Plays as Team Plans for Coordination and Adaptation *

Michael Bowling, Brett Browning, Allen Chang and **Manuela Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh PA, 15213-3891

Abstract

Coordinated action for a team of robots is a challenging problem, especially in dynamic, unpredictable environments. We examine this problem in the context of robot soccer, a complex domain with multiple teams of robots in an adversarial setting. The adversarial nature creates a great deal of uncertainty, in both the opponent's behavior and capabilities. Traditional approaches focus on building reactive systems that use simple or even complex evaluation procedures for selecting team and individual robot actions given the state of the world. We introduce the concept of a *play* as a team plan, combining both reactive and deliberative principles. We introduce the concept of a *playbook* as a method for seamlessly combining many different team plans. The playbook provides a set of alternative team behaviors, and is the basis for our third contribution of *play adaptation*. We describe how these concepts were concretely implemented in the CMDragons robot soccer team, the first RoboCup robot team to adapt to its opponent during the game. We also show empirical results of the importance of adaptation in adversarial or other unpredictable environments.

1 Introduction

Coordination and adaptation are two of the most critical challenges for deploying teams of robots to perform useful tasks. These challenges become especially difficult in environments involving other agents, particularly adversarial ones, not under the team's control. In this paper, we examine these challenges within the context of robot soccer [6], a multi-robot goal-driven task in an adversarial environment. The robot soccer task is goal-driven and highly dynamic. The presence of adversaries creates significant uncertainty for predicting the outcome of interactions. This is particularly true if the opponent's behavior and capabilities are unknown a priori, as is the case in a robot soccer competition. As such, this task encapsulates many of the issues found in realistic multi-robot settings.

Despite this unpredictability, most robot soccer approaches involve single, static, monolithic team strategies (e.g., see robot team descriptions in [1].) Although these strategies entail complex combinations of reactive and deliberative approaches, they can still perform poorly against unknown opponents or in unexpected situations. With the uncertainty present in the task, such situations are common.

An alternative approach uses models of opponent behavior, constructed either before or during the competition [5], and then determine the best team response. The model may be used in a reactive fashion to trigger a pre-coded static strategy, or in a deliberative fashion through the use of a planner [7]. Although these techniques have had success, they have limitations such as the requirement for an adequate representation of opponent behavior. For a completely unknown opponent team, a prior model of their strategy is impractical.

Here, we take a novel approach based on observing our own team's effectiveness rather than observing the opponent. We replace a single monolithic team strategy, with multiple team plans that are appropriate for different opponents and situations, which we call *plays*. Each play defines a coordinated sequence of team behavior, and is explicit enough to facilitate evaluation of that play's execution. A *playbook* encapsulates the plays that a team can use. Each execution of a play from the playbook can then be evaluated and this information collected for future play selection. Successful plays, which may be due to weaknesses in the opponent or particular strengths of our team, are selected more often, while unsuccessful plays are ignored.

In Section 2, we overview our CMDragons'02 robot soccer team and the role of plays in the team's decision making. We then, in Section 3, describe plays and the play execution system in detail. In Section 4 we describe play adaptation with empirical results demonstrating its importance and effectiveness, and then conclude.

2 Overview

In this section, we briefly describe our CMDragons'02 robot soccer team, the basis for the work explored in this paper. This overview focuses on how the strategy system, described in Section 3, interacts with the system as a whole.

