CMDragons 2010 Team Description

Stefan Zickler, Joydeep Biswas, Kevin Luo, and Manuela Veloso

Carnegie Mellon University
{szickler,veloso}@cs.cmu.edu
{joydeep,kwluo}@andrew.cmu.edu

Abstract. In this paper we present an overview of CMDragons 2010, Carnegie Mellon's entry for the RoboCup Small Size League. Our team builds upon the research and success of RoboCup entries in previous years. The overview describes both the robot hardware and the general software architecture of our team. Technical improvements include a new attacker control system, a short-term physics-based motion planner, and an experimental system for inferring opponent state from ball dynamics observations.

1 Introduction

Our RoboCup Small Size League entry, CMDragons 2010, builds upon the ongoing research used to create the previous CMDragons teams (1997-2003,2006-2009) and CMRoboDragons joint team (2004, 2005). Our team entry consists of five omni-directional robots that are wirelessly controlled by an offboard computer. Sensing is provided by two overhead mounted cameras via the SSL-Vision system. This paper describes the robot hardware and the offboard control software required to implement a robot soccer team.

2 System Overview

Our team consists of five homogeneous robot agents. In Figure 1, an example robot is shown with and without a protective plastic cover. The hardware is mostly the same as used in RoboCup 2006-2009. We believe that our hardware is still highly competitive and allows our team to perform close to optimal within the tolerances of the rules. One noticable hardware improvement for 2010 however, is a new dribbler-mount assembly, better protecting the robot's infrared sensors and dribbler motor. Besides this hardware improvement, we focus most of our efforts on improving the software to fully utilize the robots' capabilities instead.

2.1 Robot Hardware

Each robot is omni-directional, with four custom-built wheels driven by 30 watt brushless motors, each featuring a reflective quadrature encoder. The kicker is

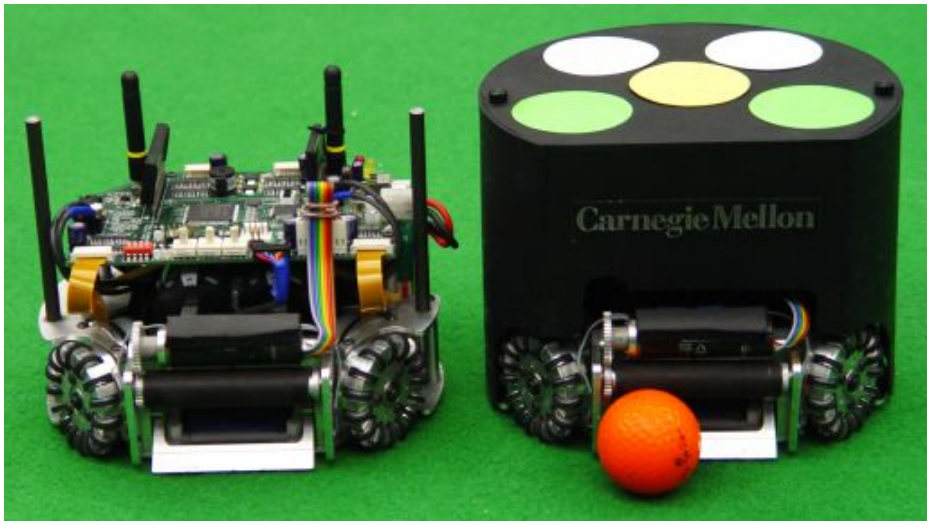

Fig. 1. A CMDragons robot shown with and without protective cover.

a large diameter custom wound solenoid attached directly to a kicking plate. It is capable of propelling the ball at speeds up to $15m/s$, and is fully variable so that controlled passes can also be carried out. The CMDragons robot also has a chip-kicking device, implemented by a custom-made flat solenoid located under the main kicker, which strikes an angled wedge visible at the front bottom of the robot. It is capable of propelling the ball up to $4.5m$ before it hits the ground. Both kickers are driven by a bank of three capacitors charged to $200V$. Ball catching and handling is performed by a motorized rubber-coated dribbling bar which is mounted on an hinged damper for improved pass reception. A more detailled description of the robot's design and electronics can be found in [1].

