Multi-Robot Team Response to a Multi-Robot Opponent Team *

James Bruce, Michael Bowling, Brett Browning, and Manuela Veloso
{jbruce,mhb,brettb,mmv}@cs.cmu.edu

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213, USA

Abstract

Adversarial multi-robot problems, where teams of robots compete with one another, require the development of approaches that span all levels of control and integrate algorithms ranging from low-level robot motion control, through to planning, opponent modeling, and multiagent learning. Small-size robot soccer, a league within the RoboCup initiative, is a prime example of this multi-robot team adversarial environment. In this paper, we describe some of the algorithms and approaches of our robot soccer team, CMDragons'02, developed for RoboCup 2002. Our team represents an integration of many components, that are in themselves state-of-the-art, into a framework designed for fast adaptation and response to the an changing environment.

1 Introduction

RoboCup small-size robot soccer is a game where two teams of five robots each play soccer on a 2.8m x 2.3m field with an orange golf ball [7]. Each team must be autonomous and must obey FIFA-like rules as dictated by a human referee. Small-size robot soccer is unique in that teams are allowed to use global vision, via overhead cameras, to augment any local sensors. Additionally, teams are allowed to use off-field computers to provide additional computational resources. Our approach to robot soccer focusses on global vision with an off-field PC, which communicates commands to the robots via a wireless radio link (see Figure 1).

Small-size games are highly dynamic where the ball can reach speeds of 2 to 3m/s and robots can reach speeds of over 1m/s. Controlling a team of robots in such a highly dynamic and competitive environment is a challenging research problem. Successful teams require both good team coordination and good individual robot skills. Indeed, it has been our experience that team performance as a whole is only as strong as the weakest component of the system on any control path. Thus, our development has placed equal emphasis on individual control skills and team cooperation. In this paper we focus on the issues critical to intelligent team response to an opponent. These issues cover individual robot skills of motion control and path planning for obstacle avoidance through to team coordination and strategy generation.

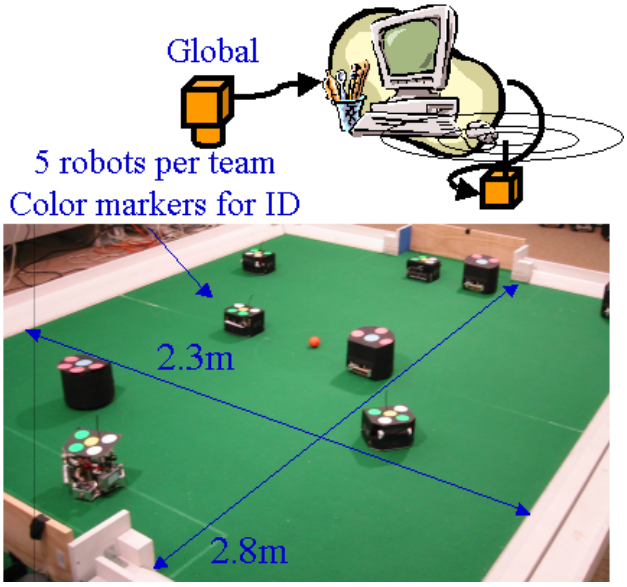

Figure 1: *An overview of the CMDragons small-size robot soccer team.*

We have developed a complete tactics-strategy framework for addressing these issues. In our framework, tactics are high-level individual robot skills. Examples include shooting, passing, blocking etc.

*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.

Strategies are formulated as *plays*, where a play is a sequence of tactics assigned to each robot that are appropriate for the current game state. Tactics, the primitives used by plays, control the robots through a way-point based motion control module. The motion control module utilizes fast planning for obstacle avoidance, and trapezoidal control rules for high-speed motion. In this paper, we describe each of these components and their role within the control hierarchy. We do not describe the robot hardware, or vision and tracking modules. Instead, we refer the reader to [4], [3] and [2] for more information.

2 Individual Robot Skills

Obstacle avoidance and motion control are closely related issues. In our framework, obstacle avoidance and motion control form the conduit through which high-level tactics influence the world. Regardless of how good the team coordination algorithms are, if the robots cannot move quickly to their target destinations while avoiding moving and potentially hostile obstacles, then the system performance as a whole will be poor. Solving the general control problem, for high-speed robots, is one of the key challenges in small-size robot soccer.