*This research was sponsored by Grants Nos. DABT63-99-1-0013, F30602-98-2-013 and F30602-97-2-020. The information in this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

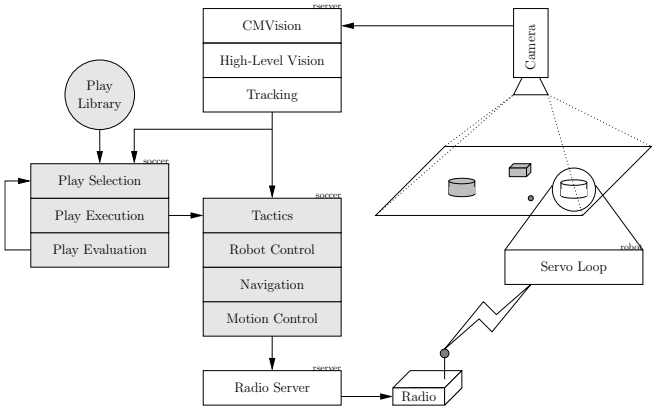


Figure 1: Overview of the CMDragons'02 team architecture.

The CMDragons are a team of small-size league (SSL) soccer robots that participated at RoboCup 2002. SSL robot soccer, part of the RoboCup initiative [6], consists of two teams of five robots that play soccer with an orange golf-ball on a 2.8m by 2.3m field surrounded by short, sloped walls using FIFA-like rules enforced by a human referee. Robots must conform to size and shape specifications, but no standard platforms exist. SSL is characterized by the allowance of cameras mounted above the field for shared global perception and additional off-field computation resources making a team as a whole autonomous rather than individual robots. SSL robots are typically fast, cruising at speeds of 1–2m/s while the ball moves at over 2m/s, and occasionally much faster. This makes SSL an environment that requires fast response, good long-term strategy, strong individual robot skills and capable multi-robot coordination, for a team to be successful.

Figure 1 shows the major components of the CMDragons system. The control loop, synchronized with image frames at 30Hz, consists of taking an image from the camera via the frame grabber and processing it, determining the control for each robot on the team, and sending these velocity commands using radio communication to the robots. Each robot operates local servo loops to enact its commands. Due to space limitations, we refer the reader to our earlier works [3] and [4], to describe the system in more detail. Instead, we focus on the tactics and strategy layers of the control software.

The tactics and strategy layers, the shaded regions in Figure 1, make up the bulk of the system. The tactics layer, encompasses individual robot skills. Here the notion of a *tactic* is synonymous with the term *Behaviors* often used in the literature. We use the terms tactics and strategy to reflect the differentiation between determining what the team members will do, and how each team member will do it. In practice, each robot executes a single tactic independently of the others each and every frame. The strategy layer provides the coordination mechanism and executes one instance for the entire team. Thus, the strategy layer must meld individual robot skills into powerful and adaptable team behavior. Here we focus on tactics, leaving discussion of strategy to the remainder of the paper.

Tactics are defined to be any behavior executable by a single robot. Table 1 shows the list of implemented tactics . Each tactic is highly parameterized and performs a complex, single robot task that itself may consist of many sub-components. Each tactic makes use of the robot control layer that maintains robot specific information persistent even after a robot's tactic changes. The layer transforms tactic commands into target points for navigation. The navigation layer produces short term, obstacle free, way-points for the motion control system using a fast randomized planner. These way-points are used by the motion control module to generate the actual velocity commands sent to the robot.

A tactic, therefore, is a complex interaction between low-levels of navigation and motion control and higher-level skill-based code. For example, consider the `position_for_deflection` tactic. The tactic itself determines the best location within a region for deflecting passes into the goal. This requires sampling points over the region and evaluating the deflection angles using a deflection heuristic. The best evaluated point is then fed into the navigation layer and in turn to the motion control layer to generate the actual commands necessary to drive the robot to the calculated position.

3 Play-Based Strategy

The main question addressed in this work is: "Given a set of effective and parameterized individual robot behaviors, how do we select each robot's behavior to achieve the team's goals?" This is the problem addressed by our strategy component, which is diagrammed by the left-most shaded components of Figure 1. Our team strategy utilizes the concept of a *play* as a team plan with multiple plays collected into a *playbook*.

