STP: Skills, tactics and plays for multi-robot control in adversarial environments

Brett Browning, James Bruce, Michael Bowling, and Manuela Veloso
{brettb,jbruce,mhb,mmv}@cs.cmu.edu
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, USA

November 22, 2004

Abstract

In an adversarial multi-robot task, such as playing robot soccer, decisions for team and single robot behavior must be made quickly to take advantage of short-term fortuitous events when they occur. When no such opportunities exist, the team must execute sequences of coordinated action across team members that increases the likelihood of future opportunities. We have developed a hierarchical architecture, called STP, to control an autonomous team of robots operating in an adversarial environment. STP consists of *Skills* for executing the low-level actions that make up robot behavior, *Tactics* for determining what skills to execute, and *Plays* for coordinating synchronized activity amongst team members. Our STP architecture combines each of these components to achieve autonomous team control. Moreover, the STP hierarchy allows for fast team response in adversarial environments while carrying out actions with longer goals. In this article, we present our STP architecture for controlling an autonomous robot team in a dynamic adversarial task that allows for coordinated team activity towards long-term goals, with the ability to respond rapidly to dynamic events. Secondly, we present the sub-component of skills and tactics as a generalized, single-robot control hierarchy for hierarchical problem decomposition with flexible control policy implementation and reuse. Thirdly, we contribute our play techniques as a generalized method for encoding and synchronizing team behavior, providing multiple competing team responses, and for supporting effective strategy adaptation against opponent teams. STP has been fully implemented on a robot platform and thoroughly tested against a variety of unknown opponent teams under in a number of RoboCup robot soccer competitions. We present these competition results as a mechanism to analyze the performance of STP in a real setting.

Keywords: Multi-robot coordination, autonomous robots, adaptive coordination, adversarial task

1 Introduction

To achieve high performance, autonomous multi-robot teams operating in dynamic, adversarial environments must address a number of key challenges. The team must be able to coordinate the activities of each team member towards long-term goals, but also be able to respond in real-time to unexpected situations. Here, real-time means responding at least as fast as the opponent. Moreover, the team needs to be able to adapt its response to the actions of the opponent. At an individual level, the robots must be able to execute sequences of complex actions leading towards long-term goals, but also respond in real-time to unexpected situations. Secondly, each robot must have a sufficiently diverse behavior reportoire and be able to execute these behaviors robustly even in the presence of adversaries so as to make a good team strategy viable. Although these contrasting demands are present in multi-robot [30, 17] and single-robot problems [9, 2, 32], the presence of adversaries compounds the problem significantly. If these challenges are not addressed for a robot team operating in a dynamic environment, the team performance will be degraded. For adversarial environments, where a team's weaknesses are actively exploited by good opponents, the team performance will degrade significantly.

The sheer complexity of multi-robot teams in adversarial tasks, where the complexity is essentially exponential in the number of robots, creates another significant challenge to the developer. Thus, control policy reuse across similar sub-problems, as well as hierarchical problem decomposition, are necessary to make effeciently use of developer time and resources.

Addressing all of these challenges in a coherent, seamless control architecture is an unsolved problem, to date. In this paper, we present a novel architecture, called STP, for controlling a team of autonomous robots operating in a task-driven adversarial environment. STP consists of three main components – *Skills*, *Tactics*, and *Plays* – built within a larger framework providing real-time perception and action generation mechanisms. Skills encode low-level single-robot control algorithms for executing a complex behavior to achieve a short-term, focused objective. Tactics encapsulate what the robot should do, in terms of executing skills, to achieve a specific long-term goal. Plays encode how the team of robots should coordinate their execution of tactics in order to achieve the team's overall goals. We beleive that STP addresses many of the challenges to multi-robot control in adversarial environments. Concretely, STP provides three key contributions. Firstly, it is a flexible architecture for controlling a team of robots in a dynamic, adversarial task that allows for both coordinated actions towards long-term goals, and fast response to unexpected events. Secondly, the skills and tactics component can be decoupled from plays, and supports hierarchical control for individual robots operating within a dynamic team task, potentially with adversaries. Lastly, the play-based team strategy provides a generalized mechanism for synchronizing team actions and providing for a diversity of team behavior. Additionally, plays can be effectively used to allow for strategy adaptation against opponent teams. STP has been fully implemented and extensively validated within the domain of RoboCup robot soccer [23]. In this paper, we detail the development of STP within the domain of RoboCup robot soccer, provide evidence of its performance in real competitions with other teams, and discuss how our techniques apply to more general adversarial multi-robot problems.

This article is structured as follows. In the following section, we begin by describing the problem domain of RoboCup robot soccer within which STP has been developed. Section 3 presents an overview of the STP architecture and its key modules leading to a detailed description of the single robot components of skills and tactics in section 4 and team components of plays in section 5. Section 6 describes the peformance of STP in RoboCup competitions against a variety of unknown opponent teams, and discusses how STP can be improved and applied to other adversarial problem domains. Finally, section 7 presents related approaches to STP, and section 8 concludes the paper.

2 The Robot Soccer Problem

The STP architecture is applicable to an autonomous robot team performing a task in an adversarial, dynamic domain. To concretely explore this problem, we chose RoboCup robot soccer as the test-bed domain. More specifically, we have chosen the Small-Size League (SSL), a division within the RoboCup initiative. In this section, the SSL robot soccer problem is concretely defined along with the challenges it poses. This section also details the specific test-bed, the CMDragons system, used to validate the STP architecture to provide a backdrop for the ensuing sections.

2.1 Small-size RoboCup Robot Soccer League

RoboCup robot soccer is a world-wide initiative designed to advance the state-of-the-art in robot intelligence through friendly competition, with the eventual goal of achieving human-level playing performance by 2050 [23]. RoboCup consists primarily of teams of autonomous robots competing against one another in games of soccer, along with an associated symposium for research discussion. There are a number of different leagues within RoboCup, which are designed to focus on different parts of the overall problem: developing intelligent robot teams. This article is primarily focused on the Small-Size League (SSL).

A SSL game consists of two teams of five robots play soccer on a 2.8m x 2.3m field with an orange golf ball [3]. Each team must be *completely* autonomous for the duration of the game, which typically lasts for two 10-minute halves. Here, autonomy means that there are no humans involved in the decision making cycle while the game is in progress. The teams must obey FIFA-like rules as dictated by a human referee. An assistant referee translates referee commands into a computer-usable format, which is transmitted to each team via RS-232 using a standardized protocol, via a computer running the *RefBox* program [3]. Figure 1 shows the general setup as used by many teams in the SSL. The SSL is designed to focus on team autonomy. Therefore, global vision via overhead cameras and off-field computers, which can communicate with the robots via wireless radio, are allowed to be used.

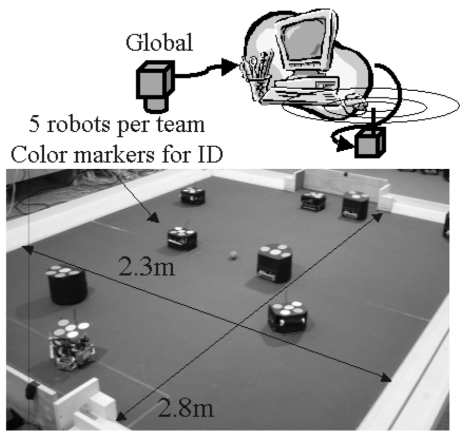

Figure 1: An overview of the CMDragons small-size robot soccer team.

SSL robot soccer involves many research issues. Examples of some of the research challenges include:

- Building *complete*, autonomous control systems for a dynamic task with high-performance;
- Team control in a dynamic environment, and response to an unknown opponent team;
- Behavior generation given real sensor limitations of occlusion, uncertainty, and latency;
- Fast navigation and ball manipulation in a dynamic environment are real-world sensors;
- Fast, robust, low-latency vision, with easy to use calibration routines;
- Robust, high performance robots with specialized mechanisms for ball manipulation.

A typical SSL game is highly dynamic, where ball speeds of 3 to $4m.s^{-1}$ and robots speeds of 1 to $2m.s^{-1}$ are common. With such speeds in a small environment, it becomes critical for information to be translated into action quickly in order for the team to be responsive to sudden events in the world. For example, if a robot kicks a ball at $3.5m.s^{-1}$, a latency of $100ms$ means that the ball will have moved over

$35cm$ before the robots could possibly respond to the observation that the ball had been kicked. High speed of motion and latency impact on control in the following ways:

- Vision, tracking, and modeling algorithms must compromise between the need to filter noise and detect unexpected events in minimum time;
- Prediction mechanisms are required to compensate for latency for effective control;
- Team and single robot control must adapt quickly to dynamic changes.

The last point means that all control decisions need to be recalculated as often as possible to allow the system to react quickly to unexpected events. As a rough guide, the CMDragons system [12] recalculates everything for each frame, at a rate of $30Hz$. Typically, high-level decisions change at a slower rate than low-level decisions. For an approximate guide, a play typically lasts 5-$30s$, while a tactic may operate over a time frame of 1-$30s$, and a skill may operate over a $300ms$-$5s$ time frame. However, any decision at any level can be switched in the minimum time of one frame period ($33ms$) to respond to any large scale dynamic change.