Our team hardware consists of two robot types, differential drive robots (diffbots) and omnidirectional robots (omnibots), making our team heterogeneous. Our approach to individual robot skills abstracts away the specifics of the robot drive configuration at the control levels. We have developed a hierarchical, way-point based, architecture capable of driving our robots reliably at speeds that peak at 2m/s. This layer abstracts away the specifics of the robot hardware by using standard target settings and accounting for the different acceleration/velocity bounds internally. Returning time estimates to achieve the desired target, rather than any robot specific information, allowing higher-level software to be independent of robot parameters.

Our control framework uses the notion of a target waypoint as the primitive for communicating within the hierarchy. Figure 2 shows the major control modules and associated parameters that form the hierarchy. Here, x, y, and θ are robot pose parameters in world-space with the origin at the center of the field and the x-axis pointing towards the opponents' goal. The velocity parameters labelled v_x, v_y are robot-centric speeds with v_x in the direction of the robot's kicker and $v_t max$ for the max speed the robot should reach at the target point. For the robot command, ω is the robot's rotational velocity while $\omega_1, \omega_2, \omega_3$ are the robot's motor speeds, respectively. While three motor speeds are shown, diffbots ignore the third speed parameter.

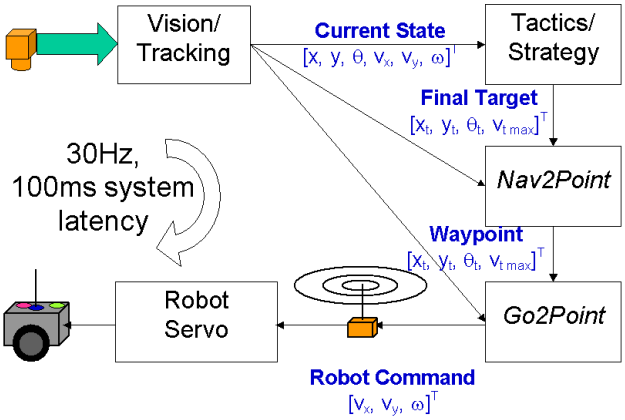

Figure 2: *The CMDragons control architecture is based on way-point primitives.*

Commands filter through the system usually in a waypoint format. Target points with direction and speed, that are generated by the active tactics are sent to the *Nav2Point* module. *Nav2Point* plans a path through the obstacle field to achieve this target. Section 2.2 describes the *Nav2Point* planning algorithm in more detail. The output of the *Nav2Point* module consists of an intermediate waypoint target for the *Go2Point* module. The following section describes the *Go2Point* algorithm and robot velocity servo loops that form the low-level control primitives.

2.1 Motion Control

The role of motion control is to drive the robot as quickly as possible to the target point with the desired target speed and orientation. Optimal control for a diffbot or omnibot, is known to be a difficult problem. To our knowledge, there are no known general solutions to the problem of finding optimal control trajectories to general target points for a diffbot or omnibot. For our team, motion control is part of a larger framework, meaning it cannot consume all the available computational resources. In fact, the dynamic nature of the task and environment means that the motion control algorithm must be run as quickly and as often as possible for each and every robot. For our team this means motion control must run at the frame rate of 30Hz for each robot on the field. Thus, we require efficient, near-optimal control.

The main limitations to robot motion are derived from the robot physics and the control path dynamics. All robots have limited acceleration and velocity capabilities. Acceleration bounds occur through limited motor torque and traction. Our robots are predominantly traction limited providing an upper bound to acceleration. If the robot exceeds the traction limit, the wheels slip causing highly non-linear

perturbations to the robot motion that are difficult to predict and control. For velocity, limited motor turning speeds provides an upper speed bound for linear and rotation motion. Thus we need both acceleration and velocity bounded control. The remaining motion limitations are derived from the control path. Vision is our primary sensor, meaning sensor updates are limited to the frame rate and have a latency which in our system is about 100ms (see [3] for more information). As with all sensors, there are the usual noise artifacts which must be addressed. Our probabilistic tracking module (described in [2]) compensates for most of these issues. More importantly, our tracking module provides prediction mechanisms to reduce the impact of latency.

We have a distributed control architecture by virtue of the separate sensing and actuator apparatuses, raising the question of how best to distribute control. In our approach, each robot operates local Proportional-Integral-Derivative (PID) velocity control loops to maintain velocities set from *Go2Point* via the wireless network. Future work is required to determine if this approach is indeed the best way to distribute the control between the off-field PC and robot base. *Go2Point* generates the velocity command using a trapezoidal control rule. Trapezoid control translates to constant accelerating at the traction limit, coasting at the velocity bound, and then braking at the traction limit to reach the target as quickly as possible.

