CMDragons 2010 Extended Team Description

Stefan Zickler, Joydeep Biswas, James Bruce,
Michael Licitra, and Manuela Veloso

Carnegie Mellon University
{szickler,jbruce,veloso}@cs.cmu.edu
{joydeep,mlicitra}@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. Our team's success can be contributed to a well-balanced combination of sophisticated robot hardware and a layered, intelligent software architecture consisting of computer vision, multi-robot play selection, as well as single robot tactics and navigation.

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 controlled by an offboard computer. Sensing is provided by two overhead mounted cameras via the SSL-Vision system. This first half of this paper describes the robot and vision hardware of the system. The second half of this paper describes the overall architecture and individual components of the offboard control software required to implement a robot soccer team.

2 Hardware

The modern Small Size League has brought along many challenging demands on hardware. To perform well as a team, robots need to be reliable, fast, and accurate in their actuations. Similarly, global vision sensing needs to not only be low-latency, but also needs to have sufficient resolution to accurately localize the robots and the ball on the relatively large field. This Section describes these hardware components of our system.

2.1 Vision

In previous years, the CMDragons team has relied heavily on traditional off-the-shelf camcorders in conjunction with low-cost capture cards as its vision system. Our previous hardware delivered a 640×240 pixel video stream at 60Hz. The

continued increases in field size however, have led this system to reach its limits in terms of resolution. Another limitation of this traditional vision hardware is the inability to modify video settings without physically interacting with the camera which is mounted above the playing field. To overcome these issues, CMDragons 2010 features two new IEEE 1394B (Firewire 800) cameras (AVT Stingray F-46C) which provide a 780×580 YUV422 progressive video stream at 60Hz, thus delivering roughly three times as much data throughput as our previous system. Each camera is connected to the central computer via two dedicated PCI-Express IEEE 1394B cards. The images delivered by the cameras are then processed by the vision software which is described in detail in Section 3.1.

2.2 Robots

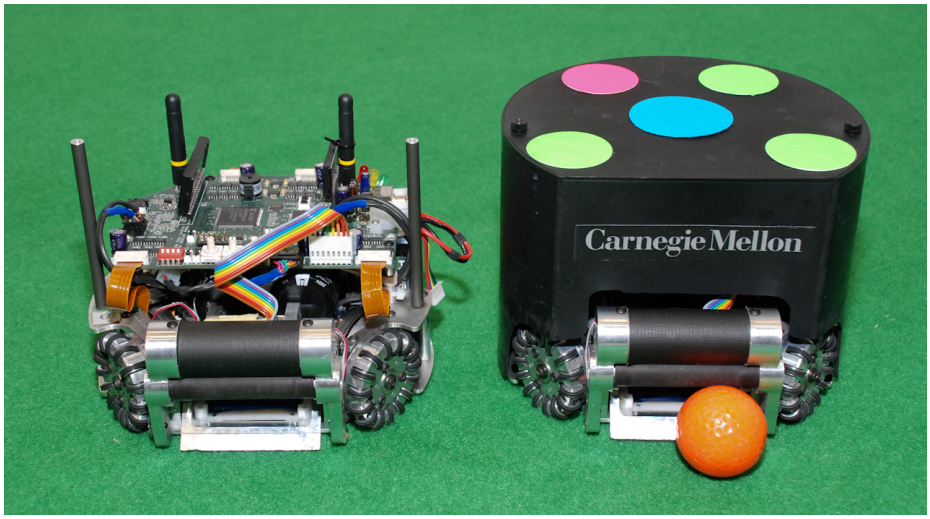

Fig. 1. A CMDragons robot shown with and without protective cover.

Our team consists of seven homogeneous robot agents, with five being used in a game at any point in time. In Figure 1, an example robot is shown with and without a protective plastic cover. The hardware is 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 noticeable 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.

The robot drive system and kicker are shown in Figure 2. Each robot is omni-directional, with four custom-built wheels driven by 30 watt brushless motors.