3.1 Goals

The main criterion for team strategy is performance. A single, static, monolithic team strategy that maximizes performance, though, is impractical. Indeed, in an adversarial domain with an unknown opponent, a single optimal strategy is unlikely to exist. Therefore we have broken down the performance criterion into easier to achieve subgoals. The goals of a strategy system are:

1. Coordinates team behavior,
2. Executes temporally extended sequences of action,
3. Allows for special behavior for certain circumstances,
4. Allows ease of human design and augmentation,
5. Enables exploitation of short-lived opportunities, and
6. Allows on-line adaptation to the specific opponent.

The first four goals require plays to be able to express complex, coordinated, and sequenced behavior among teammates. In addition, plays must be human readable to make strategy design and modification simple. These goals also requires a powerful system capable of executing the complex behaviors the play describes. The fifth goal requires the execution system to also recognize and exploit opportunities that are not explicitly described by the current play. Finally, the sixth goal requires the system to alter its behavior over time.

Active Tactics	Non-active Tactics
shoot (Aim | Noaim | Deflect ⟨*role*⟩)	position_for_loose_ball ⟨*region*⟩
steal [⟨*coordinate*⟩]	position_for_rebound ⟨*region*⟩
clear	position_for_pass ⟨*region*⟩
active_def [⟨*coordinate*⟩]	position_for_deflection ⟨*region*⟩
pass ⟨*role*⟩	defend_line ⟨*p1*⟩ ⟨*p2*⟩ ⟨*min-dist*⟩ ⟨*max-dist*⟩
dribble_to_shoot ⟨*region*⟩	defend_point ⟨*p1*⟩ ⟨*min-dist*⟩ ⟨*max-dist*⟩
dribble_to_region ⟨*region*⟩	defend_lane ⟨*p1*⟩ ⟨*p2*⟩
spin_to_region ⟨*region*⟩	block ⟨*min-dist*⟩ ⟨*max-dist*⟩ ⟨*side-pref*⟩
receive_pass	mark ⟨*orole*⟩ (ball | our_goal | their_goal | shot)
receive_deflection	goalie
dribble_to_position ⟨*coordinate*⟩ ⟨*theta*⟩	stop
position_for_start ⟨*coordinate*⟩ ⟨*theta*⟩	velocity ⟨*vx*⟩ ⟨*vy*⟩ ⟨*vtheta*⟩
position_for_kick	position ⟨*coordinate*⟩ ⟨*theta*⟩
position_for_penalty	
charge_ball	

Table 1: List of tactics along with their parameters.

Notice that these goals, although critical to the robot soccer task, are also of general importance for the coordination of agent teams in other unpredictable or adversarial environments. We have developed a play-based team strategy, using a specialized play language, to meet these goals. In the following sections, we will describe the three major components of the play-based strategy engine: play specification using the play language, the play execution system, and the playbook adaptation mechanism used to autonomously alter team strategy to a specific opponent during a competition.

3.2 Play Specification

Plays are specified using the play language, which is in an easy-to-read text format (e.g., Tables 2 and 4). Plays use keywords, denoted by all capital letters, to mark different pieces of information. Each play has two components: *basic information* and *role information*. The basic information describes when a play can be executed ("APPLICABLE"), when execution of the play should stop ("DONE"), and some execution details (e.g., "FIXEDROLES", "TIMEOUT", and "OROLE"). The role information ("ROLE") describes how the play is executed, making use of the tactics described above (see Section 2). We describe these keywords below.

Applicability. The APPLICABLE keyword denotes when a play can be executed. What follows the keyword is a conjunction of high-level predicates that all must be true for the play to be considered executable. Multiple APPLICABLE keywords can be used to denote different disjunctive conditions for when the play may be executed. This allows plays to effectively specify when they can be executed as a logical DNF of high-level predicates. In the example play in Table 2, the play can only be executed when the offense predicate is true. The offense predicate is a complex combination of the present and past possession of the ball and its field position. Predicates are easily added and Table 3 lists the predicates in use in our system.

