CMDragons'03 Team Description Paper

Brett Browning, James Bruce, Michael Bowling,
Michael V Sokolsky, and Manuela Veloso

School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213, USA
{brettb, jbruce, mhb, mmv}@cs.cmu.edu,
WWW home page: http://www.cs.cmu.edu/~robosoccer/small

1 Introduction

Like all robot soccer teams, CMDragons'03 (shown in Figure 1) is an integration of a number of complex components into a seamless whole. In this team description, we describe the key components for our system as used at RoboCup 2003. Specifically, we address how the system as a whole fits together and what issues must be considered for such integration. Readers should note that much of the work described here is available as an implementation in our official source code release, which can be found at *http://www.cs.cmu.edu/~coral*. Throughout this paper, we assume that the reader is familiar with the general structure of small-size robot soccer applications (for further details, please see other articles in this book). Further details on our system can be found in [3, 4].

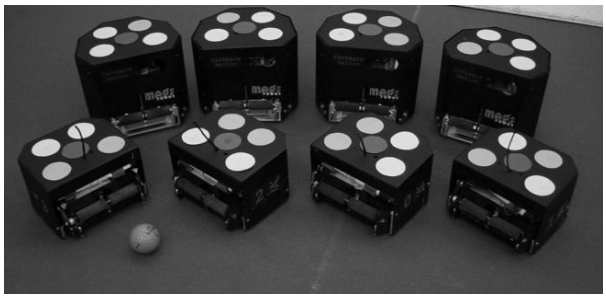

Fig. 1. The full CMDragons team.

Figure 2 shows the major components of our system. The system consists of three main, hierarchical control loops based on the two main sources of sensory information. The control loops are distributed across the central PC, or multiple PCs if desired, and the actual robots. Each robot contains a DSP processor that executes the inner control loop. This control loop maintains the robot velocities at the commanded values using the fast odometry feedback from the motors. The tactic control loop executes on the central PC and generates commands that are sent to the robot. An instantiation of the tactic control loop operates

independently for each robot, and is executed once per received vision frame at 30Hz. Furthermore, each robot may be executing a different tactic depending upon the task and current team strategy at hand. In parallel to the tactics runs the strategy control loop. This loop executes once per received vision frame, although it only performs heavy computation when required. The strategy layer instantiates new tactics, with appropriate parameterization for each robot on the field. Through this mechanism, team coordination and synchronization is enforced. Both the strategy and tactics control loops use the output of the vision system as the only sensor.

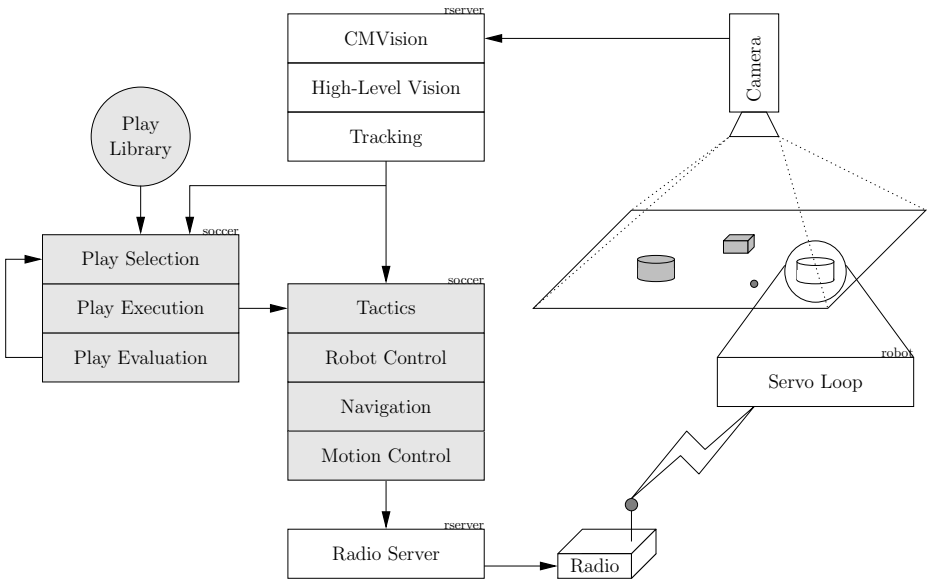

Fig. 2. Overview of the CMDragons'03 team architecture.

2 Vision

The CMDragons vision processing heavily utilizes color-based recognition techniques. We only briefly describe our vision system here and refer the interested reader to [5]. The CMDragons'03 vision system is based on the CMVision color segmentation library. Using CMVision, and appropriate camera calibration, the pixels are segmented based on perceived color and then grouped into connected regions with some noise removal. The resulting color 'blobs' are then matched against the known robots on the field. High confidence matches are then reported to the tracking system.