Each motor has a reflective quadrature encoder for accurate wheel travel and speed estimation. The kicker is a large diameter custom wound solenoid attached directly to a kicking plate, which offers improved durability compared to designs using a standard D-frame solenoid. The kicker 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 in the style of FU-Fighter's 2005 robot design. It is a custom-made flat solenoid located under the main kicker, which strikes an angled wedge visible at the front bottom of the robot. The wedge is hinged and travels a short distance at a 45% angle from the ground plane, driving the ball at a similar angle. 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 $200V$. The capacitors are located directly above the kicker and below the electronics, leading to our unusual mechanical design which lacks a single connected "midplate". By using a slightly thicker baseplate, and several partial midplates with multiple standoffs for support, we were still able to design a highly robust robot.

Ball catching and handling is performed by a motorized rubber-coated dribbling bar which is mounted on an hinged damper for improved pass reception. The dribbling bar is driven by a brushless motor so that it can achieve a high speed without sacrificing torque. The hinged damper can also be retracted using a small servo. This is used during certain kicking maneuvers, so that the dribbling bar does not interfere with speed or accuracy.

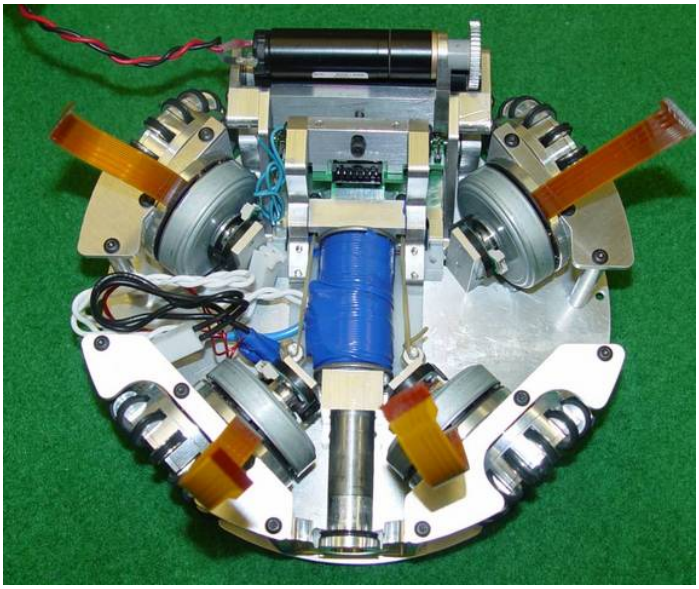

Fig. 2. Top view of the robot drive system, kicker, and dribbler.

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.

The robot electronics consists of an ARM7 core running at 58MHz linked to a Xilinx Spartan2 FPGA. The ARM7 core handles communication, runs the PD drive control calculations, and monitors onboard systems. The FPGA implements the quadrature decoders, PWM generation, serial communication with other onboard devices, and operates a beeper for audible debugging output. Figure 3 shows the main electronics board which integrates all electronic components except for a separate kicker board and IR ball sensors. This high level of integration helps to keep the electronics compact and robust, and helps to maintain a low center of gravity compared to multi-board designs. Despite the limited area, a reasonable amount of onboard computation is possible. Specifically, by offloading the many resource intensive operations onto the FPGA, the ARM CPU is freed to perform relatively complex calculations.

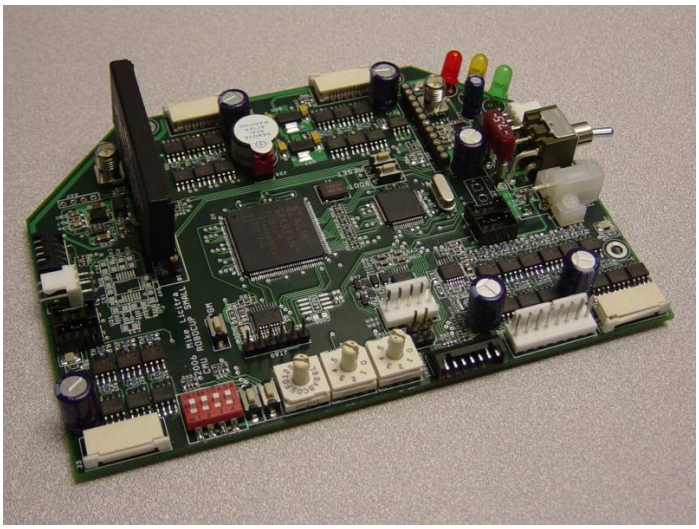

Fig. 3. The main robot electronics board for CMDragons 2010.

