speed. Short term velocity prediction is the key to these decisions.

4.2 Dribbling

Dribbling is the skill which allows the player to move down the field while keeping the ball close to the player the entire time. The basic idea is fairly simple: alternate kicks and dashes so that after one of each, the ball is still close to the player.

Every cycle, the agent looks to see that if it dashes this cycle, the ball will be in its kickable area (and not be a collision) at the next cycle. If so, then the agent dashes, otherwise it kicks. A kick is always performed assuming that on the next cycle, the agent will dash. As an argument, the low-level dribbling code takes the angle relative to the direction of travel at which the player should aim the ball (see Figure 4). This is called the "dribble angle" and its valid values are $[-90, 90]$. Deciding what the dribble angle should be is discussed in Section 4.3.

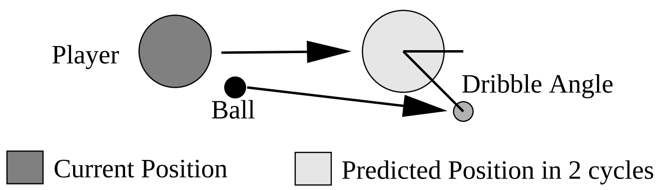

Fig. 4. The basic dribbling skill.

First the predicted position of the agent (in 2 cycles) is calculated:

$$p_{new} = p_{current} + v + (v * pdecay + a)$$

where p_{new} is the predicted player position, $p_{current}$ is the current position of the player, v is the current velocity of the player, $pdecay$ is the server parameter `player_decay`, and a is the acceleration that a dash gives. The a value is usually just the dash power times the `dash_power_rate` in the direction the player is facing, but stamina may need to be taken into account.

Added to p_{new} is a vector in the direction of the dribble angle and length such that the ball is in the kickable area. This is the target position p_{target} of the ball. Then the agent gets the desired ball trajectory by the following formula:

$$traj = \frac{p_{target} - p_{ball}}{1 + bdecay}$$

where $traj$ is the target trajectory of the ball, p_{ball} is the current ball position, and $bdecay$ is the server parameter `ball_decay`. This process is illustrated in Figure 4.

If for some reason this kick can not be done (it would be a collision for example), then a turnball kick is done to get the ball in the right position. Then the next cycle, a normal dribble kick should work.

As can be seen from these calculations, the basic dribbling is highly predictive of the positions and velocities of the ball and player. It is also quite local in that it only looks 2 cycles ahead and recomputes the best action every cycle.

4.3 Smart Dribbling

The basic dribbling takes one parameter that was mentioned above: the dribble angle. Smart dribbling is a skill layered on the basic dribbling skill that decides the best dribble angle based on opponent positions. Intuitively, the agent should keep the ball away from the opponents, so that if an opponent is on the left, the ball is kept on the right, and vice versa.

The agent considers all nearby opponents that it knows about. Each opponent is given a "vote" about what the dribble angle should be; each opponent votes for the valid angle $[-90, 90]$ that is farthest from itself. For example, an opponent at 45 degrees, would vote for -90, while an opponent at -120 degrees would vote for 60. Each opponent's vote is weighted by the distance and angle relative to the direction of motion. Closer opponents and opponents more in front of the agent are given more weight (see Figure 5).

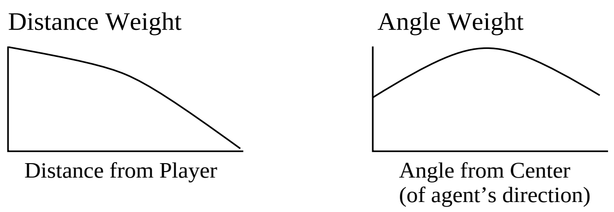

Fig. 5. The weights for smart dribbling.

4.4 Ball Interception

There are two types of ball interception, referred to as active and passive interception. The passive interception is used only by the goaltender in some particular cases, while the rest of the team uses only the active interception. Each cycle, the interception target is recomputed so that the most up to date information about the world is used.

The *active interception* is similar to the one used by the Humboldt '97 team[3]. The active interception predicts the ball's position on successive cycles, and then tries to predict whether the player will be able to make it to that spot before the ball does, taking into account stamina and the direction that the player is facing. The agent aims for the earliest such spot.

This process can be used for teammates as well as for the agent itself. Thus, the agent can determine which player should go for the ball, and whether it can get there before the opponents do.