We use a probabilistic tracking system based on Extended Kalman-Bucy filters to obtain filtered estimates of ball and robot positions. Additionally, the

filters provide velocity estimates, appropriate co-variances, and robust mechanisms for predicting the short-term future position of a robot. The latter is critical to enable our system to overcome the debilitating effects of latency when robots move at speed. With our current hardware, system latency is around 100ms. While robots can reach speeds approaching 2m/s. Over such a small and cluttered field, the system latency can significantly degrade the obstacle avoidance capabilities of the team if not compensated for properly.

In our framework, vision and the radio daemon are run as a single server with UDP connections to client programs. Thus, vision information is provided at frame-rate with a low-latency connection to any client programs running on the network, while robot velocity commands are received from clients running on the same network. This arrangement is required to allow the tracking system to make use of the commands sent to the robots.

3 Single Robot Behavior

In the CMDragons system, single robot behavior is controlled by a hierarchy of tactics, skills, navigation, motion control, and finally the low-level control routines running on the robot itself.

3.1 Tactics

Tactics are the highest level of single robot behavior. Each tactic, which is heavily parameterized, encodes a single robot behavior and provides the action primitives for team strategy. Tables 1 and 2 show the list of tactics used at RoboCup 2003. Each tactic executes the high-level calculations required to decide the parameter values passed to the skills. For example, when shooting, the `shoot` tactic determines where the robot should aim as well as evaluating where the robot should move to in order to get the best shot on goal. In all cases, the tactics base their evaluations upon the estimated world state which is encapsulated within the `World Model`. In this context, the world model manages all the predicted and estimated information about the world including:

- Robot state – position, orientation and velocities,
- Ball state – position, velocity, and tracking covariances
- Game state and referee commands,
- High-level predicates derived from the observed world, state. Examples include ball possession, particular opponent roles, opponent with the best shot,
- Models of opponent capabilities including observed speed/acceleration capabilities and shooting positions.

All predictions are through the known measured latency period, which is measured to be approximately $100ms$ with our current hardware. The only exception to this rule are ball predictions by the goal keeper. In this case, the goal keeper predicts ahead to determine where the ball will cross the goal line, if indeed it will. Our goal keeping algorithm at RoboCup 2003 made use of the

predicted covariances to mix between intercepting the moving ball and choosing the ideal 'static' defense position based on the ball's current location. The latter is determined by equalizing the available open angles on either side of the goalie from the location of the ball predicted through the latency period.

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

Table 1. List of active tactics with their accepted parameters.

3.2 Skills

Tactics instantiate behavior through skills. Each skill is a focused control policy for performing a short-term complex action such as manipulating the ball off the wall, kicking the ball, or stealing the ball off an opponent. Each skill can generate robot action by sending direct raw velocity commands, by sending target way-points to the motion control module, or by sending target way-points to the RRT-based navigation planner. The mode of action generation used depends upon the requirements of the skill.

As each skill performs a focused control policy, it is often only useful for specific world states. Thus, to meet the objectives of any given tactic requires executing a sequence of skills. For example, to shoot the ball on goal, the robot must move into position near the ball, get the ball on its dribbler, move to where it can get a shot, aim, and then kick the ball. Depending upon whether the ball is against the wall, or in the possession of an opponent, the details of the sequence, or even the sequence itself may need to change. Thus, skills are collected into state-machines, where each skill is effectively a state in the state-machine. A skill may belong to multiple state machines and state transitions are dependent upon the executing tactic as well as the state of the world. Each skill performs its actions based on the observed state of the world, as well as the parameters

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 non-active tactics with their accepted parameters.

passed to it each frame by the active tactic. Figure 3 shows a log sequence of a single robot performing a steal, followed by a kick, in order to shoot a goal. The robot executes the `shoot` tactic the entire time, but progresses through a sequence of skills based upon the situation at each moment.

Developing and debugging skills is a complex, time-intensive process. Moreover, given the speed of the game, it is often difficult to observe the executing of a skill in practice. Thus, we make use of two types of logging facilities to record skill execution. The first is a raw game log, which allows us to play back what happened in simulation or on the real robots and step through what the skills and tactics were 'thinking' while executing. A second type of logging is to record the different transitions in the skill layer and their cause. Figure 3 shows such a log recorded at RoboCup 2003. For each skill, the recorded transition count and frequency is shown with a label describing the basic cause for the transition. Using such a log enables us to quickly identify how often a skill is used, and how effective a particular transition is. Armed with such knowledge, it becomes significantly easier to adjust the transitions between skills and to focus effort on the skills that need the most improvement.