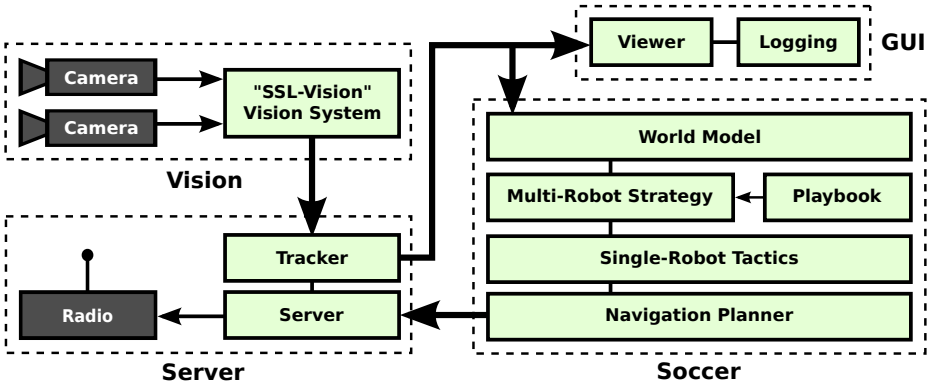

Fig. 4. The general architecture of the CMDragons offboard control software.

3 Software

The software architecture for our offboard control system is shown in Figure 4. Our entire software architecture has been written in C++ and currently runs under Linux. It follows the same overall structure as has been used in the previous year, outlined in [1, 2]. The major organizational components of the system are a server program which receives the vision system's output and manages communication with the robots, and two 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 details of the soccer program are described in the following Sections.

The server program consists of tracker, radio, and a multi-client server. The tracker receives the raw positions and orientations from the SSL-Vision system (see Section 3.1). 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 [6]. Final commands are communicated by the server program using a radio link as described in Section 3.2. Alternatively to running the server which observes and interfaces with the actual robots, it is also possible to employ the simulator which appears semantically identical to the server, but uses an internal, physics-based world model. The details of this simulator are described in Section 3.3.

3.1 Vision

CMDragons 2010 operates using SSL-Vision as its vision system [7]. 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 4 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.

3.2 Wireless Communication

Wireless communication with the robots is performed in the 902-927 MHz frequency range through the Linx HP3 module connected via a standard serial port. The CMDragons robots are fully equipped with both receivers and transmitters which provides them with bi-directional communication abilities, thus not only allowing the central software to control the robots, but also allowing the control software to receive IR sensor feedback and battery status from the robots. However, due to the added latency of these robot observations, the system relies mostly on one-way communication for control only. Sensor observations however, are used if vision data becomes unavailable (e.g. in case of occlusion).

3.3 Simulator

Being able to accurately simulate robot behaviors is an integral part of our team's development cycle. Simulation allows rapid testing of new code without the need to impose drain on our robotic hardware. Additionally, it allows us to simulate scenarios which we are unable to recreate by our limited number of physical robots, such as full five-on-five RoboCup games. In recent years, our ball-manipulation capabilities and behaviors have become very sophisticated. Our strategies often rely on complex dynamics interactions such as deflecting a ball off from opponents. The introduction of chip-kicks has added the additional requirement of three-dimensional simulation. Finally, many of our most recent behaviors rely heavily on our robots' ball-dribbling capabilities, and thus should also be accommodated in a simulated model.

In order to model such behaviors, we use a simulator that is able to compute complex rigid body dynamics. Various free and commercial simulation engines exist to perform this task. We choose NVIDIA's PhysX as our simulation engine due to its performance and ease of use, but other candidates such as e.g. the Open Dynamics Engine (ODE) might be equally suited. The concept of such simulation engines is that, given a scene of rigid-bodies and a set of initial positions and velocities, we can apply forces and torques to arbitrary objects in the scene. The engine will then integrate the scene forward in time, automatically resolve any rigid-body interactions (such as collisions) using basic Newtonian physics, and finally provide us with a new state of positions and velocities. Our simulator is able to act as a full replacement of the standard server depicted in Figure 4, thus processing the soccer-system's commands and returning the newly observed state of the world.