Our approach resolves control into two independent 1D problems. We first build a Cartesian frame of reference and then treat the control of the x and y components as separate problems (see Figure 3). The origin of the reference frame is set to the robot's current location (accounting for latency) and is rotated to the target direction. Control for each axis is calculated using assuming trapezoidal control. Using time estimates for the completion of each axis, the accelerations are adjusted so that each axis completes as quickly as possible and at the same time. The total acceleration remains set to the empirically determined traction limit.

Generating the velocity command uses the calculated trapezoids to determine the target velocity for the next time period. For the omnibots this is just the velocity components for each trapezoid rotated into the robot-centric frame of reference. The diffbots are a little more complicated. When in motion the diffbots can accelerate laterally by turning while driving, but when stationary this method fails. In this case, the diffbots have a special case of first turning to the desired direction and then driving.

Although not optimal, our trapezoidal control strategy requires minimal computational resources and produces very efficient paths. Moreover, its simple structure allows us to easily compute reasonable time estimates for completion of tasks. The control algorithm takes no account of obstacles. Rather, obstacles are accounted for at the *Nav2Point* level described in the next section.

2.2 Navigation and Obstacle Avoidance

A dynamic multi-robot adversarial environment, such as robotic soccer, necessarily creates a challenging problem for robot navigation. Indeed a robot faces two major goals, namely (i) to reach its destination in the minimum possible time, and (ii) to avoid a field of moving obstacles, which may be adversarial.

The path-planning problem in general has been widely researched on, but it is not one that has found a universal solution. Specifically, in complicated, fast evolving environments such as RoboCup [7], currently popular approaches have their strengths, but still leave something to be desired. In particular, most require a state discretization and are best suited for domains with relaxed time constraints for planning. One of the relatively recently developed tools that may help tackle the problem of real-time path planning are Rapidly-exploring Random Trees (RRTs) [6]. RRTs employ randomization to explore large state spaces efficiently, and can form the basis for a probabilistically complete though non-optimal kinodynamic path planner. Their strengths are that they can efficiently find plans in high dimensional spaces because they avoid the state explosion that discretization faces. Furthermore, due to their incremental nature, they can maintain complicated kinematic constraints if necessary. We have been able to demonstrate the feasibility of an RRT-based algorithm for real robot navigation in a multi-robot domain [5].

Applying the basic RRT to real robots encounters several problems, in particular: (i) the paths found are near optimal, but not smooth as search proceeds stepwise; (ii) the algorithm is fast, but not fast enough for real robot navigation with frame-rate replanning; (iii) the search finds a complete path to the goal, but in real-time problems, there may not be enough time to expand the complete path to the goal; and (iv) the search expands a tree starting from scratch, and therefore efficient replanning is not considered. Our RRT-based algorithm, ERRT, addresses these different limitations by extending the basic RRT algorithm [5]. The contributions of our path planner are:

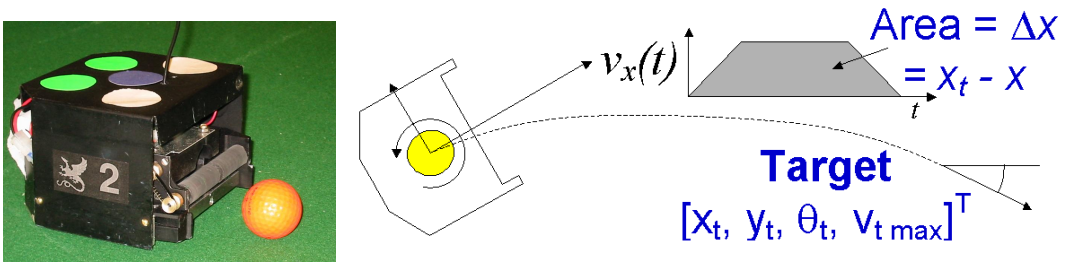

Figure 3: *The robot decomposes the problem into two 1D control problems.*

- Fast post-planning path smoothing
- KD-trees for efficient data storage and access
- Replanning

We focus on explaining the replanning component, which is crucial for the real-time navigation with difficult obstacle avoidance specifically due to the other moving robots. (Further details are available at [5].) ERRT allows for the input of a "bias" to its search in the form of *cached waypoints* that act as intermediate target points in addition to the goal point. The cached waypoints in principle can come from any source. But the interesting issue is that they can come from previous incomplete searches, opening therefore a principled approach to replanning. ERRT proceeds as follows:

- Initially, when no cached waypoints are provided, ERRT does a basic RRT-based search in real-time and with time constraints. The outcome of the search is a path towards the goal. The path may be complete or incomplete. The world changes continuously creating a challenging problem for navigation with the robots as dynamic moving obstacles. Whether the path is complete or incomplete to the goal, the robot still needs to find a path at every cycle. The outcome of ERRT is transformed into a set of cached waypoints remembering the path that was found in the immediately previous search iteration. These waypoints are given as input to the path planning search at the next time step.
- When ERRT receives cached waypoints in its input, it uses them as a bias. ERRT then expands its search tree in three possible directions. It chooses a target state with three probabilities, namely:
  - The goal with probability p_g,
  - A random waypoint with probability p_r,
  - A random point with probability $1-p_g-p_r$.

Figure 4 illustrates these three expansions of the ERRT algorithm.

With the ERRT approach, we achieve excellent coordinated multi-robot team navigation. The waypoint cache provides much improved performance on difficult but possible path planning problems. In our small-size team, all the robots successfully navigate to points using ERRT. The selection of the points to which navigation proceeds is performed by the high-level behavior architecture which we now describe.

3 Multi-Robot Coordination

For multi-robot teams in adversarial settings, good individual skills must be employed within the team's high-level strategy to be effective. This strategy should emphasize the team's strengths while exploiting the opponent's weaknesses. There are three important goals that our team behavior architecture, or strategy, achieves:

- Coordinated team behavior,
- Ease of human and automated augmentation of the team's strategy, and
- On-line adaptation to the opponent.

The goal of on-line adaptation, specifically, is a very challenging problem. In adversarial environments, we want our multi-robot team to adapt its team play to different opponents. Adaptation during the short time that comprises a game is clearly not trivial. We have pursued research on theoretically justified multiagent learning algorithms that allow for players to learn to respond to opponents [1]. In simulated robotic soccer, we have also used a coach agent to position team members in response to the opponent team [8]. The problem is more complicated with a team of real robots.

All of these goals, including opponent adaptation, are achieved through the combination of our *playbook* team behavior representation and execution modules, and through the explicit gathering and exploita-

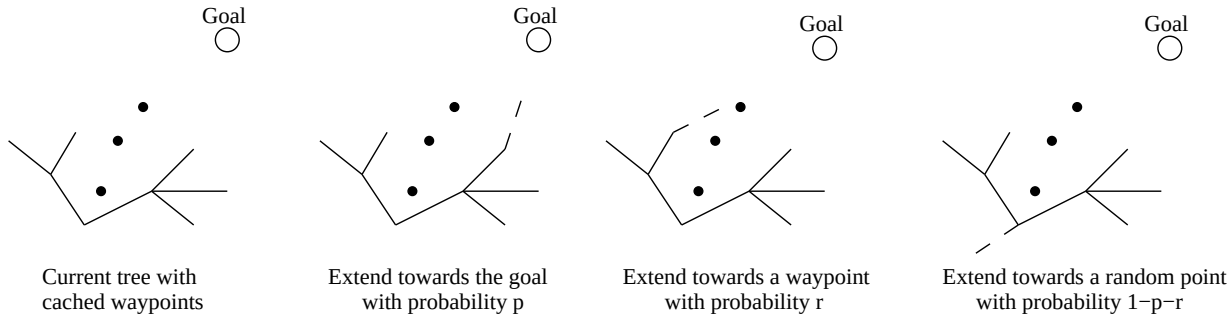

Figure 4: *Extended RRT with a waypoint cache for efficient replanning.*

tion of information about the opponent during the game.

3.1 Playbook: Team Behavior Representation

Possible coordinated actions of the team are stored as *plays* in the playbook. The plays are self-contained, abstract, coordinated plans for an entire team of agents. Plays may be situation specific, or very general. They may involve just one simple action for each team member, or may be a complex sequence of coordinated behavior. An example of a simple play, as parsed by our system, is shown in Table 1(a). Notice that the use of a human-readable syntax greatly contributes to the ability for the team designers to understand, modify, and add new plays into the team's strategy.