3.3 Navigation and Motion Control

The robot control layer uses the navigation and motion control layers to cause the robot to move through the world. The navigation layer utilizes a Rapidly exploring Randomized Tree (RRT) based algorithm to quickly generate near-optimal, obstacle free paths for each robot at each frame [4]. Given the need to dedicate computational resources to other parts of the system, path planning must be extremely efficient to be of use. In our current implementation on a 1GHz Athlon PC, path planning takes approximately 2ms on average for each robot. Finally, actual motion commands are generated each frame by the motion control layer. Motion control takes a short-term, obstacle-free, target way-point

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 3. Robot log from RoboDragons game

generated by the navigation system and calculates the commanded velocities required for the robot to reach that target point in near-minimal time while respecting the robot's limitations. We use a trapezoidal control scheme to achieve this goal, which is described in more detail in [3].

4 Team Behavior through Fixed Team Plans

The final component of the system, and in some respects the most interesting, is the strategy layer. Team strategy for the CMDragons is controlled via a *playbook* based approach. Essentially, team strategy is encoded as a collection of fixed team plans, or *plays*, to borrow sporting terminology. Each play contains a set of roles, which are assigned to robots during execution dynamically using evaluation functions for the tactics that the role uses. Each role, in turn, contains a fixed sequence of actions, or in this case tactics with appropriate parameters. Thus, a team plan is enacted by assigning robots to roles, and then executing the appropriate tactics for each role. Virtually any sequence of team behavior can therefore be generated in this fashion. The most interesting aspect of the playbook approach is that multiple plays can be developed for the same situation, where selection of which play to execute can be adapted over time based on team performance. If done correctly, this adaptation of play selection can allow a team to adapt its strategy to perform best against an opponent team.

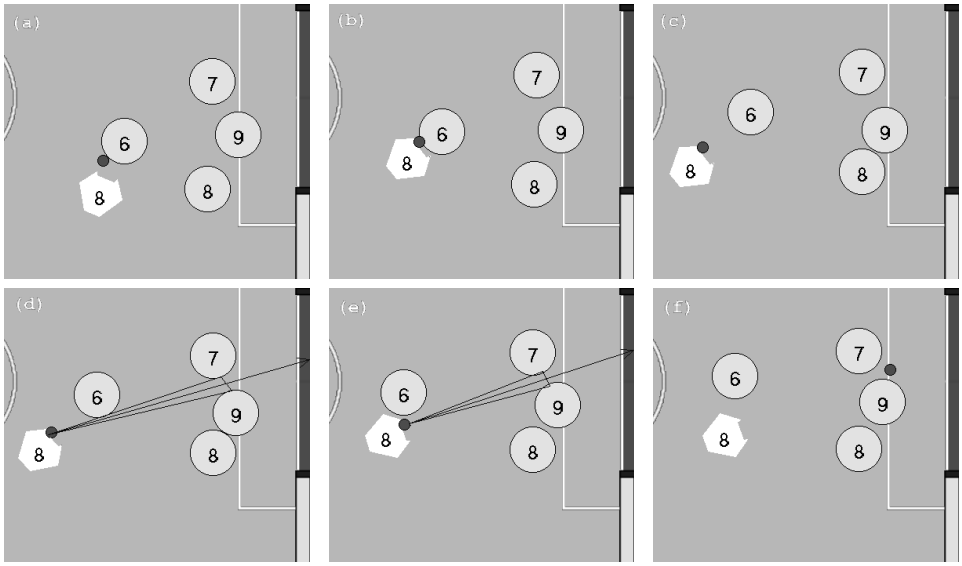

Fig. 3. An example shoot sequence taken from the RoboCup 2003 round robin game of CMDragons'03 vs RoboDragons. The robot first executes the `steal_ball` skill (image (a) and (b)), followed by `goto_ball` (image (c)). Once the ball is safely on the robot's dribbler, it begins `drive_to_goal`, image (d), 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.

4.1 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.

4.2 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.

We now describe each of these components in greater detail.

Applicability Conditions define when a play can execute as a function of the world state. Applicability is specified as a logical DNF, prefixed with the `APPLICABLE` keyword, with each disjunct specified separately. The DNF is formed using high level boolean predicates derived from the estimated world state. In the example play in Table 4, the play can only be executed from a state where the `offense` predicate is true. The `offense` predicate is actually a complex combination of the present and past possession of the ball and its present and past position on the field.

Termination Conditions specify when the play's execution should stop using a logical DNF formed from the high-level predicates, prefixed with the `DONE` keyword. 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. In the example play in Table 4, 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 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.

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 4, 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, which produces a `completed` result. The second occurs when a play runs for too long a time without completing or being successful. 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.