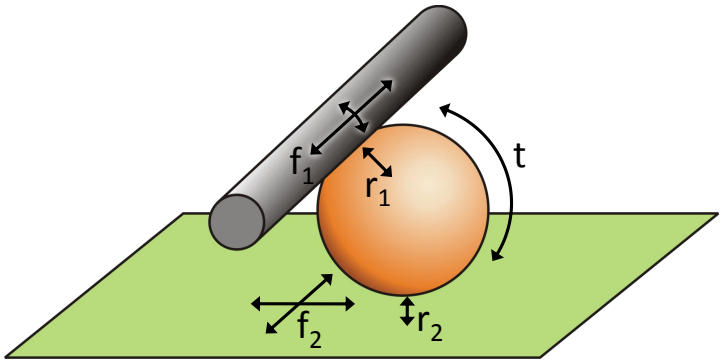

Fig. 5. A diagram of the simulated dribbling model and its parameters.

Our robots are modeled using a convex triangulated shell in combination with other convex rigid-body primitives. The golf-ball is approximated by using a simple sphere. In order to achieve accurate dynamics simulations, it is important that we have an adequate model of our robot's ball-handling capabilities. One particularly challenging issue is to correctly model the robots' dribbler-bar. One approach is to model the dribbler implicitly, by defining a motorized, rotating, high-friction cylinder which is connected to the robot by using a rotary joint. In practice however, this approach failed to accurately model the dribbling behavior, possibly due to the accumulation of joint-error and an insufficient simulated torque transfer between the bar and the ball. The approach we took instead was to model the dribbling action explicitly: if the ball is in contact with the dribbler-bar then we directly apply a torque to the ball facing towards our robot. The exact dribbling behavior can be controlled by modifying the various variables that affect the ball's dynamics. A diagram of the dribbling model is shown in Figure 5. The applied torque is depicted as t. Variables that affect our ball, are the friction in relation to the floor f_2 and in relation to the dribbler bar f_1.

The dribbler bar uses an anisotropic friction model, defining low friction against the vertical direction of the bar (thus, roughly simulating a freely spinning bar), and higher friction against the horizontal direction of the bar (thus modeling the high friction of the bar's rubber). In addition to friction, we can control the ball's coefficient of restitution against the dribbler and the floor respectively. Naturally r_1 will have a fairly low coefficient, thus roughly modeling the dribbler's softness and dampening, whereas r_2 will have slightly higher coefficient to model the bounciness of the carpet. Additional constants in our physics model are angular and linear damping of both the robot and ball, thus simulating air-drag. The robot's motions are simulated explicitly as well. Instead of modeling the wheels which will impose friction on the floor, we model the robot as having a flat, low-friction base-plate. We then directly apply forces and torques to the robot to simulate its motions. Kicks and chip-kicks are simulated in a similar fashion, by exerting linear impulses directly on the ball. A simulated sequence of one of our robots performing a "dribble and fling"-maneuver can be found in Figure 6.

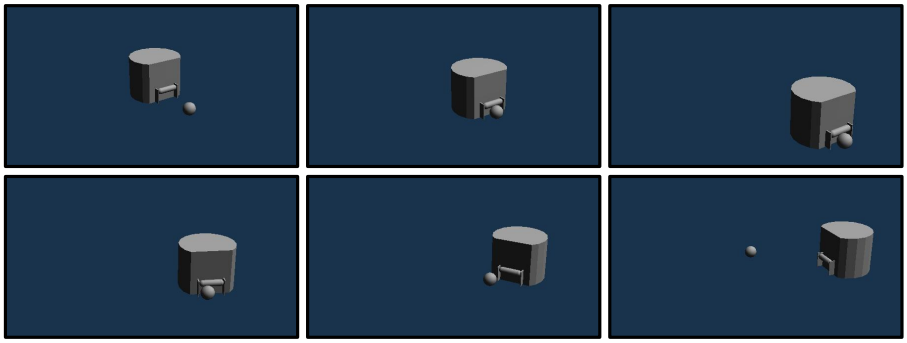

Fig. 6. A simulation sequence of a "dribble-and-fling" behavior.