Our robot is designed for full rules compliance at all times. The robot fits within the maximum dimensions specified in the official rules, with a maximum diameter of 178mm and a height of 143mm. The dribbler holds up to 19% of the ball when receiving a pass, and somewhat less when the ball is at rest or during normal dribbling. The chip kicking device has a very short travel distance, and at no point in its travel can it overlap more than 20% of the ball due to the location of the dribbling bar. While technically able to perform kicks of up to $15m/s$, the main kicker has been hard-coded to never exceed kick-speeds of $10m/s$ for full rule compliance.

2.2 Software

The software architecture for our offboard control system is shown in Figure 2. It follows the same overall structure as has been used in the previous year, outlined in [2, 1]. The major organizational components of the system are a server program which performs vision and manages communication with the robots, and two

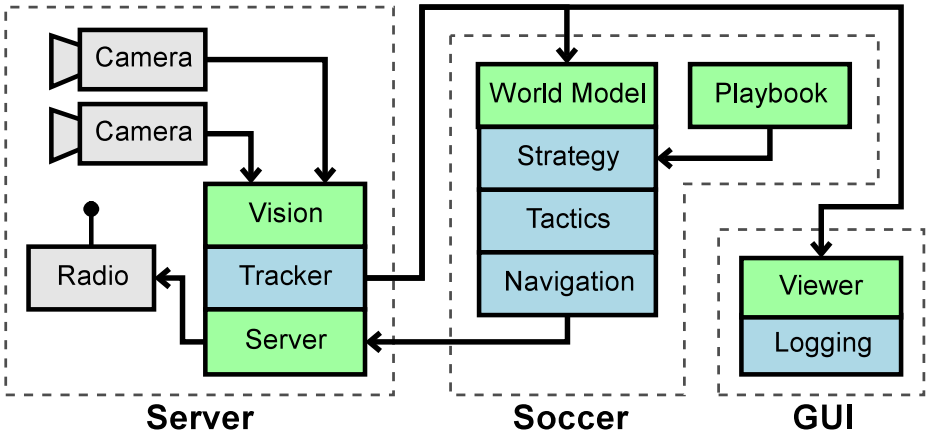

Fig. 2. The general architecture of the CMDragons offboard control software.

client programs which connect to the server via UDP sockets. The first client is a soccer program, which implements the soccer playing strategy and robot navigation and control, and the second client is a graphical interface program for monitoring and controlling the system.

The server program consists of vision input, tracker, radio, and a multi-client server. The vision input is supplied via ethernet from the RoboCup SSL shared vision system *SSL-Vision* [3]. Some of the integration details are described in section 3 of this paper. Tracking is achieved using a probabilistic method based on Extended Kalman-Bucy filters to obtain filtered estimates of ball and robot positions. Additionally, the filters provide velocity estimates for all tracked objects. Further details on tracking are provided in [4]. Final commands are communicated by the server program using a RS232 radio link.

The soccer program is based on the STP framework [4]. A world model interprets the incoming tracking state to extract useful high level features (such as ball possession information), and act as a running database of the last several seconds of overall state history. This allows the remainder of the soccer system to access current state, and query recent past state as well as predictions of future state through the Kalman filter. The highest level of our soccer behavior system is a strategy layer that selects among a set of plays [5, 6]. Below this we use a tree of tactics to implement the various roles (attacker, goalie, defender), which in turn build on sub-tactics known as skills [4]. One primitive skill used by almost all behaviors is the navigation module, which uses the RRT-based ERRT randomized path planner [7–9] combined with a dynamics-aware safety method to ensure safe navigation when desired [10]. It is an extension of the Dynamic Window method [11, 12]. The robot motion control uses trapezoidal velocity profiles (bang-bang acceleration) as described in [13, 4]. Additionally, our system features a detailed physics-based simulator based on rigid-body dynamics as described in [2]. Two improvements to our software this year are an improved

control system for the *Attacker* robot behavior (see section 4), as well as a short-term physics-based motion planner (see section 5) for improved ball dribbling.