A play's applicability condition is very similar to an operator's preconditions in classical planning. By constraining the applicability of a play we can design special purpose plays for very specific circumstances. Table 4 shows an example play that uses the in_their_corner predicate to constrain the play to execute only when the ball is in one of the opponent's corners. The play explicitly involves dribbling the ball out of the corner to get a better angle for a shot on goal.

```
PLAY Two Attackers, Pass

APPLICABLE offense
DONE aborted !offense

OROLE 0 closest_to_ball

ROLE 1
  pass 3
  mark 0 from_shot
  none
ROLE 2
  block 320 900 -1
  none
ROLE 3
  position_for_pass { R { B 1000 0 } ...
  receive_pass
  shoot A
  none
ROLE 4
  defend_line { -1400 1150 } ...
  none
```

Table 2: A complex play involving sequencing of behaviors.

Termination. Unlike classical planning, the level of uncertainty in this task makes it difficult to predict the outcome of a particular plan. Although, a play does not have effects, it does have termination conditions. Termination conditions are specified by the keyword DONE followed by a result (e.g., aborted) and a conjunctive list of high-level predicates similar to the applicability conditions. Plays may have multiple DONE conditions, each with a different result, and a different conjunct of predicates. Whenever one of these DONE conditions are satisfied the play is terminated, and a new play must be selected. In the example play in Table 2, the only terminating condition is if the team is no longer on offense. In this case the play's result is considered to have been aborted.

The results for plays are: succeeded, completed, aborted, and failed. These results are used to evalu-

offense	our_kickoff
defense	their_kickoff
their_ball	our_freekick
our_ball	their_freekick
loose_ball	our_penalty
their_side	their_penalty
our_side	ball_x_gt ⟨X⟩
midfield	ball_x_lt ⟨Y⟩
in_our_corner	ball_absy_gt ⟨Y⟩
in_their_corner	ball_absy_lt ⟨Y⟩
nopponents_our_side ⟨N⟩	

Table 3: List of high-level predicates.

ate the success of the play for the purposes of reselecting the play later. This is the major input to the play adaptation system which we describe in the next section.

There are two other ways in which plays can be terminated. The first is when the sequence of behaviors defined by the play are `completed`, which is described with the play execution system below. The second occurs when a play runs for too long without terminating. The timeout causes the play to terminate with an `aborted` result and a new play is selected. Thus, the team commits to a course of action, but if no progress is made due to unforeseen circumstances, another approach will be tried. The timeout period has a team configurable default value, however, a play may use the `TIMEOUT` keyword to override this default timeout limit (e.g. Table 4).

Roles. Roles are the active component of each play, and each play has four roles corresponding to the non-goalie robots on the field. Each role contains a list of tactics (also called behaviors) with associated parameters for the robot to perform in sequence. As tactics are heavily parameterized, the range of tactics can be combined into nearly an infinite number of play possibilities. Table 4 shows an example play where the first role executes two sequenced tactics. First the robot dribbles the ball out of the corner and then switches to the shooting behavior. Meanwhile the other roles execute a single behavior for the play's duration.

Sequencing implies an enforced synchronization, or coordination between roles. Once a tactic completes, all roles move to their next behavior in their sequence (if one is defined). Thus, in the example in Table 2, when the player assigned to pass the ball completes the pass, then it will switch to the mark behavior. The receiver of the pass will simultaneously switch to receive the pass, after which it will try to execute the shooting tactic.

Opponent Roles. Some behaviors are dependent on the positions of specific opponents on the field. Opponent roles are used to identify a specific opponent based on an evaluation method for the tactic to use. The example in Table 2, shows an opponent role defined using the `OROLE` keyword and the `closest_to_ball` method. Thus, the first role will try to mark the opponent closest to the ball away from the ensuing shot, after executing the pass.