It should be noted that currently, all of the variables have been tweaked experimentally to obtain a relatively accurate model of the robot. In the future, it will be interesting to employ supervised learning techniques to automate this task.

3.4 Behavior

The three major components of the soccer behavior control system are: (1) world state evaluation and play selection; (2) skills and tactics; (3) navigation.

1. World state evaluation involves determining high level states about the world, such as whether the team is on offense or defense. This allows us to select among a set of possible plays [8]. It also involves finding and ranking possible subgoals, such as evaluating a good location to receive a pass.

2. Tactics and skills implement the primitive behaviors for a robot, and can range from the simple "go to point" skill, up to complex tactics such as "pass" or "shoot". A *role* in our system is defined as a tactic with all parameters fully specified, which is then assigned to a robot to execute. The executed tactic generates a navigation target either directly or by invoking lower level skills.
3. The navigation module takes the targets specified by tactics and plans a path using a randomized path planner. The path is then processed by motion control and dynamic obstacle avoidance to generate robot velocity commands. These resulting commands are sent to the robots by a radio link.

In order to take advantage of opportunities as they arise in the dynamic environment of a competitive soccer game, the entire control system executes at 60 times a second, synchronously with the vision system. This requires all of the cooperating components to run within a real-time constraint.

Teamwork Multi-robot domains can be categorized according to many different factors. One such factor is the underlying parallelism of the task to be achieved. In highly parallel domains, robots can complete parts of the task separately, and mainly need to coordinate to achieve higher efficiency. In a more serialized domain, some part of the problem can only be achieved by one robot at a time, necessitating tighter coordination to achieve the objective efficiently. Robotic soccer falls between parallel and serialized domains, with brief periods of joint actions. Soccer is a serialized domain primarily due to the presence of a single ball; At any given time only one robot from a team should be actively handling the ball. In a domain such as soccer, multi-robot coordination methods need to reason about the robot that actively addresses the serial task and to assign supporting objectives to the other members of the team. Typically, there is one *active* robot with multiple robots in *supporting* roles. These supporting roles add a parallel component to the domain, as supporting robots can execute actions in a possibly loosely coupled way to support the overall team objective.

Multi-Robot Role Assignment A large number of methods exists for task assignment and execution in multi-agent systems. [9] provides an overview and taxonomy of task assignment methods for multi-robot systems. [10] points out the direct conflicts that can arise between multiple executing behaviors, as well as complications arising when the number of tasks does not match the number of robots. [11] present a cooperation method that handles tight coordination in robot soccer using messaging between the behaviors controlling each agent. The STP architecture [6] describes another method of cooperation using centralized plays and communicating single-robot behaviors.

In addition to the assignment of tasks to agents, their still lies the problem of describing how each agent should implement its local behavior. [12] presents an artificial potential field for local obstacle avoidance. There are limitations of direct local execution of potential functions however [13], including local minima in

the potential field gradient and the inability to represent hard action constraints. The SPAR algorithm [14], describes a method combining binary constraints with linear objective functions to define a potential over the workspace, but is only used for navigation target selection rather than direct actions.

Task allocation in our system follows the STP model [6]. Our systems adopts a split of active and support roles and solves each of those subtasks with a different method. Active roles which manipulate the ball generate commands directly, receiving the highest priority so that supporting roles do not interfere. Supporting roles are based on optimization of potential functions defined over the configuration space, as in SPAR and later work. With these two distinct solutions, part of our system is optimized for the serialized aspect of ball handling, while another part is specialized for the loosely coupled supporting roles. We address the need for the even more tight coupling that is present in passing plays through behavior dependent signalling between the active and supporting behaviors as in [11].