Roles are the main component of the team plan. Each play has four roles, one for each non-goalie robot on the field. A role consists of a list of tactics and parameters, for the robot to perform in sequence. In the example play in Table 4, roles 1 and 3 have two sequenced behaviors. In this case, the robot assigned role 1 will get the ball and pass it to role 3 to take a shot on goal, and then attempt to mark the opponent that is 'closest to the ball'. 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. Considering our example again (Table 4), 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.

Execution consists of recalculating the role assignment, monitoring the termination conditions and execution of the active tactic. If the active tactic completes, each role progresses to the next tactic in its respective sequence. If a play terminates, the result is recorded and used for play adaptation. A new play is then selected from the set of applicable plays and the process repeats.

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 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.

4.3 Playbook and Play Selection

Plays, by themselves, allow one to quickly build team behavior, however, most of the advantage of plays resides in the ability to have multiple play options for each possible world state and to adapt selection of plays as a function of team

```
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 4. A complex play involving sequencing of behaviors.

performance. To achieve this, a playbook associates a weight with each play. This weight corresponds to how often the play should be selected when applicable.

Play selection then amounts to finding the set of applicable plays and selecting one based on the weights. Specifically, if $p_{1...k}$ 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 [6] and Exp3 [1]) 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 [2].

4.4 Performance

In terms of effectiveness, the play architecture proves capable at multiple levels. It is flexible, allowing the entire team strategy to be changed relatively quickly. Indeed, between the round-robin and the elimination round-robot at RoboCup 2003, we completely rewrote our playbook. The resulting playbook consisted of around 16 plays which were used for the remainder of the competition with only minor changes. Although we only offer anecdotal results, play adaptation proved effective where the playbook would focus onto a small collection of dominant plays against the given opponent. Our future work will examine the logs recorded at RoboCup to determine exactly how effective play adaptation was. Figure 4 shows a play sequence captured from the game against ToinAlbatross. Here, the robot in attack is unable to shoot on goal. Instead, it attempts to pass to its teammate, which is executing the `position_for_deflection` tactic. The remaining robots are executing defensive tactics. With no shot on goal, the attacker passes to its teammate, which then successfully shoots the ball directly on goal leading to the maximum reward.

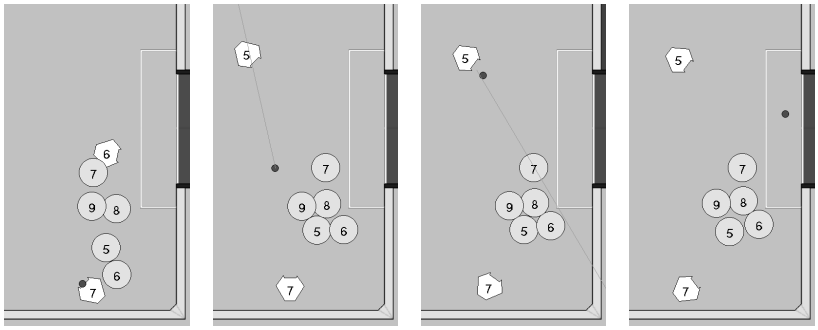

Fig. 4. A play sequence captured from the game against ToinAlbatross from Japan. Our robots are shown as triangular polygons, their robots as circles. We are shooting to the right. The line shows the tracked ball velocity. The image sequence shows the active player unable to shoot on goal (a), and instead passing to a teammate (b). The teammate is positioned for a deflection on goal, which occurs in (c) and (d).

5 Conclusions and Future Directions

CM-Dragons'03 represents an extension of our previous research efforts on teams of heterogeneous robots. In this short paper, we have surveyed our system structure as a whole. We refer the interested reader to our other works for

further details on the individual components that make up our system. Finally, much of our software is publicly available on our web pages and we encourage the interested reader to peruse our work as it was meant to be (see *http://www.cs.cmu.edu/~coral* for more details).

Acknowledgements

The authors would like to thank Paul Harada, Jennifer Lin, Ling Xu, Betsy Ricker, David Rozner, and Dinesh Govindaraju, for their all their hard work and effort preparing the team for RoboCup 2003 and the RoboCup American Open 03. Without all their effort, the CMDragons'03 team would not have been possible.

References

1. 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.
2. Michael Bowling, Brett Browning, Allen Chang, and Manuela Veloso. Plays as team plans for cooperation and adaptation. In *IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modelling, Planning, Learning, and Communicating*, Acapulco, Mexico, August 2003.
3. 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.
4. James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, Switzerland, October 2002. An earlier version of this paper appears in the Proceedings of the RoboCup-2002 Symposium.
5. 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.
6. N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.