Coordinate Systems. Parameters for tactics are also very general by allowing for a variety of coordinate systems in specifying points and regions on the field. Coordinates may be specified as absolute field positions or ball relative field positions. In addition, a coordinate system's positive *y*-axis can be oriented to point towards the side of the field that the ball is on, the side of field the majority of the opponents are on, or even a careful combination of these two factors. This allows tremendous flexibility in the specification of the behaviors used in plays and prevents unnecessary duplication of plays for symmetric field situations.

```
PLAY Two Attackers, Corner Dribble 1

APPLICABLE offense in_their_corner
DONE aborted !offense
TIMEOUT 15

ROLE 1
  dribble_to_shoot { R { B 1100 800 } ...
  shoot A
  none
ROLE 2
  block 320 900 -1
  none
ROLE 3
  position_for_pass { R { B 1000 0 } ...
  none
ROLE 4
  defend_line { -1400 1150 } ...
  none
```

Table 4: A special purpose play that is only executed when the ball is in an offensive corner of the field.

3.3 Play Execution

The play execution module is responsible for instantiating the active play into actual robot behavior. Instantiation consists of many key decisions: role assignment, role switching, sequencing tactics, opportunistic behavior, and termination.

Role assignment is dynamic, rather than being fixed, and is determined by uses tactic-specific methods. To prevent conflicts, assignment is prioritized by the order in which roles appear. Thus, the first role, which usually involves ball manipulation, is assigned first and considers all four field robots. The next role is assigned to one of the remaining robots, and so on. The prioritization provides the execution system the knowledge to select the best robots to perform each role and also provides the basis for role switching. Role switching is a very effective technique for exploiting changes in the environment that alter the effectiveness of robots fulfilling roles. The executor continuously reexamines the role assignment for possible opportunities to improve it as the environment changes. Although, it has a strong bias toward maintaining the current assignment to avoid oscillation.

Sequencing is needed to move the entire team through the list of tactics in sequence. When the tactic executed by the *active player*, the robot whose role specifies a tactic related to the ball (see Table 1), succeeds then the play transitions *each* role to the next tactic in their relative sequence. Finally, opportunistic behavior accounts for unexpected situations where a very basic action would have a valuable outcome. For example, the play executor evaluates the duration of time and

potential success of each robot shooting immediately. If a robot can shoot quickly enough and with a high likelihood of success, it will immediately switch its behavior to take advantage of the situation. Thus, opportunistic behavior enables plays to have behavior beyond that specified explicitly. As a result, a play can encode a long sequence of complex behavior without encumbering its ability to respond to unexpected short-lived opportunities.

Finally, the play executor checks the play's termination criteria, the completion status of the tactics, and the incoming information from the referee. If the final active tactic in the play's sequence of tactics completes then the play terminates as completed. If the game is stopped by the referee for a goal, penalty, or free kick, the play terminates. The outcome of the play depends upon the condition. Foals and penalty kicks result in a success or failure as appropriate. A free kick is results in a completion or an abortion as appropriate.

3.4 Play Selection

The final detail of the playbook strategy system is the mechanism to select plays, and adapt that selection given experience. Our basic selection scheme uses the applicability conditions for each play to form a candidate list from which one play is selected at random. To adapt play selection, we modify the probability of selecting a play using a weighting scheme. In the next section, we describe this scheme and present experimental results showing the usefulness of a playbook approach and the effectiveness of adaptation.

3.5 Implementation

Here we briefly, due to space considerations, describe our implementation of the play system. Firstly, let us consider the larger issues of how plays fit in within the larger framework. Within the system architecture there are two main control paths (see figure 1). The primary control path flows from vision, through tactics, and to the robots. This path must handle information in a high bandwidth, low latency manner. The second control path flows from vision, through the playbook engine, then through tactics to the robot. In short, the playbook engine controls the robots through the tactics layer. Since the plays generate team behavior through the instantiation of tactics, which have a response time of seconds to tens of seconds, it is not strictly necessary for the playbook engine to operate with high bandwidth, which in CPU limited implementations can be useful. However, in practice, the playbook executor operates at frame rate to ensure that its latency, or response time, when reacting to changing situations is kept as small as possible. This is a critical issue to ensuring real-time team response. Moreover, great effort is exerted to ensure that all algorithms process within a single frame time (33ms). Hence, once accounting for the unavoidable latency incurred through using vision, the tactics and plays work in harmony to produce a new robot response within a single frame. Overall this produces a total system latency of 100ms when responding to events. Of course, the robot response times (e.g. driving to intercept a moving ball) then form an additional latency which is on the order of seconds usually.