3 Vision Hardware and Software

CMDragons 2010 operates using SSL-Vision as its vision system [3]. In our lab, we use two Firewire 800 cameras (AVT Stingray F-46C) which provide a 780×580 progressive video stream at 60Hz. SSL-Vision is released as open source and is therefore available to all teams. In order to use SSL-Vision, the "Vision" component in Figure 2 represents a network client that receives packets from the SSL-Vision system. These packets will contain the locations and orientations of all the robots, as well as the location of the ball. However, data fusion of the two cameras and motion tracking will continue to be performed within our system, as SSL-Vision does not currently support such functionality.

For competing in RoboCup 2010, SSL-Vision provides several advantages compared to CMDragons pre-2009 system. One major improvement is the geometry calibration of the cameras. Our previous system required the use of paper calibration patterns to be carefully placed on the field for calibration purposes. SSL-Vision does not require any such calibration patterns and can be fully calibrated through its user interface. Another improvement is that SSL-Vision provides direct access to all DCAM parameters of our Firewire cameras, thus allowing configuration of settings such as exposure, white balance, or shutter speed, during runtime. Finally, SSL-Vision contains a very open and extendible architecture, allowing the interchangeability of different image processing "plug-ins". This will allow teams to develop their own improvements and extensions to the system, such as faster image processing algorithms or improved calibration routines. It furthermore allows quick switching and performance comparisons between such plugins.

4 The Attacker Control System

The CMDragons robots perform motion profiling off-board, on the central computer. This raises three problems, namely:

System latency: Latency in the control loop introduces a phase delay between the expected and actual motion profiling. This however is minimized by forward-predicting the observed world state and computing the motion profile on this future state.

Hesitation: Precise motion control can lead to pauses while changing target locations due to switching of motion profiles.

Underperformance: The robot's motion profile is computed using expected robot acceleration and top speeds, although the true values might differ, and in certain cases the robot might actually be capable of exceeding the expected values.

To counter the effect of these problems, we implemented an "Attacker Control System". The Attacker Control System has two main features:

1. The motion profile parameters (acceleration and velocity limits) are separate for AI calculations and for execution. Specifically, the parameters used for AI calculations are more conservative than the true robot parameters, while the execution parameters marginally exceed the true parameters.
2. Intercept and target locations are explicitly modified by a proportional-derivative (PD) controller

The PD controller is implemented as follows. Let the target location of the attacker, as computed by the AI be denoted by l_d. Let the current robot location be denoted by l_r. The modified target location $\tilde{l}_d$ is given by,

$$\tilde{l}_d = l_d + k_p(l_d - l_r) + k_d \frac{d(l_r)}{dt} \tag{1}$$

The proportional and derivative gains k_p, k_d are hand-tuned, and two separate sets are used during the acceleration and the deceleration stages. The modified target location $\tilde{l}_d$ is then used for motion profiling using the execution motion profile parameters.

5 Physics-Based Short-Term Motion Planner

One major problem in Small Size robot soccer is ball manipulation. Traditional navigation planners, such as ERRT, are typically used to provide robot paths or trajectories that are unaware of inter-body dynamics, including ball manipulation. Because these motion planners have no awareness of the ball's dynamics, they tend to generate paths that are unlikely to be dynamically sound in terms of ball dribbling. While the generated paths are safe in terms of avoiding collisions with other robots, it is not likely that the ball will remain in front of the robot when it begins to execute the solution (due to e.g., the build-up of the ball's inertia that was not modeled during planning). To alleviate this problem, we introduce and integrate a short-term *physics-based* motion planner that is aware of multi-body dynamics.

We define the motion planning problem as follows: given a state space X, an initial state $x_{\text{init}} \in X$, and a set of goal states $X_{\text{goal}} \subset X$, a motion planner searches for a sequence of actions $a_1, \ldots, a_n$, which, when executed from x_{init}, ends in a goal state $x_{\text{goal}} \in X_{\text{goal}}$. Additional constraints can be imposed on all the intermediate states of the action sequence by defining only a subset of the state-space to be valid ($X_{\text{valid}} \subseteq X$) and requiring that all states of the solution sequence $x_{\text{init}}, x_1, x_2, \ldots, x_{\text{goal}}$ are elements of X_{valid}.