2.2 The CMDragons

Figure 2 shows the major components of the control system developed for our CMDragons SSL team. This architecture is the result of a long series of developments since RoboCup 1997 [37, 36, 35, 10, 12]. Figure 3 shows the robot team members. As shown, the architecture consists of a number of modules beginning with vision and tracking, the STP architecture, navigation and motion control, and finally the robot control software and hardware. We briefly describe each of the non-STP components in the following paragraphs to provide the context for later discussions.

Information passes through the entire system synchronized with incoming camera frames at $30Hz$. Thus a new frame arrives, vision and tracking are run on the processed frame, the resulting information is fed into the world model. The STP architecture is executed, followed by navigation and motion control. The resulting motion command is sent to the robot and the robot executes the command with local control routines.

2.2.1 Perception

Vision is the primary means of perception for the CMDragons team. Everything in the SSL is color coded (see Figure 3), making color vision processing algorithms a natural choice. The ball is orange and the field is green carpet with white lines and white angled walls. Each robot is predominantly black with a yellow or blue circular marker in its center. Depending upon who wins the toss of the coin before the game, one team uses yellow markers while the other uses blue. Each robot typically has another set of markers arranged in some geometric pattern that uniquely identifies the robot and its orientation. Knowledge of an opponents additional markers is usually not available before a game.

In the CMDragons team, images from the camera arrive at a frame rate of 30Hz into an off-field computer. For reference purposes, most of the system described here runs on a 2.1GHz AMD Athlon XP 2700+ system, although a 1.3GHz processor was used previously without any difficulties. Using our fast color vision library, CMVision [11], colored blobs are extracted from each image. The colors are identified based on prior calibration to produce a threshold mapping from pixel values to symbolic color. With knowledge of each robots unique marker layout, high-level vision finds each robot in the image and determines its position and orientation. The position of the ball and each opponent robot is also found. Orientation for opponents

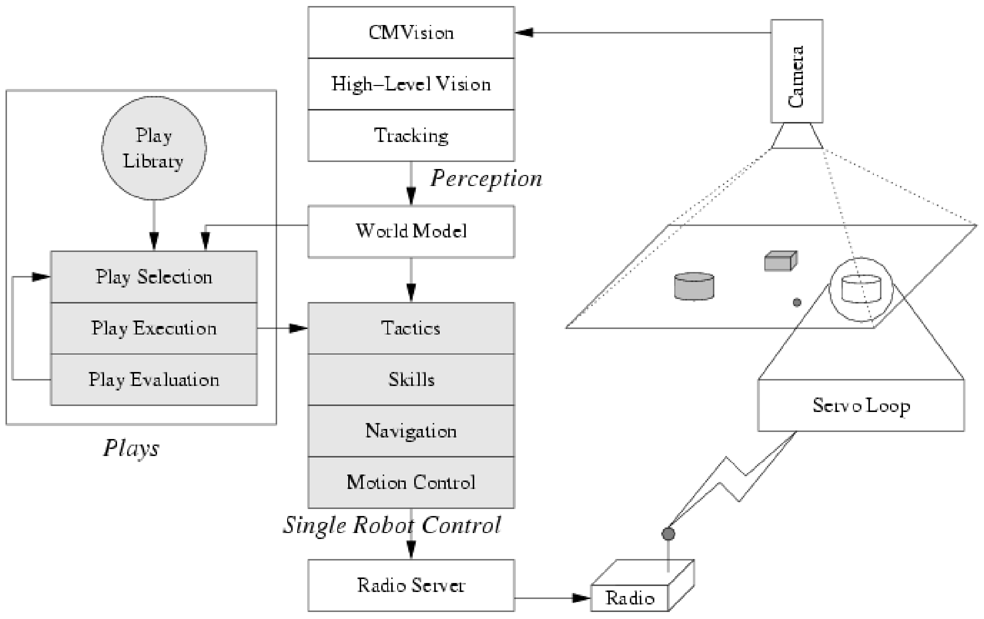

Figure 2: Overview of the CMDragons team architecture.

cannot be found owing to the lack of advance knowledge on their marker layout. The world position of each object is then determined via a parametric camera model learned during game setup. Full details on the vision algorithms can be found in [14].

Filtered position and velocity information is derived using a set of independent Extended Kalman-Bucy Filters (EKBF) for each object (see [12]). As velocity information cannot be derived from each camera image alone, and there is too much noise for frame differentials to be effective, the EKBF's are used to both nullify the effects of noise and intermittency (missing data). Additionally, the EKBF's provide a prediction mechanism through forward modeling which is useful for overcoming latency. In summary, the full vision and tracking module provides estimates of each robot location and orientation, each opponent location, and ball location, with velocities for all eleven objects. Taken together, these estimates provide the robot's belief state about the state of the world.

2.2.2 World Model Belief State

All beliefs about the state of the world, where the robots are etc., are encapsulated in a world belief model. In short, the world model acts as a centralized storage mechanism for beliefs for all layers of the control architecture to use. The belief model contains:

- All perceptual information obtained from the tracker (e.g., robot positions and velocities);

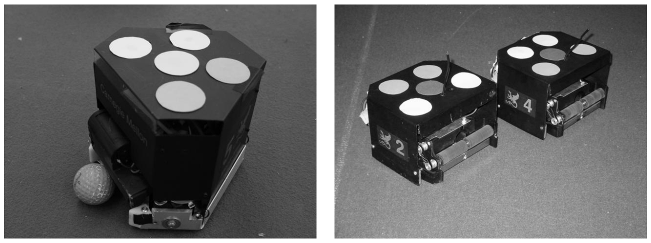
Figure 3: The CMDragons robots. The robot on the left is an OmniBot, while the robots on the right are DiffBots. Each robot fits within an 18cm diameter cylinder that is 15cm tall.

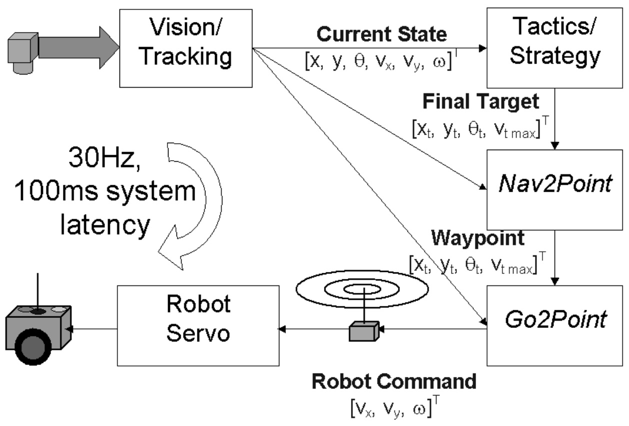
Vision/ Tracking
Current State
$[x, y, \theta, v_x, v_y, \omega]^T$
Tactics/ Strategy
Final Target
$[x_t, y_t, \theta_t, v_{t\,max}]^T$
30Hz, 100ms system latency
Nav2Point
Waypoint
$[x_t, y_t, \theta_t, v_{t\,max}]^T$
Robot Servo
Go2Point
Robot Command
$[v_x, v_y, \omega]^T$

Figure 4: The CMDragons control architecture is based on way-point primitives.

- Game state information derived from the received referee commands;
- Opponent modeling information derived from statistical models of observed opponent behavior;
- High-level predicates derived from the perceived state, such as which team has possession of the ball, is in attack, is in a particular role etc.

Each high-level predicate, is a boolean function of the the tracker and/or game state belief. To account for noise, each boolean function incorporates empirically determined hysteresis to prevent undue oscillation at the decision boundary. These predicates, due to their boolean nature, provide a symbolic representation that is often more useful for making decisions than the raw belief models. For example, deciding whether to run an attacking play or a defensive play.

2.2.3 Navigation and Motion Control Action Interface

The STP architecture consists of team control and individual robot control. Following the common technique of hybrid hierarchical control [19, 33], we have developed lower modules for obstacle-free navigation and motion control. Essentially, these modules provide resources to the robot for generating actions in the world. The resources provide are obstacle-free navigation, motion control, and direct robot commands. Figure 4 shows the control hierarchy.

The navigation module generates a near-optimal, obstacle free path to the goal location using the beliefs stored in the world model. Based on this path, the motion control module calculates a trajectory to achieve a short-term target way-point on the path that does not collide with obstacles. Using this trajectory, a velocity command is issued to the robot hardware to execute.

Due to the dynamic nature of robot soccer, both navigation and motion control are recalculated each frame, for each robot. This places strict computational limitations on each of these modules. We have developed and implemented a fast, randomized path planner [13] based on the Rapidly-exploring Random Trees (RRT's) algorithm [24]. Similarly, we have developed a trapezoidal-based, near-optimal motion control algorithm for quickly generating robot motion commands [12].

2.2.4 Robot Hardware

Each robot is an omni-directional platform capable of spinning while driving in any direction. Each robot is equipped with a ball manipulation device that includes a solenoid actuated 'kicker' and a motorized 'dribbler'. The kicker moves an aluminium plate to contact with the ball, propelling it at speeds of around $3.5-4m.s^-1$. The dribbler is a rubber coated bar that is mounted horizontally at ball height and connected to a motor. As the bar spins against a ball, it causes the ball to spin backwards against the robot thereby allowing the robot to move around effectively with the ball. Each robot has an on-board processor, and runs local velocity-based servo loops using integrated encoder feedback and standard PID control techniques [29]. Additionally, the robot is equipped with an FM radio receiver which it uses to receive movement commands from the external computer.