The playbook engine consists of a number of interacting C++ classes. At the base are the *PlayRole*, *Play*, and *PlayBook* classes which encapsulate the data and methods for roles, plays and the playbook, respectively. A specialized class, called *PlayAscii* supports the reading of ASCII play files. The key part in the chain though, is the *PlayExecutor* class. This class is responsible for selecting plays, controlling the flow of play execution, and then modifying the play's weights based upon its performance.

The operations of the executor are as described in this section. There are two important aspects to help understand its implementation. Firstly, tactics are sub-classed from a base *Tactics* class. Tactics parameterization, as has been described, occurs through the constructor mechanisms. Thus, the play engine operates by creating with appropriate parameters and destroying tactics, which then in turn take generate robot actions. In addition each tactic comes armed with a range of evaluation functions, which the play executor makes use of in order to assign robots to roles. It is important to note here that tactics themselves are complex mechanisms involving predictive calculations, navigation planning, motion control and in some cases manipulation.

The second aspect to understanding the play executor is that its operation is greatly simplified by the use of predicates which are in turn derived from the output of the tracking system. The latter is encapsulated in the precociously named *World* class. Through Kalman filtering, and higher level processing of the output of the visual stream, the world model provides some fairly high level primitives for forming the predicates described in this paper. Furthermore, these predicates are reasonably, although not totally, free of noise thereby allowing reasonable encoding of plays using these symbolic predicates. As explained, these predicates form the basis for determining when a play is applicable, and also for detecting the success or failure of a play. The case where success or failure is determined by a goal being scored is detectable through the use of a referee box. This is essentially a laptop, which was introduced in 2002, which sends commands to each team computer translating the referee signals ('goal', 'free kick' etc.) into computer readable signals.

4 Playbook Adaptation

Playbook adaptation is the problem of adapting play selection based on past execution to find the dominant play or plays for the given opponent. We perform this adaptation using the execution outcomes of past selected plays. In order to facilitate the compiling of past outcomes into the selection process we associate with each play a weight, $w_{p_i} \in [0, \infty)$. These weights are then normalized over all the set of all applicable plays, A, to define a probability distribution,

$$Pr(\text{selecting } p_i) = \frac{w_{p_i}}{\sum_{p_j \in A} w_{p_j}}.$$

Playbook adaptation involves adjusting the selection weights given the outcome of a play's execution. An adaptation rule is a mapping, $W(\mathbf{w}, p_i, o) \rightarrow [0, \infty)$, from a weight vector, a selected play, and its outcome, to a new weight for that play. These new weights are then used to select the next play.

There are a number of obvious properties for an adaptation rule. All things being equal, more successes or completions should increase the play's weight. Similarly, aborts and failures should decrease the weight. In order for adaptation to have any effect, it also must change weights drastically to make an impact within the short time span of a single game.

The basic rule that we implemented for the RoboCup 2002 competition uses a weight multiplication rule, where each outcome multiplies the play's previous weight by a constant. Specifically, the rule is,

$$W(\mathbf{w}, p_i, o) = C_o w_{p_i},$$

where C_o is the constant associated with the particular outcome. We nominally set these to,

$$C_{\texttt{succeeded}} = 4 \qquad C_{\texttt{completed}} = 4/3$$
$$C_{\texttt{aborted}} = 3/4 \qquad C_{\texttt{failed}} = 1/4.$$

These weight values capture the basic properties described in the above paragraphs. Little work has gone into optimizing these weight selections. Since the opponent is unknown a priori, it seems difficult to know in advance what weight values are best. Rather, our approach has been to select reasonable values, that ensure changes in play selection within the short durations that make up a game ie. two 10 minute halves. In practice, we have found that these weight values are sufficient to ensure modified behavior within a half against a range of opponents of widely varying capabilities and strategies.