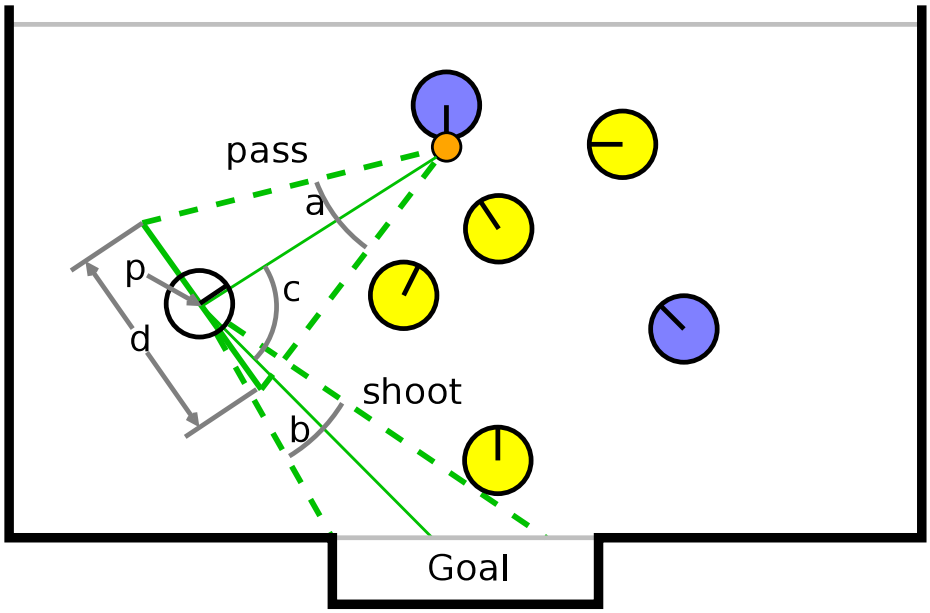

Fig. 7. An example demonstrating the pass evaluation function inputs. The input values for evaluating a point p are the subtended angle a of the reachable area d, the unobstructed goal angle b, and the angle e between the pass and shoot centerlines. These are combined in specific evaluation functions to achieve the desired behavior.

Objective Evaluation Functions In our system, the navigation targets of supporting roles are determined by world state evaluation functions defined over

the entire soccer field. Each function holds the world state external to a robot constant, while varying the location of the robot to determine a real valued evaluation of that state within the workspace. Hard constraints are represented using multiplicative boolean functions, whereas soft constraints are modelled as general real-valued functions.

Passing Evaluation An example of the general form of the pass position evaluation function is given in Figure 7. There are several variables given the current world state and a point p to evaluate:

- d is the reachable area, which is defined by the perpendicular distance the robot can travel in the time it takes a pass to reach a perpendicular centered on p.
- a is the subtended angle of d from the current ball location
- b is the angular width of the free angle toward the goal from point p
- c is the angle between the pass from the ball to p, and the shot from p to the center of the free angle on the goal

Using this general model, we can define several types of passes:

- A one-touch pass-and-shoot which intercepts the moving ball to kick it at the goal. This estimates the a time t as the pass length divided by the pass speed, plus the shot length divided by the shooting speed. An angular preference $k(e)$ is calculated which increases linearly from 0 at $e = 0$ to 1 at $e = 45°$. It stays at 1 until $e = 90°$, where is decreases rapidly to 0. The evaluation is then $[k(e) \min(a, b)/t]$.
- A pass-receive-turn-shoot which explicitly receives the ball and then aims and shoots. The estimates the t as the sum of the pass time, turning time for e, and shot time. The evaluation using this t is $[\min(a, b)/t]$.
- Partial chip-pass variants of the above passing methods, where a chip shot is used to pass partway to the receiver, but dropping soon enough that it will roll by the time it reaches the receiver. These are used when a robots blocks the direct passing path.
- A direct chip deflection "header". Here a chip pass is calculated to a reachable robot position (a part of d), with a target point that passes over d at a height midway up the robot. The robot will deflect the ball directly into the goal, so the pass and shoot speed are identical. The evaluation is then identical to the one-touch pass-and-shoot evaluation.

The resulting plots from two example passing situations are shown in Figure 8, with the pass evaluation function for one-touch pass-and-shoot overlaid on a field. The location of the ball and other robots causes very different target locations to be selected by the supporting robot. Because large portions of the field are covered by a continuous evaluation, the existence of an unobstructed maximum is likely.