Plays have three main components: applicability ("APPLICABLE") conditions, termination ("DONE") conditions, and roles ("ROLE"). The applicability conditions are similar to operator preconditions in classical planning. They use logical combinations of high-level predicates to constrain the possible situations when this play may be executed. In the naive play in Table 1(a) the play is applicable anytime our team is on offense, which is a high-level predicate capturing details such as ball possession and field position.

Unlike classical planning operators, plays have no deterministic effects due to the unpredictability of the noisy environment and unknown opponent. Termination conditions, though, provide the ability to specify when the play should be stopped. In addition, the termination conditions specify whether the play, upon termination, should be considered a success or a failure. In the naive play, the only termination condition is that the team is no longer on offense, which as specified is considered a play failure. Success and failure of plays is a very powerful piece of information, and its use in adaptation during the game is described later.

PLAY Naive Offense

```
APPLICABLE offense
DONE failure !offense
FIXEDROLES 1 3 4 2

ROLE 1  shoot true
ROLE 2  block 300 700 1
ROLE 3  defend_lane ...
ROLE 4  block 900 1400 -1
```

PLAY Corner Out and Cross

```
APPLICABLE offense in-their-corner
DONE success !offense their-side
!in-their-corner
DONE failure !offense
FIXEDROLES 1 3 4 2

ROLE 1  pass 2
        block 900 1400 -1
ROLE 2  position_for_pass { c { B 1100 400 } 300 }
        receive_pass
        pass 3
ROLE 3  position_for_pass { c { B 1100 -400 } 300 }
        none
        none
        receive_pass
        shoot
ROLE 4  block 300 700 1
```

Table 1: *Two example plays. The top is a simple play, "Naive Offense", while the bottom shows a more complex play involving two passes. The "..." correspond to omitted parameters that further refine the specific behavior.*

Ball Tactics
shoot
pass
receive_pass
receive_deflection
clear
steal
active_def

Other Tactics
defend_point
defend_line
defend_lane
block
mark
screen
position_for_pass
position_for_loose_ball
position_for_deflection
position_for_rebound
dribble_to_pass
dribble_to_shoot

Table 2: *List of available tactics.*

The final and main component of the play are the roles. A role is a sequence of behaviors to be carried out by a member of the team. Each line in a role's description defines a *tactic* or individual behavior that requires no team coordination. A list of tactics is shown in Table 2. Tactics are often highly parameterized for flexibility, e.g., the "`block`" tactic listed in Table 1(a), and so the possible individual behaviors that can be specified is very rich. In addition, multiple tactics can sequenced together to make a temporally extended coordination of team behavior, e.g., a play to pass to another robot to shoot. In this case, the role's are synchronized so that all the roles are on the same index of their tactic list. If no tactic is specified for a role at the index then the role continues to execute its previous tactic. This synchronization and sequencing is handled by the execution module. The ordering of the roles is also important as it corresponds to their relative importance to the play. The first role is the most critical, while the last role is the least important. The execution module makes use of this information to simplify the problem of assigning robots to the play's roles.

The other information provided in the play, the "`FIXEDROLES`" keyword, is used when certain aspects of the execution module are disabled. In normal circumstances it is ignored. Another example of a play, making use of specialized predicates and sequences of tactics, is shown in Table 1(b).

3.2 Playbook: Team Behavior Execution

Plays are a powerful and general representation of team behavior, but require a robust and specialized execution module to contribute to the goals of easy augmentation and opponent adaptation. The execution module is composed of two parts: a play executor and a play selector.

Play Executor. The play executor is responsible for actually instantiating the play into real robot behavior. This instantiation consists of many key decisions: role assignment, role switching, sequencing tactics, and opportunistic behavior. Role assignment uses tactic-specific methods for selecting a robot to fill each role. This is performed in 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 which change the effectiveness of robot's fulfilling roles. The play executor handles this 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. This is performed by monitoring the current *active player*, the robot whose role specifies a tactic related to the ball (see Table 2.) When this tactic succeeds the play is transitioned to the next 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 this can be quickly enough and with a high likelihood of success the robot immediately switches its behavior to take advantage of the situation.

These aspects of the play executor makes plays more general by providing basic behavior beyond what the play specifies. This also gives our 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.