4.1 Evaluation

Our strategy system was used effectively during RoboCup 2002 against a wide range of opponents with vastly different strategies and capabilities. Although effective, the nature of robot competitions prevent them from being a systematic evaluation in a controlled, scientific, or statistically significant way. Therefore, we have constructed a number of simplified scenarios to evaluate our play system. These scenarios first compare whether multiple plays are actually necessary, and also examine the usefulness of playbook adaptation.

The scenarios compare the effectiveness of four different offensive plays paired against three different defensive behaviors. To simplify evaluation, only two offensive robots were used against one defensive robot. The robots start in the usual "kick off" position in the center of the field. For each scenario 750 trials were performed in our UberSim SSL simulator [2]. A trial was considered a success if the offense scored a goal within a 20 second time limit, and a failure otherwise. Table 5 lists the specifics of the offensive plays and defensive behaviors. The first two defensive behaviors, and first three offensive plays, formed the core of the playbook used for RoboCup 2002.

Table 6 shows the play comparison results. Each trial is independent, hence the maximum likelihood estimate of the play's success probability is the ratio of successes to trials. Note that there is no static strategy that is optimal. The best strategy depends upon the defensive behavior even in this simplified scenario. In fact, each of the offensive plays is actually the optimal response for some distribution over defense behaviors. These differences are statistically significant with 95% confidence using binomial difference tests.

Offensive Plays

Name	Attacker 1	Attacker 2
Shoot 1	`shoot Aim`	`position_for_rebound`
Shoot 2	`shoot NoAim`	`position_for_rebound`
Screen 1	`shoot Aim`	`mark 0 from_shot`
Screen 2	`shoot NoAim`	`mark 0 from_shot`

Defensive Behaviors

`block`	Positions to block incoming shots
`active_def`	Actively tries to steal ball
`brick`	Defender does not move

Table 5: List of offensive and defensive behaviors tested.

Play	`block`	`active_def`	`brick`
Shoot 1	**72.3%**	49.7%	**99.5%**
Shoot 2	66.7%	57.3%	43.1%
Screen 1	40.8%	59.0%	92.4%
Screen 2	49.2%	**66.0%**	72.0%

Table 6: Play comparison results. For each scenario, the percentage of successes for the 750 trials is shown. The boldfaced number corresponds to the play with the highest percentage of success for each defensive behavior.

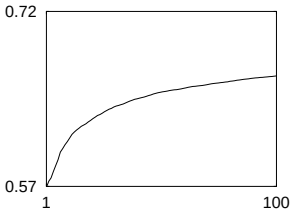
Our results support the notion that play-based strategies are capable of exhibiting many different behaviors with varying degrees of effectiveness. For instance, against a "conservative" blocking defense, the play where a robot takes the time to align itself for a good shot performs the best. On the other hand, against more "aggressive" defenses, the above play performs poorly in comparison. Instead, the play where the robots take less time to aim while the assisting robot attempts to set a screen for the shooter, is more successful.

To explore the effectiveness of playbook adaptation we use an offensive playbook for two robots with all four offensive plays described above against a fixed defender running either `block` or `active_def`. We initially used the algorithm above, but discovered an imperfection in the approach. Due to the strength of the reinforcement for a completed play, it is possible for a moderately successful but non-dominant play to quickly dominate, and remain dominant, in weight. This phenomenon did not occur in competition due to the larger disparity in plays against a given opponent and lower success probabilities. The problem is that there is no normalization in the weight adjustment for plays that have a higher selection probability, which are updated more often. Therefore, we included a normalization factor in the weight updates. Specifically, we used the following rule,