3 The STP Architecture

This section overviews the STP architecture leading into a detailed discussion of skills, tactics, and plays.

3.1 Goals

The presence of an opponent has many, sometimes subtle, effects on *all* levels and aspects of control. Generating robust behavior that responds to the actions of the opponent is a significant challenge. The challenges for team control are:

1. Execute a temporally extended sequence of coordinated activities amongst team members towards some longer term goal while simultaneously responding as a team to unexpected events both fortuitous and disastrous ones.
2. The ability to respond as a team to the capabilities, tactics, and strategies of the opponent.
3. Execute robust behavior despite sensor limitations and world dynamics.
4. Provide a modular, compact architecture with facilities for easily configuring team play, and for analyzing the performance of the decision making process.

The first and second goals are direct impacts from controlling a team of robots in an adversarial environment. We desire the team control architecture to generate robust behavior that increases the chance of future opportunities against the opponent. Whenever such opportunities arise, whatever the cause, the team must take advantage of this opportunity immediately. Conversely, if an opportunity arises for the opponent team, our team must respond quickly and intelligently to minimize the damage the opponent can cause. Such responsive behavior must occur throughout the architecture. Building responsive team while overcoming the usual limitations of real world sensors, such as latency, noise, and uncertainty, is the major goal of the STP framework.

In robot soccer, robust development is a significant issue. Many teams have gone through bad experiences caused by poor development procedures or facilities. Thus, a good architecture is one that is compact and modular such that changes in one module have a minimal impact on the operation of another module. Given the number of parameters in a complex team architecture, the ability to easily reconfigure those parameters and to analyze the performance of different parameter settings is extremely useful to the development cycle.

3.2 Skills, Tactics and Plays

To achieve the goals of responsive, adversarial team control, we have developed the STP architecture. The key component of STP is the division between single robot behavior and team behavior. In short, team behavior results from executing a coordinated sequence of single robot behaviors for each team member. We now define *plays*, *tactics*, and *skills*, and how they interact for a team of N robots.

A *play*, P, is a fixed team plan which consists of a set of applicability conditions, termination conditions, and N roles, one for each team member. Each role defines a sequence of tactics $T^1, T^2 \ldots$ and associated parameters to be performed by that role in the ordered sequence. Assignment of roles to team members is performed dynamically at run time. Upon role assignment, each robot i is assigned its tactic T_i to execute from the current step of the sequence for that role. Tactics, therefore, form the action primitives for plays to influence the world. The full set of tactics can be partitioned into active tactics and non-active tactics. Active tactics are those involved with ball manipulation. There is only one active tactic amongst the roles per step in the sequence. The successful completion of the active tactic is used to trigger the transition to the next step in the sequence for *all* roles in the play. Plays are discussed in greater detail in section 5.

A *tactic*, T, encapsulates a single robot behavior. Each robot i executes its own tactic as created by the current play P. A tactic T_i determines the skill state machine SSM_i to be executed by the robot i. If the tactic is an active one, it also contains evaluation routines to determine if the tactic has completed. If the skill state machine differs from that executed previously, then execution begins at the first skill in the state machine i.e. S_i. If the skill state machine did not change, then execution continues at the last skill transitioned to. The tactic T_i also sets parameters $SParams_i$ to be used by the executing skill S_i. Thus, skills form the action primitives for tactics.

A *skill*, S, is a focused control policy for performing some complex action. Each skill is a member of one, or more, skill state machines $SSM_1, SSM_2, \ldots$. Each skill S determines what skill it transitions to S' based upon the world state, the time skill S has been executing for, and the executing tactic for that robot. The executing tactics may reset, or change and reset, the executing skill state machine. Each skill can command the robot to perform actions either directly, through motion control, or through navigation. If commanded through navigation, navigation will generate an intermediate, obstacle free way-point for motion control which will then generate a command to send to the robot.

Both skills and tactics must evaluate the world state, in sometimes complex ways, to make useful decisions. For example, some tactics determine the best position to move to in order to receive a pass. Alternatively, some defensive tactics evaluate which opponent robot might move to receive a pass and where to go to prevent the opponent achieving this goal. To prevent unnecessary duplication, and to greater modularize the architecture, we extract these evaluations into an evaluation module which is usable by both tactics and skills. Tactics, skills, evaluations are detailed in section 4.

Plays, tactics, and skills, form a hierarchy for team control. Plays control the team behavior through tactics, while tactics encapsulate individual robot behavior and instantiate actions through sequences of skills. Skills implement the focused control policy for actually generating useful actions. Table 1 shows the main execution algorithm for the STP architecture. The clear hierarchical arrangement of plays for team control, tactics for single robot behavior, and skills for focused control are shown.

```
Process STP Execution
1.  CaptureSensors()
2.  RunPerception()
3.  UpdateWorldModel()
4.  $P \leftarrow$ ExecutePlayEngine()
5.  for each robot $i \in \{1, \ldots, N\}$
6.      $(T_i, TParams_i) \leftarrow$ GetTactic($P, i$)
7.      $(SSM_i, SParams_i) \leftarrow$ ExecuteTactic($T_i, TParams_i$)
8.      if NewTactic($T_i$) then
9.          $S_i \leftarrow SSM_i(0)$
10.     $(command_i, S'_i) \leftarrow$ ExecuteStateMachine($SSM_i, S_i, SParams_i$)
11.     $robot\_command_i \leftarrow$ ExecuteRobotControl($command_i$)
12.     SendCommand($i, robot\_command_i$)
```

Table 1: The main STP execution algorithm.

4 Tactics and Skills for Single Robot Control

Single robot control in the STP architecture consists of tactics and skills. Tactics provide the interface for team control via plays, while skills provide the mechanisms for generating behavior in a compact, reusable way. We begin by describing tactics in greater depth, followed by skills, and finally the evaluation module .

4.1 Tactics

Tactics are the topmost level of single robot control. Each tactic encapsulates a single robot behavior. Each tactic is parameterized allowing for more general tactics to be created which are applicable to a wider range of world states. Through parameterization a wider range of behavior can be exhibited through a smaller set of tactics, making play design easier. Table 2 provides the list of tactics we have implemented for robot soccer. The meaning of each tactic should be reasonably obvious from the tactic name.

During execution, one tactic is instantiated per robot. A tactic, as determined by the executing play, is created with the parameters defined for the play. That tactic then continues to execute until the play transitions to the next tactic in the sequence. As described above, each tactic instantiates action through the skill layer. In short, the tactics determine which skill state machine will be used, and sets the parameters for executing those skills. Example parameters include target way-points, target points to shoot at, opponents to mark, and so on. Different tasks may use many of the same skills, but provide different parameters to achieve the different goals of the tactic. The shooting and passing tactics are good examples. The skills executed by the two are very similar, but the resulting behavior can be quite different due to the different parameter assignments. Finally, each tactic may store any local state information it requires to execute appropriately.

Table 3 shows the algorithm for the `shoot` tactic used to kick the ball at the goal or towards teammates for one-shot deflections on goal. Not shown are the conditioning of the tactic decision tree on the parameters specified by the active play. In this case, the play can only disable deflection decisions. The tactic consists of evaluating the options of shooting directly on goal, or shooting to a teammate to deflect or kick on goal in a so-called one-shot pass. Each option is assigned a score which, loosely, defines a likelihood of success.

```
Active Tactics
shoot (Aim | Noaim | Deflect ⟨role⟩)
steal [⟨coordinate⟩]
clear
active_def [⟨coordinate⟩]
pass ⟨role⟩
dribble_to_shoot ⟨region⟩
dribble_to_region ⟨region⟩
spin_to_region ⟨region⟩
receive_pass
receive_deflection
dribble_to_position ⟨coordinate⟩ ⟨theta⟩
position_for_start ⟨coordinate⟩ ⟨theta⟩
position_for_kick
position_for_penalty
charge_ball

Non-Active Tactics
position_for_loose_ball ⟨region⟩
position_for_rebound ⟨region⟩
position_for_pass ⟨region⟩
position_for_deflection ⟨region⟩
defend_line ⟨coordinate-1⟩ ⟨coordinate-2⟩ ⟨min-dist⟩ ⟨max-dist⟩
defend_point ⟨coordinate-1⟩ ⟨min-dist⟩ ⟨max-dist⟩
defend_lane ⟨coordinate-1⟩ ⟨coordinate-2⟩
block ⟨min-dist⟩ ⟨max-dist⟩ ⟨side-pref⟩
mark ⟨orole⟩ (ball | our_goal | their_goal | shot)
goalie
stop
velocity ⟨vx⟩ ⟨vy⟩ ⟨vtheta⟩
position ⟨coordinate⟩ ⟨theta⟩
```

Table 2: List of tactics with their accepted parameters.

Much of the operation of determining the angles to shoot at and generating the score is pushed into the evaluation module, described in section 4.3.