Play Selector. The play selector is responsible for selecting the play out of the playbook that will be executed. The selection system first finds the set of plays whose applicability condition evaluates to true. If there is only a single play, this play is selected for execution. If there is more than one play available the selector chooses stochastically based on the weights assigned to the play as specified in the playbook. The second task of the selector is to ad-

just the weights of the plays during the game. This is adjustment is performed using the success and failure conditions of the play. A successful play has its weight increased, while a failing play has its weight decreased. During the game plays that do not work well against the current opponent will be selected less and less frequently, while successful plays will be executed more often. This provides a powerful layer of opponent adaptation at the highest level of strategy.

3.3 Gathering information towards responding to the opponent

We have previously used the position of the opponents at each particular time step to position our own team in response to the opponents' positions [9]. Our current goal is to respond to patterns of opponent positioning over time rather than their positions at single time steps. Virtually all teams within the small-size league currently use fixed strategies. Most teams have fixed role assignments within these strategies. We wish to exploit this static structure to make better strategic decisions. Our approach to this is to collect a variety of histograms that describe the opponent's capabilities, in terms of speed and acceleration, and elements of the opponent's strategy. The latter includes occupancy (i.e., what positions the opponents occupy on the field) and lanes of movement (i.e., how the robots move around the field).

As an example of how clear this strategic information can be, we collected histograms of our team performing a common strategy. The strategy and corresponding role assignment was fixed with two attackers, two circle defenders and one goalie. The circle defenders defend the goal by moving on a circle centered on the goal with predefined radius. Figure 5 shows the raw position information recorded from the tracking module and an occupancy histogram where lighter blocks indicate higher occupancy. The histograms were recorded over approximately 5 minutes with the ball being moved manually around the field to simulate a real game.

In our on-going work, we are analyzing which way the robots can use the information captured by the histograms. We have identified that this information can be used to:

- Determine if the opponent uses static positions. Sending our robots to these locations disrupts the opponent.
- Determine what trajectories the opponent uses. Blocking these trajectories affects opponent motion.
- Determine where the opponent attacks from. Blocking these positions improves our defense.

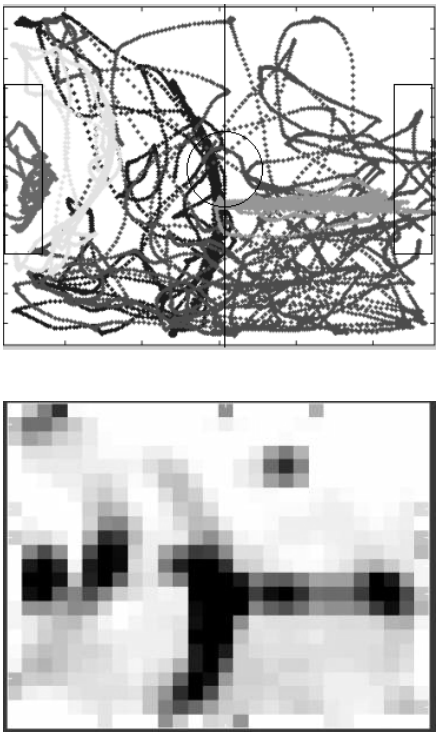

Figure 5: *Position plots and occupancy histograms of a fixed strategy, respectively. Darker parts in the histogram indicate higher occupancy. The strategy used a goalkeeper, two circle defenders each on a different radius from the goal, an attacker and a support.*

- Estimate the physical capabilities of the opponents. Knowing the opponent dynamics improves our prediction.
- Estimate general strategy properties e.g., number of robots sent to the ball. Modeling the opponent allows us to respond at the strategy level.

4 Conclusion

In this paper, we report on our most recent contributions in our developments of our RoboCup small-size team. We recognize that multi-robot teams in adversarial environments face several challenging goals, including the team needs to be composed of individually skilled robots, the robots need to have robust team behaviors, and the team needs to adapt to the opponent robots. In this paper, we describe our approach to motion control and navigation that can effectively handle a field with fast moving obstacles. We then present our new playbook team behavior architecture. The language for the representation of tactics and plays provides a solid basis for the explicit incorporation of human given or learned strategies to respond to specific opponents.

References

[1] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[2] Brett Browning, Michael Bowling, and Manuela Veloso. Improbability filtering for rejecting false positives. In *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002.

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

[4] James Bruce, Michael Bowling, Brett Browning, and Manuela Veloso. CMDragons'02: Manual and technical team description. Technical Report forthcoming, School of Computer Science, 2002.

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

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

[7] 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.

[8] Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, Best Paper Award*, Toulouse, France, April 2002.

[9] Manuela Veloso, Peter Stone, and Michael Bowling. Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, Boston, September 1999.