The *passive interception* is much more geometric. The agent determines the closest point along the ball's current trajectory that is within the field. By prediction based on the ball's velocity, the agent decides whether it can make it to that point before the ball. If so, then the agent runs towards that point.

4.5 Goaltending

The assumption behind the movement of the goaltender is that the worst thing that could happen to the goaltender is to lose sight of the ball. The sooner the goaltender sees a shot coming, the greater chance it has of preventing a goal. Therefore, the goaltender generally uses the widest view mode and uses backwards dashing when appropriate to keep the ball in view to position itself in situations that are not time-critical.

Every cycle that the ball is in the defensive zone, the goaltender looks to see if the ball is in the midst of a shot. It does this by extending the ray of the ball's position and velocity and intersecting that with the baseline of the field. If the intersection point is in the goaltender box and the ball has sufficient velocity to get there, the ball is considered to be a shot (though special care is used if an opponent can kick the ball this cycle). Using the passive interception if possible (see Section 4.4), the goaltender tries to get in the path of the ball and then run at the ball to grab it. This way, if the goaltender misses a catch or kick, the ball may still collide with the goaltender and thus be stopped.

When there is no shot coming the goaltender positions itself in anticipation of a future shot. Based on the angle of the ball relative to the goal, the goaltender picks a spot in the goal to guard; call this the "guard point." The further the ball is to the side of the field, the further the goaltender guards to that side. Then, a rectangle is computed that shrinks as the ball gets closer (though it never shrinks smaller than the goaltender box). The line from the guard point to the ball's current position is intersected with the rectangle, and that is the desired position of the goaltender.

4.6 Defending

CMUnited-98 agents are equipped with two different defending modes: opponent tracking and opponent marking. In both cases, a particular opponent player is selected as the target against which to defend. This opponent can either be selected individually or as a defensive unit via communication (the latter is the case in CMUnited-98).

In either case, the agent defends against this player by observing its position over time and position itself strategically so as to minimize its usefulness to the other team. When *tracking*, the agent stays between the opponent and the goal at a generous distance, thus blocking potential shots. When *marking*, the agent stays close to the opponent on the ball-opponent-goal angle bisector, making it difficult for the opponent to receive passes and shoot towards the goal. Defensive marking and tracking positions are illustrated in Figure 6.

When marking and tracking, it is important for the agent to have accurate knowledge about the positions of both the ball and the opponent (although the ball position isn't strictly relevant for tracking, it is used for the decision of whether or not to be tracking). Thus, when in the correct defensive position, the agent always turns to look at the object (opponent or ball) in which it is least confident of the correct position. The complete algorithm, which results in the behavior of doggedly following a particular opponent and glancing back and forth between the opponent and ball, is as follows:

- If the ball position is unknown, look for the ball.

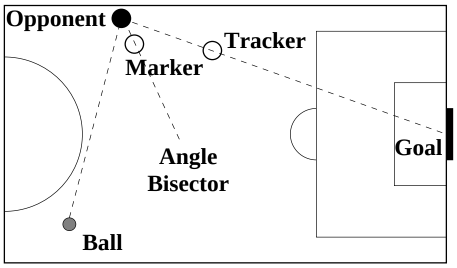

Fig. 6. Positioning for defensive tracking and marking.

– Else, if the opponent position is unknown, look for the opponent.
– Else, if not in the correct defensive position, move to that position.
– else, look towards the object, ball or opponent, which has been seen less recently (lower confidence value).

This defensive behavior is locally optimal in that it defends according to the opponent's current position, following it around rather than predicting its future location. However, in both cases, the defensive positioning is chosen in anticipation of the opponent's future possible actions, i.e. receiving a pass or shooting.

4.7 Clearing

Often in a defensive position, it is advantageous to just send the ball upfield, clearing it from the defensive zone. If the agent decides that it cannot pass or dribble while in a defensive situation, it will clear the ball. The important decision in clearing the ball is where to clear it to. The best clears are upfield, but not to the middle of the field (you don't want to center the ball for the opponents), and also away from the opponents.