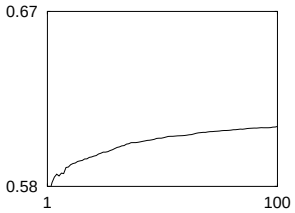
$$W(\mathbf{w}, p_i, o) = \begin{cases} w_{p_i} 2/\Pr(p_i) & \text{if } o = \texttt{Succeeded} \\ w_{p_i} \Pr(p_i)/2 & \text{if } o = \texttt{Failed} \end{cases},$$

where $\Pr(p_i)$ is the probability assigned to p_i according to $\mathbf{w}$.

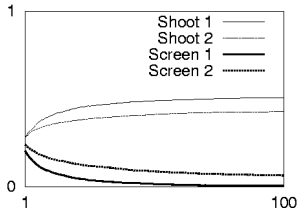
To evaluate the performance, we compare the expected success rate (ESR) of using this adaptation rule against a fixed defensive behavior. We used the results in Table 6 to simulate the outcomes of the various play combinations. All the weights are initialized to 1. Figure 2(a) and (b) show the ESR for play adaptation over 100 trials, which is comparable to a length of a competition (approximately 20 minutes). The



(a) Success Rate v. Block



(b) Success Rate v. Active



(c) Expected Play Probabilities v. Block

Figure 2: (a), (b) show ESR against block and active_def, (c) shows expected play success probabilities against block. These results have all been averaged over 50000 runs of 100 trials.

lower bound on the y-axis corresponds to the ESR of randomly selecting plays and the upper bound corresponds to the ESR of the playbook's best play for the particular defense. Figure 2(c) shows the probabilities of selecting each play over time when running the adaptation algorithm. Clearly, the algorithm very quickly favors the more successful plays.

These results, combined with the RoboCup performances, demonstrate that adaptation can be a powerful tool for identifying successful plays against unknown opponents. Note the contrast here between the use of adaptation to more common machine learning approaches. We are not interested in convergence to an optimal control policy. Rather, given the short time limit of a game, we desire adaptation that achieves good results quickly enough to impact the game. Hence a fast, but non-optimal response is desired over a more optimal but longer acting approach.

Finally, it is worth noting that play adaptation has a different role to tradition machine learning approaches. With the formulation described here, play adaptation does not allow for a team to perform better than the capabilities inherent in the underlying single robot skills (tactics in this case). Rather, it provides a team with an adaptation mechanism that ensures that its performance will be as good as possible, within the limits of the available tactics and the plays encoded in its playbook.

5 Conclusion

In conclusion, we have introduced a novel team strategy engine based on the concept of a play as a team plan, which can be easily defined by a play language. Multiple, distinct plays can be collected into a playbook where mechanisms for adapting play selection can enable the system to improve the team response to an opponent without prior knowledge of the opponent's strategy. The system was fully implemented for our CMDragons robot soccer system and tested at RoboCup 2002, and in the controlled experiments reported here.

References

[1] Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors. *RoboCup 2001: Robot Soccer World Cup V*. Springer Verlag, Berlin, 2002.

[2] Brett Browning and Erick Tryzelaar. Ubersim: A multi-robot simulator for robot soccer. In *Proceedings of AAMAS*, 2003.

[3] James Bruce, Michael Bolwing, Brett Browning, and Manuela Veloso. Multi-robot team response to a multi-robot opponent team. In *ICRA Workshop on Multi-Robot Systems*, 2002.

[4] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, pages 2383–2388, Switzerland, October 2002.

[5] S.S. Intille and A.F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *AAAI-99*, pages 518–525. AAAI Press, 1999.

[6] Itsuki Noda, Shóji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. RoboCup-97: The first robot world cup soccer games and conferences. *AI Magazine*, 19(3):49–59, Fall 1998.

[7] Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *ICAPS-02, Best Paper Award*, Toulouse, France, April 2002.