A *physics-based planner* uses domain models that aim to reflect the inherent physical properties of the real world. The *Rigid Body Dynamics* model [14] provides a computationally feasible approximation of basic Newtonian physics, and allows the simulation of the physical interactions between multiple mass-based non-deformable bodies. The term *Dynamics* implies that rigid body simulators

are second order systems, able to simulate physical properties over time, such as momentum and force-based inter-body collisions. Physics-Based Planning is an extension to *kinodynamic planning* [15], adding the simulation of rigid body interactions to traditional second order navigation planning [16].

5.1 Domain Model and Parameters

A rigid body system is composed of n rigid bodies $r_1 \dots r_n$. A rigid body is defined by two disjoint subsets of parameters $r = \{\hat{r}, \bar{r}\}$ where

- $\hat{r}$: the body's mutable state parameters,
- $\bar{r}$: the body's immutable parameters.

$\hat{r}$ is a tuple, containing the second order state parameters of the rigid body, namely its position, orientation, and their derivatives:

$$\hat{r} = \langle \alpha, \beta, \gamma, \omega \rangle$$

where

- α : position (3D-vector),
- β : orientation (unit quaternion or rotation matrix),
- γ : linear velocity (3D-vector),
- ω : angular velocity (3D-vector).

$\bar{r}$ is a tuple $\bar{r} = \langle \phi_{\text{Shape}}, \phi_{\text{Mass}}, \phi_{\text{MassC}}, \phi_{\text{FricS}}, \phi_{\text{FricD}}, \phi_{\text{Rest}}, \phi_{\text{DampL}}, \phi_{\text{DampA}} \rangle$, describing all of the rigid body's inherent physical parameters, namely: 3D shape, mass, center of mass, static friction, dynamic friction, restitution, linear damping, and angular damping. With the exception to ϕ_{Shape} (which is a 3D mesh or other 3D primitive) and ϕ_{MassC} (which is a 3D vector), all parameters are single finite continuous values.

The physics-based planning state space X is defined by the mutable states of all n rigid bodies in the domain and time t. That is, a state $x \in X$ is defined as the tuple

$$x = \langle t,\ \hat{r}_1, \dots, \hat{r}_n \rangle.$$

The action space A is the set of the applicable controls that the physics-based planner can search over. An action $a \in A$ is defined as a vector of sub-actions $\langle a_1, \dots, a_n \rangle$, where a_i represents a pair of 3D force and torque vectors applicable to a corresponding rigid body r_i.

A physics-based planning domain d is defined as the tuple $d = \langle G, \bar{r}_1 \dots \bar{r}_n, M \rangle$ where

- G : global gravity force vector,
- $\bar{r}_1 \dots \bar{r}_n$: immutable parameters of all n rigid bodies,

A physics-based planner searches for solutions by reasoning about the states resulting from the actuation of possible actions. The state computations are performed by simulation of the rigid body dynamics. There are several robust rigid body simulation frameworks freely available, such as the Open Dynamics Engine (ODE), and NVIDIA PhysX. Frequently referred to as *physics engines*,

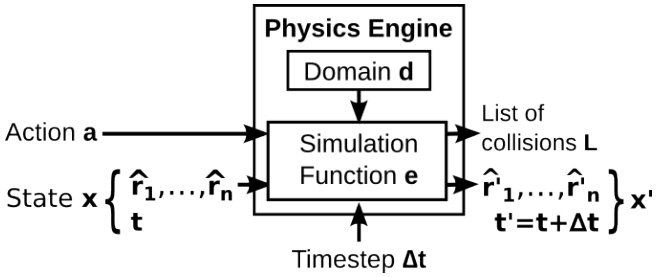

Fig. 3. A physics engine computes state transitions.

these simulators are then used as a "black box" by the planner to simulate state transitions in the physics space (see Figure 3). We define the physics state transition function e:

$$e : \langle \hat{r}_1, \ldots, \hat{r}_n, a, d, \Delta t \rangle \rightarrow \langle \hat{r}'_1, \ldots, \hat{r}'_n, L \rangle.$$