The actual calculation is as follows. Every angle is evaluated with respect to its usefulness, and the expected degree of success. The usefulness is a sine curve with a maximum of 1 at 30 degrees, .5 at 90 degrees, and 0 at -90, where a negative angle is towards the middle of the field. The actual equation is (Θ is in degrees):

$$\text{usefulness}(\Theta) = \frac{sin(\frac{3}{2}\Theta + 45) + 1}{2} \quad (1)$$

The expected degree of success is evaluated by looking at an isosceles triangle with one vertex where the ball is, and congruent sides extending in the direction of the target being evaluated. For each opponent in the triangle, its distance from the center line of the triangle is divided by the distance from the player on that line. For opponent C in Figure 7, these values are w and d respectively. The expected success is the product of all these quotients. In Figure 7, opponent A would not affect the calculation, being outside the triangle, while opponent

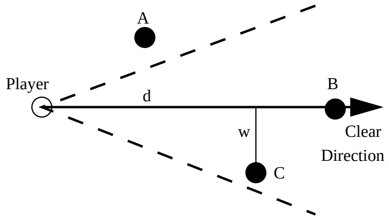

Fig. 7. Measuring the expected success of a clear.

B would lower the expected success to 0, since it is on the potential clear line ($w = 0$).

By multiplying the usefulness and expected success together for each possible clear angle, and taking the maximum, the agent gets a crude approximation to maximizing the expected utility of a clear.

There is a closely related behavior of offensive "sending." Rather than trying to clear the ball to the sides, the agent sends the ball to the middle of the offensive zone with hopes that a teammate will catch up to the ball before the defenders. This is useful if the agent is too tired or unable to dribble for some reason. It is especially useful to beat an offsides trap because it generally requires the defenders to run back to get the ball.

The only difference with defensive clearing is the usefulness function. For sending, the usefulness function is linear, with slope determined by the agent's Y position on the field. The closer the agent is to the sideline, the steeper the slope, and the more that it favors sending to the middle of the field.

5 Coordination

Given all of the individual skills available to the CMUnited-98 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. Of course one and only one agent should execute the goaltending behavior. But it is not so clear how to determine when an agent should move towards the ball, when it should defend, when it should dribble, or clear, etc.

If all players act individually — constantly chase the ball and try to kick towards the opponent goal — they will all get tired, there will be nowhere to pass, and the opponents will have free reign over most of the field. Building upon the innovations of the CMUnited-97 simulator team [8], the CMUnited-98 team uses several complex coordination mechanisms, including reactive behavior modes, pre-compiled multi-agent plans and strategies, a flexible teamwork structure, a novel anticipatory offensive positioning scheme, and a sophisticated communication paradigm.

5.1 Behavior Modes

A player's top-level behavior decision is its behavior mode. Implemented as a rule-based system, the behavior mode determines the abstract behavior that the player should execute. For example, there is a behavior mode for the set of states in which the agent can kick the ball. Then, the decision of what to do with the ball is made by way of a more involved decision mechanism. On each action cycle, the first thing a player does is re-evaluate its behavior mode.

The behavior modes include:

Goaltend: Only used by the goaltender.
Localize: Find own field location if it's unknown.
Face Ball: Find the ball and look at it.
Handle Ball: Used when the ball is kickable.
Active Offense: Go to the ball as quickly as possible. Used when no teammate could get there more quickly.
Auxiliary Offense: Get open for a pass. Used when a nearby teammate has the ball.
Passive Offense: Move to a position likely to be useful offensively in the future.
Active Defense: Go to the ball even though another teammate is already going. Used in the defensive end of the field.
Auxiliary Defense: Mark an opponent.
Passive Defense: Track an opponent or go to a position likely to be useful defensively in the future.

The detailed conditions and effects of each behavior mode are beyond the scope of this article. However, they will become more clear in subsequent sections as the role-based flexible team structure is described in Section 5.3.

5.2 Locker-Room Agreement

At the core of the CMUnited-98 coordination mechanism is what we call the Locker-Room Agreement [10]. 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.

The locker-room agreement includes specifications of the flexible teamwork structure (Section 5.3) and the inter-agent communication paradigm (Section 5.5). A good example of the use of the locker-room agreement is CMUnited-98'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.

A new addition to CMUnited-98's locker-room agreement is a defensive offsides strategy. Since the rules of the soccer server prohibit an opponent from receiving a pass when located behind the last defender on the opponent's attacking half of the field[2], it is an effective defensive strategy to move all the defenders forward towards midfield. However, if only one defender is farther back than the rest of the team, the strategy can back-fire horribly.