The tactic, indeed nearly all tactics, make use of additive hysteresis in the decision making process. Hysteresis is a necessary mechanism to prevent debilitating oscillations in the selected choice from frame to frame. Each action in the `shoot` tactic, as with any other tactic, takes a non-negligible period of time to perform that is substantially greater than a single decision cycle at $30Hz$. With the dynamics of the environment further complicated by occlusion, noise, and uncertainty, its is often the case that two or more choices will oscillate over time in terms of its score. Without hysteresis, there will be corresponding oscillations in the action chosen. The end result is often that the robot will oscillate between distinctly different actions and effectively be rendered immobile. The physical manifestation of this behavior, ironically, is that the robot appears to 'twitch' and be 'indecisive'. In most robot domains, such oscillations will degrade performance. In adversarial domains like robot soccer, where it is important to carry out an action *before* the opponent can respond, such oscillates completely destroy. Hysteresis provides a usable, easily understandable, mechanism for preventing such oscillations and is used pervasively throughout the STP architecture.

```
Tactic Execution shoot($i$):
1.  $bestscore \leftarrow 0$
2.  $(score, target) \leftarrow$ evaluation.aimAtGoal()
3.  if (was kicking at goal) then
4.      $score \leftarrow score +$ HYSTERESIS
5.  $SParam_i \leftarrow$ setCommand(MoveBall, $target$, KICK_IF_WE_CAN)
6.  $bestscore \leftarrow score$

7.  foreach teammate $j$ do
8.      if (evaluation.deflection($j$) $>$ THRESHOLD) then
9.          $(score, target) \leftarrow$ evaluation.aimAtTeammate($j$)
10.         if (was kicking at player $j$) then
11.             $score \leftarrow score+$ HYSTERESIS
12.         if ($score > bestscore$) then
13.             $SParam_i \leftarrow$ setCommand(MoveBall, $target$, KICK_IF_WE_CAN)
14.             $bestscore \leftarrow score$

15. if (No target found OR $score <$ THRESHOLD) then
16.     $target \leftarrow$ evaluation.findBestDribbleTarget()
17.     $SParam_i \leftarrow$ SetCommand(MoveBall, $target$, NO_KICK)
```

Table 3: Algorithm for the `shoot` tactic for shooting on goal directly or by one-shot passes to teammates. Each action is evaluated and assigned a score. The action with the best score better than the score for the previously selected action, is chosen and its target passed to the running skill. The skill state machine used is the $MoveBall$ state machine.

4.2 Skills

Most tactics require the execution of a sequence of recognizable skills, where the actual sequence may depend upon the world state. An example skill sequence occurs when a robot tries to dribble the ball to the center of the field. In this case, the robot will *(a)* go to the ball, *(b)* get the ball onto its dribbler, *(c)* turn the ball around if necessary, then *(d)* push the ball toward the target location with the dribbler bar spinning. A different sequence would be required if the ball were against the wall, or in the corner. Additional skills would be executed, such as pulling the ball off the wall, in order to achieve the final result.

In our other work, we have developed a hierarchical behavior based architecture, where behaviors form a state machine with transitions conditioned on the observed state and internal state [25]. Although we make no use of the hierarchical properties of the approach here, we do make use of the state machine properties to implement the sequence of skills that make up each tactic. Each skill is treated as a separate behavior and forms a unique state in the state-machine. In contrast to tactics, which execute until the play transitions to another tactic, each skill transitions to itself or another skill at each time step.

Each skill consists of three components: sensory processing, command generation, and transitions. Sensory processing consists of using or generating the needed sensory predicates from the world model. Commonly used sensors are generated once per frame, ahead of time, to prevent unnecessary duplication of effort. Command generation consists of determining the action for the robot to perform. Commands can be instantiated through the navigation module or motion control. In some cases, commands are sent directly to the robot. Transitions define the appropriate next skill that is relevant to the execution of the tactic. Each skill can transition to itself or another skill. Transitions are conditioned on state variables set by the tactics or state machine variables, such as the length of time the active skill has been running. This makes it possible to use the same skill in multiple sequences. A skill can be used for different tactics, or in different circumstances for the same tactic. Thereby allowing for skill reuse and the minimizing of code duplication.

Table 4 shows our algorithm for the `driveToGoal` skill used to drive the ball toward the ball towards the desired target, which is continually adjusted by the tactic as execution cycles. The skills first determines what skill it will transition to. If no skill is found, it transitions to itself. The decision tree shows conditioning on the active state machine, `MoveBall` in this case, and conditioning upon the active tactic. Decisions are also made using high level predicates, for example $ball\_on\_front$, derived from the tracking data by the world model. References to the world are not shown to aid clarity.

```
Skill Execution DriveToGoal($i$):
1. if ($SSM_i$ = MoveBall AND $ball\_on\_front$ AND $can\_kick$ AND $shot\_is\_good$) then
2.     Transition(Kick)
3. if ($ball\_on\_front$ AND $ball\_is\_visible$) then
4.     Transition(GotoBall)
5. if ($robot\_distance\_from\_wall < $ THRESHOLD AND $robot_stuck$) then
6.     Transition(SpinAtBall)

   Command generation
7. $command_i.navigate \leftarrow true$
8. $command_i.target \leftarrow$ calculateTarget()
```

Table 4: The `DriveToGoal` skill which attempts to push the ball towards the desired direction to kick. Shown is the transitions decision tree, which includes conditioning on the active tactic, the active state machine, and predicates derived from the world model. The command generation calculations are simplified here to aid clarity, but require a number of geometric calculations to determine the desired target point.

4.3 Evaluation Module

There are numerous computations about the world that need to be performed throughout the execution of plays, tactics, and skills in order to make good decisions. Many of these computations are evaluations of different alternatives, and are often used numerous times. Aim evaluation is a good example, as the same evaluation of alternatives is called at least 24 times during a single cycle of execution! We combine all of these evaluations into a single module. There are three classes of evaluations that occur; aiming, defense, and target positions.

Aim Evaluations. Aiming evaluations determine the best angle for the robot to aim toward to kick the ball through a specified line segment while avoiding a list of specified obstacles. Using the world model, the aim evaluations determine the different open angles to the target. It then chooses the largest open angle with additive hysteresis if the last chosen angle, assuming there is one, is still a valid option. The use of a line segment as the target allows the same evaluation to be used for aiming at the goal, for opponents aiming at our goal, as well as for passes and deflections to teammates or from opponents to their teammates.

Defensive Evaluations. Defensive evaluations determine where the robot should move to best defend a specified point or line segment. Although similar to target position evaluations, the technique used is quite different. There are a number of different variations of defensive evaluations for defending lines, points, or defending along a given line. Each evaluation uses similar techniques, but the point chosen, and hence the behavior generated vary and are useful in different situations.

The most commonly used defensive evaluation is line defenses. For line defenses, the evaluation attempts to blend between choosing a defensive point that is good if the ball could be kicked at any angle from its observed location and a point to intercept the ball if it were to remain moving at its current velocity. We first create a linear Gaussian to describe the desirability of each point on the line for defending against a static kick. The Gaussian is centered on the point that, when accounting for the robot size, equalizes the time it would take for the robot to move to block a shot at either end of the defended segment. A second linear Gaussian is generated by predicting the ball motion forward in time to where it crosses the defended line, and calculating the corresponding tracker uncertainty projected onto this line. Essentially, the faster the ball is kicked, the more certain its crossing point, which results in a much narrower, taller Gaussian. In addition, obstacles along the trajectory can add also add substantial uncertainty into the interception Gaussian. When these two Gaussian functions are multiplied, the result represents a smooth blending between the two alternatives. Generally, the static kick Gaussian dominates, but as the ball is kicked faster toward the defense line the interception Gaussian pushes the defender to intercept the current trajectory. Such a smooth shift is desirable to avoid having to develop techniques for deciding between intercepting or defending, and the corresponding hysteresis that would be required.

Target Position Evaluation. The final type of evaluation determines the best target position to achieve a given task. Examples include the best position to receive a deflection, the best position to acquire a loose ball, the best location to dribble toward to get a shot on goal or to pass to a teammate. In each case, there is a range of competing criteria that the evaluation ideally would optimize that can often be represented as an objective function of some kind. For example, to receive a pass for a shot on goal the robot needs to get into a position that gives it a clear shot on goal, a reasonable deflection angle so it has an opportunity to receive and kick the ball, and a clear shot to its teammate with ball possession. Clearly, one could write an objective function to describe this problem, and attempt to find the optimal solution. This approach is problematic due to the computational constraint that only a fraction of the processor is available for this task, and it needs to be repeated many times during a single execution cycle. Additionally, the dynamics of the environment, combined with sensing noise, mean that the optimal point will invariably be unstable over time. Thus, the robot will never stabilize, which is essential for situations like receiving a pass as its teammate needs a steady target. In many cases, however, if we consider near-optimal values, reasonably stable sets form over extended periods. Thus, we require an evaluation method with low computational requirements find quasi-static, near-optimal locations.

We have taken a sample-based approach to this problem. For each evaluation, a series of points are generated randomly drawn uniformly from the region of space of interested specified in the evaluation call. The objective function is evaluated at each point, and the best value is recorded. If there was a point chosen previously, its value is calculated and if it is within α standard deviations of the score of the best point, for some defined α, it is again selected as the target point. If there was no target point previously, or it is no longer α-optimal, the best point is chosen as the target. Thus, the α value imparts a hysteresis effect as used in the other evaluations.