Given a current state of the world x, the simulation function e supplies the contained rigid body states $\hat{r}_1, \ldots, \hat{r}_n$ and a control action vector a to the physics engine. Using the parameters contained in the domain description d, the physics engine then simulates the rigid body dynamics forward in time by a fixed timestep Δt, delivering new states for all rigid bodies $\hat{r}'_1, \ldots, \hat{r}'_n$. These rigid body states are then stored in the new resulting planning state x' along with its new time index $t' = t + \Delta t$. Additionally, e also returns a list of collisions $L = \langle l_1, l_2, \ldots \rangle$ that occurred during forward simulation. Each item $l \in L$ is an unordered pair $l = \langle \lambda_1, \lambda_2 \rangle$, consisiting of the indices of the two rigid bodies $r_{\lambda_1}, r_{\lambda_2}$ involved in the collision.

5.2 Planning and Execution

To efficiently plan in this physics-based space, we use the *Behavioral Kinodynamic Balanced Growth Trees (BK-BGT)* algorithm [17, 16]. It needs to be noted, however, that physics-based planning is computationally extremely expensive, due to the rich detailed simulations of multi-body dynamics. Additionally, because we are planning through a second-order timespace, the search space is extremely large. Effectively, this means that a full search to the goal state (i.e., the ball being in the opponent's goal-box) is infeasible. Furthermore, it is unlikely that such a long-term plan will ever succeed, due to the unpredictability of the opponent team and other factors of uncertainty.

To overcome these issues, we integrate our physics-based planner in a *finite-horizon* fashion where we limit its search to take less than one frame period and therefore limit the resulting tree to several hundred nodes (with $\Delta t = 1/60s$) on the current hardware. We evaluate these partial solutions heuristically, preferring nodes that lead the ball closer to its goal state and further away from opponents. The resulting path is executed for several frames, before the physics-based

planner is invoked again to replan. This replanning interval was tweaked experimentally. More frequent replanning (e.g., on every frame), creates unnecessary oscillations, whereas less frequent replanning struggles with too much build-up in the domain's uncertainty. Figure 4 shows the integration of the physics-based planner ("Planning" indicates planner invokation, "Plan" represents the solution generated by the physics-based planner).

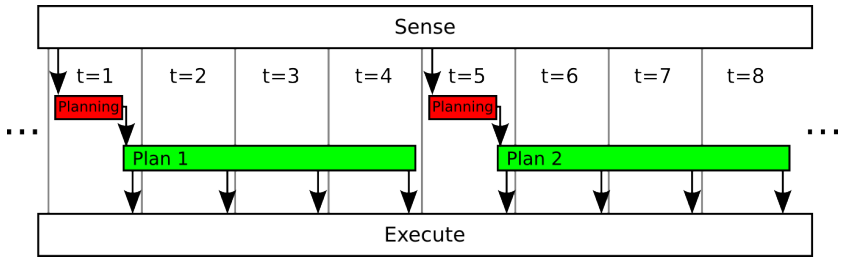

Fig. 4. Physics-based planner integration into the CMDragons execution environment.

6 Opponent state inference through observations of ball dynamics

Teams in the RoboCup Small Size League rely heavily on the existing global localization system. The weakness of this approach became very clear during our quarter-final game at RoboCup 2009. Due to an unexpected bug in our robot server component, the system failed to correctly associate the opponent robot locations reported by SSL-Vision with our system's previous internal opponent state observations. The result was that the system saw opponent robots heavily flickering and moving across the field, even though they were indeed mostly stationary (see Figure 5). Playing with these observations became impossible as our tactical system was not able to deal with this unexpected oscillation in input. Therefore, during our final timeout, we decided to proceed in the game without opponent vision. The result was slightly more stable tactical behavior, but an inability to score, because we did not see their goalie or defender robots. In the end we lost this game 0-1 and were eliminated from the quarter-finals.