To take advantage of this rule using the locker-room agreement, the team agrees on a formula based on the location of the ball and the opponent's furthest-back defender. This strategy relies on relatively consistent sensing by all of the defensive players, but it does not require any communication. Independently, the players can dynamically adjust their positions as the ball and opponents move so that the team's defenders stay in a coordinated line.

The CMUnited-98 offsides line was always at least 15 meters behind the current ball position to prevent opponents from dribbling through to goal and at least 40 meters behind the opponents last defender to allow enough room in the midfield to pass the ball amongst teammates.

5.3 Roles and Formations

Like CMUnited-97, CMUnited-98 is organized around 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, forwards never go into auxiliary defense mode and defenders never go into auxiliary offense mode.

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. After testing about 10 formations, the CMUnited-98 team ended up selecting from among 3 different formations. A standard formation with 4 defenders, 3 midfielders, and 3 forwards (4-3-3) was used at the beginnings of the games. If losing by enough goals relative to the time left in the game (as determined by the locker-room agreement), the team would switch to an offensive 3-3-4 formation. When winning by enough, the team switched to a defensive 5-3-2 formation.

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 [10].

5.4 SPAR

The flexible roles defined in the CMUnited-97 software were an improvement over the concept of rigid roles. Rather than associating fixed (x, y) coordinates with

---

[2] As in real soccer, the offsides rule is more complicated than that. But for the purposes of this article, the above definition is sufficient.

each position, an agent filling a particular role was given a range of coordinates in which it could position itself. Based on the ball's position on the field, the agent would position itself so as to increase the likelihood of being useful to the team in the future.

However, by taking into account the positions of other agents as well as that of the ball, an even more informed positioning decision can be made. The idea of *strategic position by attraction and repulsion* (SPAR) is one of the novel contributions of the CMUnited-98 research which has been applied to both the simulator and the small robot teams [12].

When positioning itself using SPAR, the agent uses a multi-objective function with attraction and repulsion points subject to several constraints. To formalize this concept, we introduce the following variables:

- P - the desired position for the passive agent in anticipation of a passing need of its active teammate;
- n - the number of agents on each team;
- O_i - the current position of each opponent, $i = 1, \ldots, n$;
- T_i - the current position of each teammate, $i = 1, \ldots, (n-1)$;
- B - the current position of the active teammate and ball;
- G - the position of the opponent's goal.

SPAR extends similar approaches of using potential fields for highly dynamic, multi-agent domains [6]. The probability of collaboration in the robotic soccer domain is directly related to how "open" a position is to allow for a successful pass. Thus, SPAR maximizes the distance from other robots and minimizes the distance to the ball and to the goal, namely:

- *Repulsion* from opponents, i.e., maximize the distance to each opponent: $\forall i, \max dist(P, O_i)$
- *Repulsion* from teammates, i.e., maximize the distance to other passive teammates: $\forall i, \max dist(P, T_i)$
- *Attraction* to the active teammate and ball: $\min dist(P, B)$
- *Attraction* to the opponent's goal: $\min dist(P, G)$

This formulation is a multiple-objective function. To solve this optimization problem, we restate the problem as a single-objective function. As each term may have a different relevance (e.g. staying close to the goal may be more important than staying away from opponents), we want to apply a different weighting function to each term, namely f_{O_i}, f_{T_i}, f_B, and f_G, for opponents, teammates, the ball, and the goal, respectively. Our anticipation algorithm then maximizes a weighted single-objective function with respect to P:

$$\max\left(\sum_{i=1}^{n} f_{O_i}(dist(P, O_i)) + \sum_{i=1}^{n-1} f_{T_i}(dist(P, T_i)) - f_B(dist(P, B)) - f_G(dist(P, G))\right)$$

In our case, we use $f_{O_i} = f_{T_i} = x$, $f_B = 0$, and $f_G = x^2$. For example, the last term of the objective function above expands to $(dist(P, G))^2$.

One constraint in the simulator team relates to the position, or role, that the passive agent is playing relative to the position of the ball. The agent only considers locations that within one of the four rectangles, illustrated in Figure 5.4: the one closest to the position home of the position that it is currently playing. This constraint helps ensure that the player with the ball will have several different passing options in different parts of the field. In addition, players don't need to consider moving too far from their positions to support the ball.