5 Plays for Multi-Robot Team Control

The final component of the STP architecture are plays. Plays form the highest level in the control hierarchy providing strategic level control of the entire team. The strategic team problem involves selecting each robot's behavior in order to achieve team goals, given a set of tactics, which are effective and parameterized individual robot behaviors. We build team strategy around the concept of a *play* as a team plan, and the concept of a *playbook* as a collection of team plans. We first explore the goals for the design of a team strategy system and then explore how plays and playbooks achieve these goals.

5.1 Goals

Obviously the main criterion for a team strategy system is performance. A single, monolithic team strategy that maximizes performance, though, is impractical. In addition, there is not likely to be a single optimal strategy independent of the adversary. Instead of focusing directly on team performance, we enumerate a set of six simpler goals, which we believe are more practical and lead to strong overall team performance:

1. Coordinated team behavior,
2. Temporally extended sequences of action (deliberative),
3. Inclusion of special purpose behavior for certain circumstances,
4. Ease of human design and augmentation,
5. Ability to exploit short-lived opportunities (reactive), and
6. 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, the language must be human readable to make play design and modification simple. These goals also require a powerful system capable of executing the complex behaviors the plays describe. 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 overall behavior over time. Notice that the strategy system requires both deliberative and reactive reasoning. The dynamic environment makes a strictly deliberative system unlikely to be able to carry out its plan, but the competitive nature often requires explicitly deliberative sequences of actions in order to create scoring opportunities.

We first introduce our novel play language along with the coupled play execution system. We then describe how playbooks can provide multiple alternative strategies for playing against the unknown opponent.

5.2 Play Specification

A play is a multi-agent plan, i.e., a joint policy for the entire team. Our definition of a play, therefore, shares many concepts with classical planning. A play consists of four main components:

- Applicability conditions,
- Termination conditions,
- Roles, and
- Execution details.

```
PLAY Naive Offense

APPLICABLE offense
DONE aborted !offense

ROLE 1
 shoot A
 none
ROLE 2
 defend_point {-1400 250} 0 700
 none
ROLE 3
 defend_lane {B 0 -200} {B 1175 -200}
 none
ROLE 4
 defend_point {-1400 -250} 0 1400
 none
```

Table 5: A simple example of a play.

Applicability conditions specify when a play can be executed and are similar to planning operator preconditions. Termination conditions define when execution is stopped and are similar to an operator's effects, although they include a number of possible outcomes of execution. The roles describe the actual behavior to be executed in terms of individual robot tactics. The execution details can include a variety of optional information that can help guide the play execution system. We now look at each of these components individually.

5.2.1 Applicability Conditions

The conditions for a play's applicability can be defined as any logical formula of the available state predicates. The conditions are specified as a logical DNF using the `APPLICABLE` keyword, with each disjunct specified separately. In the example play in Table 5, the play can only be executed from a state where the `offense` predicate is true. The `offense` predicate is actually a fairly complex combination of the present and past possession of the ball and its present and past position on the field. Predicates can be easily added and Table 6 lists the current predicates used by our system. Note that predicates can also take parameters, as in the case of `ball_x_gt` *X*, which checks if the ball is over the distance *X* down field.

Like preconditions in classical planning, applicability conditions restrict when a play can be executed. By constraining the applicability of a play, one can design special purpose plays for very specific circumstances. An example of such a play is shown in Table 7. This play uses the `ball_in_their_corner` predicate to constrain the play to be executed only when the ball is in a corner near the opponent's goal. The play explicitly involves dribbling the ball out of the corner to get a better angle for a shot on goal. Such a play only really makes sense when initiated from the play's applicability conditions.

5.2.2 Termination Conditions

Termination conditions specify when the play's execution should stop. Just as applicability conditions are related to operator preconditions in classical planning, termination conditions are similar to operator effects.

Play predicates	
offense	our_kickoff
> defense	their_kickoff
> their_ball	our_freekick
> our_ball	their_freekick
> loose_ball	our_penalty
> ball_their_side	their_penalty
> ball_our_side	ball_x_gt *X*
ball_midfield	ball_x_lt *Y*
ball_in_our_corner	ball_absy_gt *Y*
ball_in_their_corner	ball_absy_lt *Y*
nopponents_our_side *N*	

Table 6: List of state predicates.

```
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 } { B 700 800 } 300}
    shoot A
    none
ROLE 2
    block 320 900 -1
    none
ROLE 3
    position_for_pass { R { B 1000 0 } { B 700 0 } 500 }
    none
ROLE 4
    defend_line { -1400 1150 } { -1400 -1150 } 1100 1400
    none
```

Table 7: A special purpose play that is only executed when the ball is in an offensive corner of the field.

Unlike classical planning, though, there is too much uncertainty in execution to know the exact outcome of a particular play. The termination conditions list possible outcomes and associate a *result* with each possible outcome. The soccer domain itself defines a number of stopping conditions, e.g., the scoring of a goal or the awarding of a penalty shot. The play's termination conditions are in addition to these and allow for play execution to be stopped and a new play initiated even when the game itself is not stopped.

Termination conditions, like applicability conditions, use logical formulas of state predicates. In addition to specifying a conjunction of predicates, a termination condition also specifies the result of the play if the condition becomes true. In the play specification, they are delineated by the `DONE` keyword, followed by the result, and then the list of conjunctive predicates. Multiple `DONE` conditions can be specified and are interpreted in a disjunctive fashion. In the example play in Table 5, the only terminating condition, beside the default soccer conditions, is if the team is no longer on offense ("!" is used to signify negation). The play's result is then "aborted".

The results for plays are one of: succeeded, completed, aborted, and failed. These results are used to evaluate the success of the play for the purposes of reselecting the play later. This is the major input to the team adaptation system, which we describe later. Roughly speaking, we use results of succeeded and failed to mean that a goal was scored, or some other equivalently valuable result, such as a penalty shot. the completed result is used if the play was executed to completion. For example, in the play in Table 5, if a robot was able to complete a shot, even if no goal was scored, the play is considered completed. In a defensive play, switching to offense may be a completed result in the `DONE` conditions. The aborted result is used when the play was stopped without completing.

Besides `DONE` conditions, there are two other ways in which plays can be terminated. The first is when the sequence of behaviors defined by the play are executed. As we mentioned above, this gives the play a result of `completed`. This will be described further when we examine the play execution system. The second occurs when a play runs for a long time with no other termination condition being triggered. When this occurs the play is terminated with an `aborted` result and a new play is selected. This allows the team to commit to a course of action for a period of time, but recognize that in certain circumstances a particular play may not be able to progress any further.

5.2.3 Roles

As plays are multi-agent plans, the main component are the roles. Each play has four roles, one for each non-goalie robot on the field. A role consists of a list of behaviors for the robot to perform in sequence. In the example play in Table 5, there is only a single behavior listed for each role. These behaviors will simply be executed until one of the termination conditions apply. In the example play in Table 7, the first role has two sequenced behaviors. In this case the robot will dribble the ball out of the corner. After the first tactic finishes, the robot filling that role will switch to the `shoot` tactic and try to manipulate the ball toward the goal.

Sequencing also requires coordination, which is a critical aspect of multi-agent plans. Coordination in plays requires all the roles to transition simultaneously through their sequence of behaviors. For example, consider the more complex play in Table 8. In this play, one player is assigned to pass the ball to another player. Once the pass behavior is completed *all* the roles transition to their next behavior, if one is defined. So, the passing player will switch to a mark behavior, and the target of the pass will switch to a behavior to receive the pass, after which it will switch to a shooting behavior.

Roles are not tied to any particular robot. Instead, they rely on the play execution system to do this role assignment. The order of the roles presented in the play act as hints to the execution system for filling the roles. Roles are always listed in order of priority. The first role is always the most important and usually

```
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 { 1000 0 } { 700 0 } 500 }
 receive_pass
 shoot A
 none
ROLE 4
 defend_line { -1400 1150} {-1400 -1150} 1000 1400
 none
```

Table 8: A complex play involving sequencing of behaviors.

involves some manipulation of the ball. This provides the execution system the knowledge needed to select robots to perform the roles and also for role switching when appropriate opportunities present themselves.

Tactics in Roles. The different behaviors that can be specified by a role are the individual robot tactics that were discussed in Section 4.1. As mentioned, these tactics are highly parameterized behaviors. For example, the `defend_point` tactic takes a point on the field and a minimum and maximum range. The tactic will then position itself between the point and the ball, within the specified range. By allowing for this large degree of parameterization the different behaviors can be combined into a nearly infinite number of play possibilities. The list of parameters accepted by the different tactics is shown in Table 2.

Coordinate Systems. Many of the tactics take parameters in the form of "coordinates" or "regions". These parameters can be specified in a variety of coordinate systems allowing for added flexibility in specifying plays in general terms. We allow coordinates to be specified either as absolute field position or ball relative field positions. In addition, the positive y-axis can also be specified to depend on the side of the field that the ball is on, the side of field that the majority of the opponents are on, or even a combination of these two factors. This allows tremendous flexibility in the specification of the behaviors used in plays. Regions use coordinates to specify non-axis aligned rectangles as well as circles. This allows, for example, a single play to be general with respect to the side of the field and position of the ball.