The events of this game led us to rethink some of our general sensing strategies and engaged us to develop an experimental system that is able to infer opponent state solely through the observations of the ball's dynamics, without relying on any opponent localization data reported from the vision system. We explain an outline of the algorithm and report first experimental results.

6.1 Algorithm

We introduce the function `PredictPosition` (see Algorithm 1) to detect robot positions from ball state observations. Our approach to the problem assumes

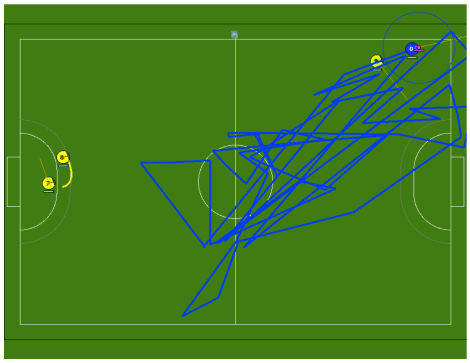

Fig. 5. False motion estimates due to a failure of the server component. Blue lines show falsely interpolated motions of opponent robots.

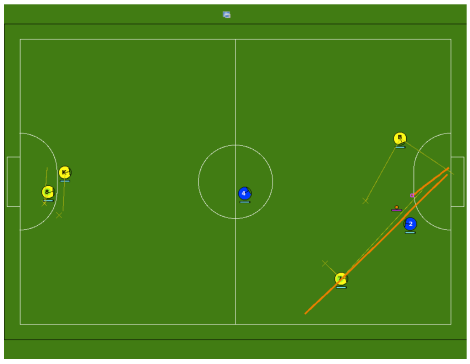

Fig. 6. The ball (orange line) deflecting off of a seemingly invisible obstacle (goalie).

that vision data on the ball is reliable and that the opponents are relatively static. By tracking the ball's position over time, we can calculate the slope and the change in slope. Let k be the number of frames we will examine in our window of time. Each frame lasts 1/60th of a second. For each consecutive pair of frames, let us calculate its slope. For each consecutive pairs of slopes (change in angle), mark as true if the difference between the two slopes is greater than some threshold α. We say a collision occurred if we detected a significant angle change in the middle of the window. We can then estimate the position of the robot based on location of the collision. Figure 6 shows an example of a ball trajectory that would generate a valid estimate through our algorithm.

6.2 Results and Discussion

We ran trials using no vision, perfect vision, and our algorithm with the former two being the control groups. The opponent team consists of three robots: a goalie and two defenders. The goalie would sit stationary at the position (2900,

Algorithm 1: PredictPosition

```
frames[k];
slopes[k − 1];
changes[k − 2];
for i ← 0 to k − 1 do
    frames[i] ← frames[i + 1]; // Make room for new frame
frames[k − 1] ← ballLoc; // Record ball location
for i ← 0 to k − 2 do
    slopes[i] ← CalcSlope(frames[i + 1], frames[i]); // Record slopes
for i ← 0 to k − 3 do
    if abs(slopes[i + 1] − slopes[i]) > α then
        changes[i] ← true; // Deflection occurred
        if i ≠ (k − 2)/2 then
            return false; // More than one deflection occurred
    else
        changes[i] ← false;
if changes[(k − 2)/2] then
    return frames[k/2]; // Deflection detected
else
    return false;
```

0) where (0,0) is the center of the field. The two defenders were placed at (2500, -400) and (2500, 300). The kicker robot shot 50 shots total from 5 different positions. For each shot, we recorded whether or not a goal was made.

As expected, in the trial with no vision, the robot was unable to score a goal reliably. Out of 50 shots, it scored 5 goals by luck when the ball bounced off the goalie and then bounced off a defender into the goal. In the trial with full vision, the robot scored nearly every shot. Out of 50 shots, it missed 5 shots due to system inaccuracies.

When the trial was run using our algorithm, it ran similarly to the trial with full vision. The first shot would miss the goal because it deflected off the goalie but after this "learning phase", all subsequent shots went in. Out of the 50 shots made, 5 only shots were not scored. This is consistent with our hypothesis that we can infer static robot positions solely from ball dynamics observations.