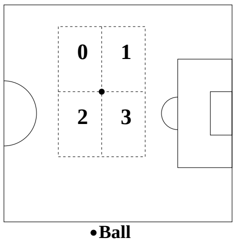

Fig. 8. The four possible rectangles, each with one corner at the ball's location, considered for positioning by simulator agents when using SPAR.

Since this position-based constraint already encourages players to stay near the ball, we set the ball-attraction weighting function f_B to the constant function $y = 0$. In addition to this first constraint, the agents observe three additional constraints. In total, the constraints in the simulator team are:

- Stay in an area near home position;
- Stay within the field boundaries;
- Avoid being in an offsides position;
- Stay in a position in which it would be possible to receive a pass.

This last constraint is evaluated by checking that there are no opponents in a cone with vertex at the ball and extending to the point in consideration.

In our implementation, the maximum of the objective function is estimated by sampling its values over a fine-grained mesh of points that satisfy the above constraints.

Using this SPAR algorithm, agents are able to *anticipate* the collaborative needs of their teammates by positioning themselves in such a way that the player with the ball would have several useful passing options.

5.5 Communication

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-98 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 [10].

5.6 Ball Handling

One of the most important decisions in the robotic soccer domain arises when the agent has control of the ball. In this state, it 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.

In CMUnited-98, the agent uses a complex heuristic decision mechanism, incorporating a machine learning module, to choose its action. The best teammate to receive a potential pass (called *potential receiver* below) is determined by a decision tree trained off-line [9]. Following is a rough sketch of the decision-making process without all of the parametric details.

To begin with, since kicks (i.e. shots, passes, and clears) can take several cycles to complete (Section 4.1), the agent remembers the goal of a previously started kick and continues executing it. When no kick is in progress (do the first that applies):

- If close to the opponent's goal and no defenders are blocking the path to the goal (defined as a cone with vertex at the ball): shoot or dribble based on the goaltender's position, the position of the closest opponent, and the distance to the goal.
- At the other extreme, if close to the agent's own goal and there is an opponent nearby: clear the ball.
- If approaching the line of the last opponent defender: dribble the ball forward if possible; otherwise send the ball (clear) past the defender.
- If the potential receiver is closer to the goal and has a clear shot: pass to the potential receiver.
- If no opponents are in the direct path to the goal: dribble to the goal.

- If fairly close to the opponent's goal and there is at most one opponent in front of the goal: shoot.
- If no opponents are in the way of one of the corner flags: dribble towards the corner flag.
- If there is a potential receiver: pass.
- If it's possible to hold onto the ball without moving (at most one opponent is nearby): hold the ball.
- Otherwise: Kick the ball away (clear).

6 Results

In order to test individual components of the CMUnited-98 team, it is best to compile performance results for the team with and without these components as we have done elsewhere [10]. However, competition against other, independently-created teams is useful for evaluating the system as a whole.

At the RoboCup-98 competition, CMUnited-98 won all 8 of its games by a combined score of 66–0, finishing first place in a field of 34 teams. Table 1 details the game results.

Opponent	Affiliation	Score (CMU–Opp.)
UU	Utrecht University, The Netherlands	22 – 0
TUM / TUMSA	Technical University Munich, Germany	2 – 0
Kasuga-Bitos II	Chubu University, Japan	5 – 0
Andhill'98	NEC, Japan	8 – 0
ISIS	Information Sciences Institute (USC), USA	12 – 0
Rolling Brains	Johannes Gutenberg-University Mainz, Germany	13 – 0
Windmill Wanderers	University of Amsterdam, The Netherlands	1 – 0
AT-Humboldt'98	Humboldt University of Berlin, Germany	3 – 0
TOTAL		66 – 0

Table 1. The scores of CMUnited-98's games in the simulator league of RoboCup-98. CMUnited-98 won all 8 games, finishing in 1st place out of 34 teams.

From observing the games, it was apparent that the CMUnited-98 low-level skills were superior in the first 6 games: CMUnited-98 agents were able to dribble around opponents, had many scoring opportunities, and suffered few shots against.