5.2.4 Execution Details

The rest of the play specification are execution details, which amount to providing hints to the execution system about how to execute the play. These optional components are: timeout and opponent roles. The timeout overrides the default amount of time a play is allowed to execute before aborting the play and selecting a new play.

Opponent roles allow robot behaviors to refer to opponent robots in defining their behavior. The play in Table 8 is an example of this. The first role, switches to marking one of the opponents after it completes the pass. The exact opponent that is marked depends upon which opponent was assigned to opponent Role 0. Before the teammate roles are listed, opponent roles are defined by simply specifying a selection criteria for filling the role. The example play uses the `closest_to_ball` criterion, which assigns the opponent closest to the ball to fill that role, and consequently be marked following the pass. Multiple opponent roles can be specified and they are filled in turn using the provided criterion.

5.3 Play Execution

The play execution module is responsible for actually instantiating the play into real robot behavior. That is, the module must interpret a play by assigning tactics to actual robots. This instantiation consists of key decisions: role assignment, role switching, sequencing tactics, opportunistic behavior, and termination.

Role assignment uses tactic-specific methods for selecting a robot to fill each role, in the order of the role's priority. The first role considers all four field robots as candidates to fill the role. The remaining robots are considered to fill the second role, and so on. Role switching is a very effective technique for exploiting changes in the environment that alter the effectiveness of robots fulfilling roles. The play executor handles role switching using the tactic-specific methods for selecting robots, using a bias toward the current robot filling the role. Sequencing is needed to move the entire team through the sequence of tactics that make up the play. The play executor monitors the current *active player*, i.e., the robot whose role specifies a tactic related to the ball (see Table 2). When the tactic succeeds, the play is transitioned to the next tactic in the sequence of tactics, for *each* role. Finally, opportunistic behavior accounts for changes in the environment 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 an opportunistic behavior can be executed quickly enough and with a high likelihood of success, then the robot immediately switches its behavior to take advantage of the situation. If the opportunity is then lost, the robot returns to executing its role in the play.

The play executor algorithm provides basic behavior beyond what the play specifies. The play executor, therefore, simplifies the creation of plays, since this basic behavior does not need to be considered in the design of plays. The executor also gives the team robustness to a changing environment, which can cause a play's complex behavior to be no longer necessary or require some adjustment to the role assignment. It also allows for fairly complex and chained behavior to be specified in a play, without fear that short-lived opportunities will be missed.

The final consideration of play execution is termination. We have already described how plays specify their own termination criteria, either through predicates or a timeout. The executor checks these conditions, and also checks whether the play has completed its sequence of behaviors, as well as checking incoming information from the referee. If the final active tactic in the play's sequence of tactics completes, then the play is considered to have completed and is terminated. Alternatively, the game may be stopped by the referee to declare a penalty, award a free kick, award a penalty kick, declare a score, and so on. Each of these conditions terminates the play, but also may effect the determined outcome of the play. Goals are always considered successes or failures, as appropriate. Penalty kicks are also considered play successes and failures. A free kick for our team deems the play as completed, while a free kick for the opponent sets the play outcome to aborted. Play outcomes are the critical input to the play selection and adaptation system.

5.4 Playbook and Play Selection

Plays define a team plan. A playbook is a collection of plays, and, therefore, provides a whole range of possible team behavior. Playbooks can be composed in a number of different fashions. For example, one could insure that for all possible game states there exists a single applicable play. This makes play selection simple since it merely requires executing the one applicable play from the playbook. A more interesting approach is to provide multiple applicable plays for various game states. This adds a play selection problem, but also adds alternative modes of play that may be more appropriate for different opponents. Multiple plays also give options from among which adaptation can select. In order to support multiple applicable plays, a playbook also associates a weight with each play. This weight corresponds to how often the play should be selected when applicable.

Play selection, the final component of the strategy layer, then amounts to finding the set of applicable plays and selecting one based on the weights. Specifically, if $p_{1\dots k}$ are the set of plays whose applicability condition are satisfied, and w_i is their associated weight, then p_j is selected with probability,

$$Pr(p_j|\mathbf{w}) = \frac{w_j}{\sum_{i=1}^{k} p_i}.$$

Although these weights can simply be specified in the playbook and left alone, they also are the parameters that can be adapted for a particular opponent. We use a weighted experts algorithm (e.g., Randomized Weighted Majority [26] and Exp3 [4]) tailored to our specific domain to adapt the play weights during the course of the game. The weight changes were based on the outcomes from the play execution. These outcomes include obvious results such as goals and penalty shots, as well as the plays' own termination conditions and timeout factors. These outcomes are used to modify the play weights so as to minimize the play selection regret, i.e., the success that could have been achieved if the optimal play had been known in advance less the actual success achieved. This adaptation is described elsewhere in more detail [7].

5.5 Achieving Our Goals

Our play-based strategy system, achieves all six goals that we set out in Section 5.1. Sequences of synchronized actions provide a mechanism for coordinated team behavior, as well as deliberative actions. Applicability conditions allow for the definition of special purpose team behavior. The play execution system handles moments of opportunity to allow for the team to have a reactive element. Incorporating all of this into a human readable text format makes adding and modifying plays quite easy. Finally, the ability to assign outcomes to the execution of plays is also the key capability used to adapt the weights used in play selection, achieving the final goal of a strategy system.

6 Results and Discussion

RoboCup competitions provide a natural method for testing and evaluating techniques for single robot and team control against a range of unknown opponents of varying capabilities and strategies. Indeed, this is major focus the competitions. The STP architecture has been evolved through feedback from competitions.

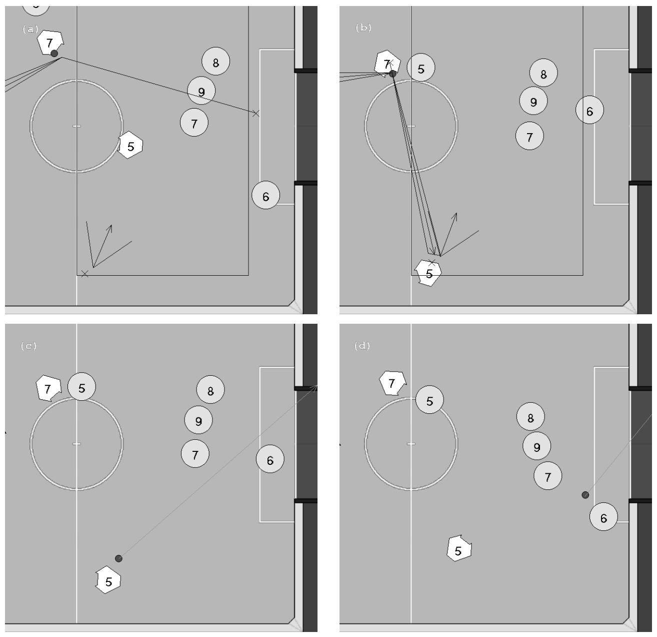

Figure 5: Example of a deflection goal against ToinAlbatross from Japan. The dark lines show debugging output from the tactics and the light line shows the tracked ball velocity. Image (a) shows the the shooting robot unable to take a shot, robot 5 begins moving to a good deflection point. Image (b) shows the kicker lined up and its target zone on robot 5. Image (c) and (d) show the kick and resulting deflection to score a goal. The entire sequence takes less than one second.

Here we mainly report on results derived from the RoboCup 2003 competition, but include anecdotal results from:

- RoboCup 2003, held in July in Padua, Italy. International competition with 21 competitive teams. CMDragons finished 4^{th}. See *http:/www.robocup2003.org*
- RoboCup American Open 2003, held in May in Pittsburgh, USA. Regional competition open to American continent teams. Included 10 teams from US, Canada, Chile, and Mexico. CMDragons won 1^{st} place. See *http://www.americanopen03.org*
- RoboCup 2002, held in June in Fukuoka, Japan. International competition with 20 competitive teams. CMDragons were quarter finalists. See *http://www.robocup2002.org*

6.1 Team Results

Overall, the STP architecture achieves the goals outlined in section 3.1. Using it, our team is able to respond quickly to unexpected situations while carrying out coordinated actions that increase the likelihood of future opportunities. The system is able to execute complex plays involving multiple passes and dribbling, however, due to the risk of loosing the ball, real game plays do not exceed dribbling with one pass for a deflection on goal or a one-shot pass on goal. A one-shot is where one robot passes to another, which then takes a shot on goal. Indeed, such one-shots were responsible for a number of goals. Figure 5 shows an example from the game against ToinAlbatross from Japan.

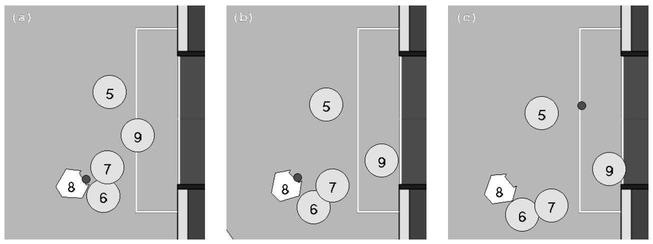

Figure 6: Example of opportunism leading to a goal. Shown is a log sequence from RoboCup 2003 against RoboDragons. The robot gets the ball in image (a). Unexpectedly, a gap opens on goal. The robot moves and shoots ((b) and (c)) to score. The entire sequence takes 15 frames, or 0.5 seconds.