A remaining limitation of this approach is of course that it assumes stationary opponents. Nevertheless, this approach could have been a deciding factor in our 2009 quarter final game, as it would likely have provided a relatively good estimate of the opponent's goalie, thus allowing our attacker to aim into the goal's open area. While we don't anticipate a similar vision failure in the future, the presented approach should be considered as a valid emergency strategy.

Competition	Result
US Open 2003	1st
RoboCup 2003	4th
RoboCup 2004	4th [1]
RoboCup 2005	4th [1]
US Open 2006	1st
RoboCup 2006	1st
China Open 2006	1st
RoboCup 2007	1st
US Open 2008	1st
RoboCup 2008	2nd
RoboCup 2009	Eliminated during quarter-final

Table 1. Results of RoboCup small-size competitions for CMDragons from 2003-09

7 Conclusion

This paper gave a brief overview of CMDragons 2010, covering both the robot hardware and the software architecture of the offboard control system. The hardware has built on the collective experience of our team and continues to advance in ability. The software uses our proven system architecture with continued improvements to the individual modules. The CMDragons software system has been used in three national and seven international RoboCup competitions, placing within the top four teams of the tournament every year since 2003, and finishing 1st in 2006 and 2007. The competition results since 2003 are listed in table 1. We believe that the RoboCup Small Size League is and will continue to be an excellent domain to drive research on high-performance real-time autonomous robotics.

References

1. Bruce, J., Zickler, S., Licitra, M., Veloso, M.: CMDragons 2007 Team Description. Technical report, Tech Report CMU-CS-07-173, Carnegie Mellon University, School of Computer Science (2007)
2. Zickler, S., Vail, D., Levi, G., Wasserman, P., Bruce, J., Licitra, M., Veloso, M.: CMDragons 2008 Team Description. In: Proceedings of RoboCup 2008
3. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-Vision: The Shared Vision System for the RoboCup Small Size League. RoboCup 2009: Robot Soccer World Cup XIII (2009) 425–436
4. Browning, B., Bruce, J.R., Bowling, M., Veloso, M.: STP: Skills tactics and plans for multi-robot control in adversarial environments. In: Journal of System and Control Engineering. (2005)
5. Bowling, M., Browning, B., Veloso, M.: Plays as effective multiagent plans enabling opponent-adaptive play selection. In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04). (2004)

[1] Provided software component as part of a joint team with Aichi Prefectural University, called CMRoboDragons

6. Bruce, J.R., Bowling, M., Browning, B., Veloso, M.: Multi-robot team response to a multi-robot opponent team. In: Proceedings of the IEEE International Conference on Robotics and Automation, Taiwan (May 2003)
7. Bruce, J.R., Veloso, M.: Real-time randomized path planning for robot navigation. In: Proceedings of the IEEE Conference on Intelligent Robots and Systems. (2002)
8. LaValle, S.M., James J. Kuffner, J.: Randomized kinodynamic planning. In: International Journal of Robotics Research, Vol. 20, No. 5. (May 2001) 378–400
9. James J. Kuffner, J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation. (2000)
10. Bruce, J.R., Veloso, M.: Safe multi-robot navigation within dynamics constraints. Proceedings of the IEEE **94** (July 2006) 1398–1411
11. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics and Automation Magazine **4** (March 1997)
12. Brock, O., Khatib, O.: High-speed navigation using the global dynamic window approach. In: Proceedings of the IEEE International Conference on Robotics and Automation. (1999)
13. Bruce, J.R.: Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments. PhD thesis, Carnegie Mellon University (Dec 2006)
14. Baraff, D.: Physically Based Modeling: Rigid Body Simulation. SIGGRAPH Course Notes, ACM SIGGRAPH (2001)
15. Donald, B., Xavier, P., Canny, J., Reif, J.: Kinodynamic motion planning. Journal of the ACM (JACM) **40**(5) (1993) 1048–1066
16. Zickler, S., Veloso, M.: Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. (2009) 27–33
17. Zickler, S., Veloso, M.: Tactics-Based Behavioural Planning for Goal-Driven Rigid-Body Control. Computer Graphics Forum **28**(8) (2009) 2302–2314