However, in the last 2 games, the CMUnited-98 strategic formations, communication, and ball-handling routines were put more to the test as the Windmill

Wanderers (3rd place) and AT-Humboldt'98 (2nd place) also had similar low-level capabilities. In these games, CMUnited-98's abilities to use set-plays to clear the ball from its defensive zone, to get past the opponents' offsides traps, and to maintain a cohesive defensive unit became very apparent. Many of the goals scored by CMUnited-98 were a direct result of the opponent team being unable to clear the ball from its own end after a goal kick: a CMUnited-98 player would intercept the clearing pass and quickly shoot it into the goal. In particular, two of the goals in the final game against AT-Humboldt'98 were scored in this manner. On the other hand, the CMUnited-98 simulator team was able to clear the ball successfully from its own zone using its ability to execute set-plays, or pre-compiled multi-agent plans. Rather than kicking the ball up the middle of the field, one player would pass out to the sideline to a second player that would then clear the ball up the field. After a series of 3 or 4 passes, the ball was usually safely in the other half of the field.

Another strategic advantage that was clear throughout CMUnited-98's games was the players' abilities to maintain a coherent defensive unit exploiting the offsides rule, and conversely, its ability to get through the defense of other teams. Often, the opposing teams were unable to get anywhere near the CMUnited-98 goal because of the defenders' ability to stay in front of some of the opposing forwards, thus rendering them offsides and prohibiting them from ever successfully receiving the ball.

In order to deal with opposing teams that tried to use a similar technique, the CMUnited-98 forwards would kick the ball towards the offensive corners of the field (the "sending" skill described in Section 4.7) and then either get to the ball before the defenders or intercept defenders' clearing passes. CMUnited-98 scored several nice goals after such kicks to the corners.

In addition to the strategic reasoning that helped the team win its final two games, the fine points of the dribbling and goaltending skills also came into play. Using their predictive, locally optimal skills (PLOS—see Section 4), the CMUnited-98 players were occasionally able to dribble around opponents for shots. At a crucial moment against the Windmill Wanderers, the CMUnited-98 goaltender made a particularly important save: winning 1–0 near the end of the game, a shot got past the goaltender, but it was able to turn and catch the ball before the ball entered the goal.

7 Conclusion

The success of CMUnited-98 at RoboCup-98 was due to several technical innovations ranging from predictive locally optimal skills (PLOS) to strategic positioning using attraction and repulsion (SPAR). Building on the innovations of CMUnited-97, including flexible formation, a novel communication paradigm, and machine learning modules, CMUnited-98 successfully combines low-level individual and high-level strategic, collaborative reasoning in a single multi-agent architecture.

For a more thorough understanding of the implementation details involved, the reader is encouraged to study the algorithms described here in conjunction with the CMUnited-98 source code [11]. Other RoboCup researchers and multi-agent researchers in general should be able to benefit and build from the innovations represented therein.

References

1. David Andre, Emiel Corten, Klaus Dorer, Pascal Gugenberger, Marius Joldos, Johan Kummeneje, Paul Arthur Navratil, Itsuki Noda, Patrick Riley, Peter Stone, Romoichi Takahashi, and Travlex Yeap. Soccer server manual, version 4.0. Technical Report RoboCup-1998-001, RoboCup, 1998. At URL http://ci.etl.go.jp/˜noda/soccer/server/Documents.html.
2. Mike Bowling, Peter Stone, and Manuela Veloso. Predictive memory for an inaccessible environment. In *Proceedings of the IROS-96 Workshop on RoboCup*, pages 28–34, Osaka, Japan, November 1996.
3. Hans-Diter Burkhard, Markus Hannebauer, and Jan Wendler. AT humboldt — development, practice and theory. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 357–372. Springer Verlag, Berlin, 1998.
4. Hiroaki Kitano, Yasuo Kuniyoshi, Itsuki Noda, Minoru Asada, Hitoshi Matsubara, and Eiichi Osawa. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.
5. 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.
6. Jean-Claude Latombe. *Robot Motion Planning*. Kluwer, 1991.
7. E. Pagello, F. Montesello, A. D'Angelo, and C. Ferrari. A reactive architecture for RoboCup competition. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 434–442. Springer Verlag, Berlin, 1998.
8. Peter Stone and Manuela Veloso. The CMUnited-97 simulator team. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 387–397. Springer Verlag, Berlin, 1998.
9. Peter Stone and Manuela Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.
10. Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 1999. To appear.
11. Peter Stone, Manuela Veloso, and Patrick Riley. CMUnited-98 source code, 1998. Accessible from http://www.cs.cmu.edu/˜ pstone/RoboCup/CMUnited98-sim.html.
12. 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.