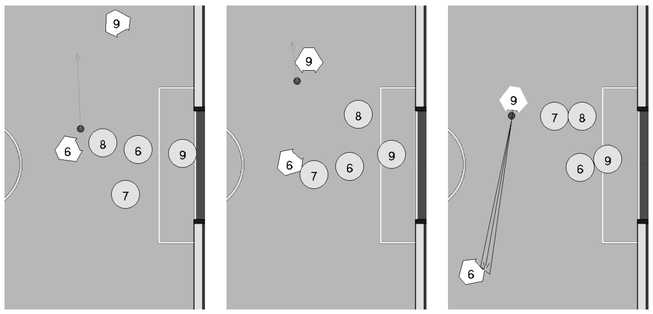

Figure 7: Example of role switching. Here the first robot is the active player, but the ball rolls too fast away from it. The second player smoothly takes over this task, while the first player moves out to receive a deflection. Taken from the game against RoboDragons.

The STP architecture is responsive to opportunistic events, both fortuitous ones and negative ones. Figures 6 shows an example of an opportunistic event occurring during an attacking maneuver against RoboDragons from Japan. The result was a goal, which would not have occurred had the architecture persisted with its team plan. It is interesting to note that the whole episode occurs in less than one second. Figure 7 shows the effectiveness of dynamic role switching during a play, which results in smoother execution of the play.

The architecture is modular and reconfigurable. As an example of this aspect, at the RoboCup 2003 competition we completely rewrote the playbook used by the team during the round robin phase. Modularity helps in making changes while minimizing the impact on the rest of the system. Reconfigurability is achieved through the play language, and use of configuration files to specify parameters for tactics and skills.

To demonstrate the need for different plays, and implicitly the need for different tactics to enable the implementation of a range of different plays. We compared the results of the play weights after the first half for two different games. Figures 9 and 10 show the weights at the end of the first half for the game against ToinAlbatross from Japan and Field Rangers from Singapore, respectively. The weights and selection rates indicate the successfulness of each play. Different strategies are required to play effectively against the different styles of each opponent. The different in play weights clearly shows this. We therefore draw the conclusion that a diversity of tactics, and correspondingly a diversity of plays, is a useful tool for adversarial environments.

Play	weight	Sel	Sel %
o1_deep_stagger	0.021	6	10.3%
o1_points_deep	2.631	11	19.0%
o2_deflection_deep	0.280	40	69.0%
o1_points_deep_deflections	0.015	1	1.7%

Table 9: Offensive weights at the end of the first half for game against ToinAlbatross

Play	weight	Sel	Sel %
o1_deep_stagger	1.080	23	50.00%
o1_points_deep	0.098	2	4.35%
o2_deflection_deep	1.123	17	36.96%
o1_points_deep_deflections	0.657	4	8.70%

Table 10: Offensive weights at the end of the first half for game against Field Rangers

6.2 Single Robot Results

Figure 8 shows a sequence of frames captured from the log of the game against RoboDragons from Japan. The robot shown is executing the shoot tactic, and progresses through a series of skills determined by the progression of world state. Given different circumstances, say if the ball were against the wall or in the open, the sequence of executed skills would be different. As with the play opportunism, the entire sequence occurs in only a few seconds.

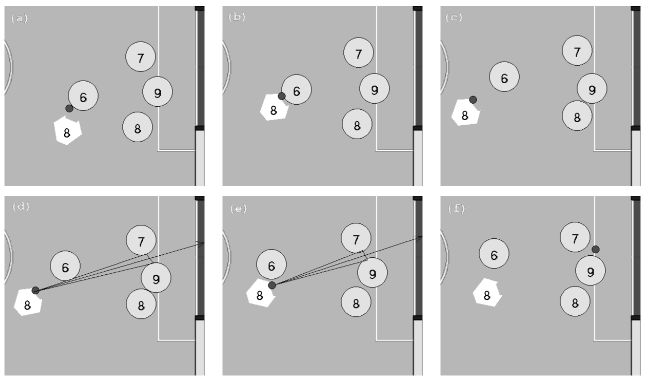

Figure 8: An example shoot sequence taken from the RoboCup 2003 round robin game of CMDragons'03 vs RoboDragons. The robot first executes the `steal_ball` skill (image (a) and (b)), followed by `goto_ball` (image (c)). Once the ball is safely on the robot's dribbler, it begins `drive_to_goal`, image (d), to aim at the selected open shot on goal or to drive to a point where it can take the shot. Upon being in position to take a good shot (image (e)), it kicks leading to a scored goal.

Given the wide range of world states that occur during a game, and the need to execute different skill sequences for different world states, it becomes difficult to analyze the performance of the skill state machine. Consequently, it becomes difficult to determine how to improve its performance for future games. We have developed a number of logging techniques to aid in this analysis. Our logging techniques take three forms. During development and game play, we record statistics for the transitions between skills as shown in table 11 for the game against RoboDragons. During development, we also monitor for the presence of one node, and two node loops, on-line. Thus, we can quickly determine when skills transitions oscillate, or a skill fails to transition to another skill as appropriate.

Skill	**Cause**	**Transition**	**Count**	**Percent**
GotoBall	Command	Position	209	62.39%
	GotAShot	Kick	3	0.90%
	WithinDriveRange	DriveToGoal	67	20.00%
	CanSteal	StealBall	33	9.85%
	SpinOffWall	SpinAtBall	3	0.90%
	CanBump	BumpToGoal	20	5.97%
StealBall	Command	Position	1	3.03%
	BallAwayFromMe	GotoBall	14	42.42%
	BallAwayFromOpp	GotoBall	18	54.55%
DriveToGoal	CanKick	Kick	15	22.39%
	BallTooFarToSide	GotoBall	52	77.61%
BumpToGoal	Command	Position	1	5.00%
	TargetTooFar	GotoBall	19	95.00%
Kick	Command	Position	1	5.56%
	BallNotOnFront	GotoBall	17	94.44%
SpinAtBall	Command	Position	1	33.33%
	BallMoved	GotoBall	2	66.67%
Position	Command	GotoBall	212	100.00%

Table 11: Robot log from RoboDragons game

6.3 Remaining Issues

Based upon its performance in RoboCup competitions, the STP architecture provides many useful mechanisms for autonomously controlling a robot team in an adversarial environments. There are issues that require further investigation in order to improve its overall capabilities however.

The greatest weakness of our current approach resides in the need to develop the skills and its corresponding state machine. The techniques and algorithms described here provide very useful tools for developing robot behavior, however, development is still not a trivial process and much improvement can still be made. Each skill requires the development of a complex control algorithm, that is necessarily dependent upon the environment conditions and the capabilities of the robot hardware. Developing high performance skills is a challenging process that requires creativity, knowledge of the robots capabilities, and large amounts of testing. Combining these skills into state machines is equally challenging. To do so, one must accurately create the decision tree to determine under what conditions a skill transitions to its counterpart. One must avoid loops caused by oscillations, and ensure that each transition occurs only in states for which the target skill can operate from. Finally, each skills typically requires a large number of parameters to define its behavior and transition properties. Determining correct values for these parameters is a difficult and tedious process. Thus, our future work will focus on easing the difficulties skill development.

Another issue that needs further investigation is the dependence of skill execution on good sensor modeling. The unavoidable occurrence of occlusion, particularly during ball manipulation, has a severe impact on skill execution. Modeling the motion of the ball while it is occluded helps reduce this impact, but raises complications for when the ball modeling is incorrect. In particular, occasional observations of the ball may show inconsistencies with the modeled behavior, causing the skills to change their mode of execution. Consequently, oscillations in output decisions occur which detract from the performance of the skill. There is no easy solution to this problem, and it is an area of ongoing investigation.

7 Related Work

There have been a number of investigations into control architectures for robot teams. Prime examples include Alliance [30], three-layered based approaches [33] which build upon the single robot versions (e.g. [19]), or the more recent market based approaches [17]. None of these architectures, however, have been applied to *adversarial* environments. As discussed throughout this article, adversarial environments create many novel challenges for team control that do not occur in non-adversarial domains. Within the domain of robot soccer, there have, naturally, been many varied approaches into single robot and team control. We now review the most relevant of these approaches. We begin by focusing on teams that have demonstrated high-levels of team cooperation and performance.

Beginning at single robot control, there are a number of related approaches to our work. In particular, our skills based behavior architecture was loosely inspired by the techniques used by Rojas et. al. FU-Fighters team [31, 6]. Their team is controlled by successive layers of reactive behaviors that operate at different characteristic time constants. There is a clear difference between a FU-Fighters' style approach and STP. Plays, although selected reactively, enable a team to easily execute sequences of actions that extend over a period of time. Moreover, with dynamic role switching, the team members may change their role assignments but still carry out the directives of the play as a whole. The state-machine component of skills also contrasts against the purely reactive approach of FU-Fighters, whereby an extended sequence of actions can occur even in the presence of ball occlusion and noise.

The use of finite state machines for single robot control is not a unique approach. Indeed, many researchers have investigated state-machine approaches in a variety of contexts (e.g. [9, 5], or see [2] for more examples). Our approach is unique, however, in that each skill is a state in the state machine sequence. The state sequence is a function of both the world and the delegating tactic. Finally, the active tactic *continually* updates the parameters used by the active skill as it modifies its decisions based on the world. For example, the `TShoot` tactic may switch its decision from shooting at one side of the goal to shooting at the other. The active skill, whatever it may be, will make a corresponding switch, and perhaps transition to another skill depending upon the current situation. This combination of features makes the skill layer a unique approach.

At the team level, a number of teams use potential field based techniques for team control in the SSL (e.g., [22, 39, 28]). Potential field based team control is also popular outside of the SSL, in the mid-size [34], simulation [27] and Sony AIBO legged leagues [20]. Potential fields are used to determine target field positions for moving, or kicking. Essentially, the potential field value is determined for each cell in a grid covering the field. The shape of the potential field is formed by combining the usual attraction/repulsion operations common to potential field techniques [21, 1]. Some teams also add to the potential field functions based on clear paths to the ball. This approach is similar to the use of evaluations described in Section 4. The major difference occurs in the use of a sample-based approach to find a near-optimal point. We have found that a sample-based approach allows greater flexibility in defining the underlying objective function, additionally it avoids the issues of grid resolution and the computational effects of increasing the complexity of the evaluation function. Both techniques must use hysteresis or some similar mechanism.

Potential field techniques are also commonly used for navigation (e.g. [15, 18, 40]), although other reactive techniques are popular as well (e.g. [16, 8]). Reactive navigation is quite successful in a a dynamic and open environment, but they have been found by us and others to be less effective in cluttered environments like robot soccer (e.g. [38]). Here fast planning based approaches have been found to be much more powerful. Please see [13] for further discussion on this topic.

D'Andrea et. al.'s Cornell Big Red team [16] utilize a playbook approach that is similar to the use of plays described here. Their approach differs to ours, in that the playbook itself is a finite-state-machine where the plays are the states, rather than each play consisting of a set of states. As a result, the whole state machine is needed to have deliberative sequences of actions. The STP play-based approach, by encoding state transitions within plays, allows for multiple plays to be encoded to be operable for the same situations. As these multiple plays will utilize different sequences, it is reasonable to expect that the plays will have different effectiveness against different opponents. The STP approach, when combined with adaptation, allows for greater robustness of behavior against a range of opponents because the best play to use in a given situation can be found from amongst a range of applicable plays.

8 Conclusions

In this article, we have presented the STP architecture for autonomous robot team control in adversarial environments. The architecture consists of plays for team control, tactics for encapsulating single robot behavior, and a skill state machine for implementing robot behavior. The contributions of the STP architecture are to provide robust team coordination towards longer-term goals while remaining reactive to short-term opportunistic events. Secondly, the STP architecture is intended to provide team coordination that is responsive to the actions of the opponent team. Finally, the architecture is designed to be modular and to allow easy reconfiguration of team strategy and control parameters.

We have fully implemented the STP architecture in the small-size robot soccer domain, and have evaluated it against a range of opponents of differing capabilities and strategies. Moreover, we have evaluated our techniques and algorithms across a number of international and regional competitions. In this article, we have presented results based on these competitions that we believe validate the STP approach.

Much work remains, however, to further improve the capabilities of play-based team control and skill-based single robot behavior. In particular, considerable future work is required to overcome the need to specify large numbers of parameters in order to gain high-performance skill execution. Our future goals are in incorporate learning and adaptation at all levels in order to address this issue.

Acknowledgements

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.

References

[1] Ronald C. Arkin. Motor schema based navigation for a mobile robot. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 264–271, Raleigh, NC, April 1987.

[2] Ronald C. Arkin. *Behaviour-based Robotics*. MIT Press, 1998.

[3] M. Asada, O. Obst, D. Polani, B. Browning, A. Bonarini, M. Fujita, T. Christaller, T. Takahashi, S. Takokoro, E. Sklar, and G. A. Kaminka. An overview of RoboCup-2002 Fukuoka/Busan. *AI Magazine*, 24(2):21–40, Spring 2003.

[4] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, pages 322–331, Milwaukee, WI, 1995. IEEE Computer Society Press.

[5] T. Balch, G. Boone, T. Collins, H. Forbes, D. MacKenzie, and J. Santamaria. Io, Ganymede and Callisto - a multiagent robot trash-collecting team. *AI Magazine*, 16(2):39–53, 1995.

[6] Sven Behnke and Raul Rojas. A hierarchy of reactive behaviors handles complexity. In *Balancing reactivity and social deliberation in multi-agent systems*, pages 239–248. Springer, 2001.

[7] Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004. in press.

[8] Michael Bowling and Manuela Veloso. Motion control in dynamic multi-robot environments. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, pages 222–230. Springer Verlag, Berlin, 2000.

[9] Rodney A. Brooks. A robost layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[10] Brett Browning, Michael Bowling, James Bruce, Ravi Balasubramanian, and Manuela Veloso. Cm-dragons'01 - vision-based motion tracking and heterogeneous robots. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[11] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*, Japan, October 2000.

[12] James Bruce, Michael Bowling, Brett Browning, and Manuela Veloso. Multi-robot team response to a multi-robot opponent team. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003, to appear.

[13] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, Switzerland, October 2002, to appear.

[14] James Bruce and Manuela Veloso. Fast and accurate vision-based pattern detection and identification. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003, to appear.

[15] Bruno D. Damas, Pedro U. Lima, and Luis M. Custodio. A modified potential fields method for robot navigation applied to dribblign in robotic soccer. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[16] Raffaello D'Andrea, Tamas Kalmar-Nagy, Pritam Ganguly, and Michael Babish. The Cornell RoboCup team. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 41–51. Springer Verlag, Berlin, 2001.

[17] M Bernardine Dias and Anthony (Tony) Stentz. Opportunistic optimization for market-based multi-robot control. In *Proceedings of IROS-2002*, September 2002.

[18] Rosemary Emery, Tucker Balch, Rande Shern, Kevin Sikorski, and Ashley Stroupe. CMU Hammerheads team description. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 575–578. Springer Verlag, Berlin, 2001.

[19] E. Gat. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, 1997.

[20] Stefan J. Johansson and Alessandro Saffiotti. Using the electric field approach in the robocup domain. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[21] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), Spring 1986.

[22] Ng Beng Kiat, Quek Yee Ming, Tay Boon Hock, Yuen Suen Yee, and Simon Koh. LuckyStar II. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[23] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 1997. ACM Press.

[24] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. In *Technical Report No. 98-11*, October 1998.

[25] Scott Lenser, James Bruce, and Manuela Veloso. A modular hierarchical behavior-based architecture. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[26] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

[27] Jens Meyer, Robert Adolph, Daniel Stephan, Andreas Daniel, Matthias Seekamp, Volker Weinert, and Ubbo Visser. Decision-making and tactical behavior with potential fields. In R. Rojas G. A. Kaminka, P. U. Lima, editor, *RoboCup-2002: Robot Soccer World Cup VI*, pages 304–311. Springer Verlag, Berlin, 2003.

[28] Yasunori Nagasaka, Kazuhito Murakami, Tadashi Naruse, Tomoichi Takahashi, and Yasuo Mori. Potential field approach to short term action planning in RoboCup F180 league. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 41–51. Springer Verlag, Berlin, 2001.

[29] Norman S. Nise. *Control Systems Engineering: Analysis and Design*. Benjamin Cummings, 1991.

[30] L. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[31] Raul Rojas, Sven Behnke, Achim Liers, and Lars Knipping. FU-Fighters 2001 (Global Vision). In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[32] R. Simmons, J. Fernandez, R. Goodwin, S. Koenig, and J. O'Sullivan. Lessons learned from xavier. *IEEE Robotics and Automation Magazine*, 7(2):33–39, June 2000.

[33] R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R. Zlot. A layered architecture for coordination of mobile robots. In *Multi-Robot Systems: From Swarms to Intelligent Automata*. Kluwer, 2002.

[34] Steve Stancliff, Ravi Balasubramanian, Tucker Balch, Rosemary Emery, Kevin Sikorski, and Ashley Stroupe. CMU Hammerheads 2001 team description. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[35] Manuela Veloso, Michael Bowling, and Sorin Achim. CMUnited-99: Small-size robot team. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, pages 661–662. Springer Verlag, Berlin, 2000.

[36] Manuela Veloso, Michael Bowling, Sorin Achim, Kwun Han, and Peter Stone. CMUnited-98: A team of robotic soccer agents. In *Proceedings of IAAI-99*, 1999.

[37] Manuela Veloso, Peter Stone, and Kwun Han. The CMUnited-97 robotic soccer team: Perception and multiagent control. *Robotics and Autonomous Systems*, 29 (2-3):133–143, 1999.

[38] Thilo Weigel, Alexander Kliener, Florian Diesch, Markus Dietl, Jens-Steffen Gutmann, Bernhard Nebel, Patrick Stiegeler, and Boris Szerbakowski. CS Freiburg 2001. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[39] Gordon Wyeth, David Ball, David Cusack, and Adrian Ratnapala. UQ RoboRoos: Achieving power an dagility in a small size robot. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[40] Gordon Wyeth, Ashley Tews, and Brett Browning. UQ RoboRoos: Kicking on to 2000. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 555–558. Springer Verlag, Berlin, 2001.