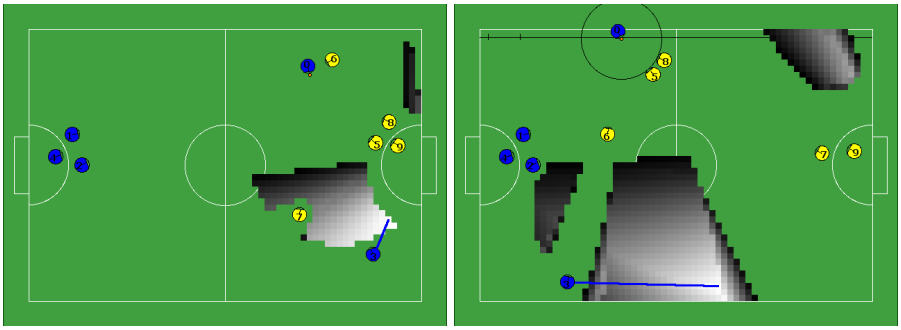

Fig. 8. Two example passing situations are shown with the passing evaluation metric overlaid. The values are shown in grayscale where black is zero and the maximum value is white. Values less than 0.5% of the maximum are not drawn. The maximum value is indicated by the line extending from the supporting robot.

Properties While the exact parameters and weights applied in evaluation functions are highly application dependent, and thus not of general importance, the approach has proved of useful throughout many revisions of our system. In particular, with well designed functions, the approach has the useful property that large areas have a nonzero evaluation. This provides a natural ranking of alternative positions so that conflicts can be resolved. Thus multiple robots to be placed on tasks with conflicting actions, or even the same task; The calculation for a particular robot simply needs to discount the areas currently occupied by other robots. Direct calculation of actions, as is used for active roles, does not inherently provide ranked alternatives, and thus leads to conflicting actions when other robots apply the same technique.

3.5 Action Execution

Numerous low level skills are required to implement a robot soccer team. Many of the skills such as shooting on a goal and described in other work on robot soccer, such as in [6]. In this section we describe some of the more unique parts of our system, namely a new control approach for the "Attacker" tactic, a "delta-margin" metric for choosing when to kick a ball, and the "one-touch pass-and-shoot" method to achieve the robot soccer equivalent of the "one-touch" goal shot from human soccer.

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.

Delta-Margin Shooting Metric The shooting metric is a skill that must determine the appropriate time of energizing the kicking device to execute an aimed shot. The input to the shooting metric is a range of angles $[g_0, g_1]$ representing the target, along with the robot's current angle $\alpha(t)$. This data is provided each frame as a data stream, and the output of the metric is the a binary value of whether to kick or not. During this time, the robot will position itself to aim at the center of the angular range. This problem is similar to an automated assembly task where a part is positioned and then dropped into place. In both cases, there is a tradeoff between probability of correct placement (i.e. within tolerance), and the time used in positioning. Ideally we would like something that maximizes the probability of correct placement while minimizing the total time taken in positioning. In the robot soccer environment, this is complicated by the fact that the target angles can change as a function of time $[g_0(t), g_1(t)]$. This situation is similar to an assembly task where both parts to be mated are moving.

Our method for shooting relies on the assumption that the probability of shooting within the tolerance is proportional to the margin, where the margin is defined as the angular distance to the nearest edge of the target range. Formally,

we can define the margin function $m(t)$ as shown in Equation 2, which in turn depends on the angle normalization function N.

$$m(t) = \max\left[N(\alpha(t) - g_0(t)), N(g_1(t) - \alpha(t))\right] \tag{2}$$

$$N(a) = \begin{cases} N(a + 2\pi) & \text{if } a < -\pi, \\ N(a - 2\pi) & \text{if } a > +\pi, \\ a & \text{otherwise.} \end{cases} \tag{3}$$

Using the definition of $m(t)$, we can define the binary shooting function $S(t)$ as shown in Equation 4. The first case will prevent shooting unless the margin is within the tolerances The second case case will shoot when the margin is nearer to the optimal margin than the constant fraction β (we use $\beta = 0.9$). The third case, which is the primary contribution of this method, prevents shooting as long as the margin is increasing with time. In all remaining cases the metric will elect to shoot.

$$S(t) = \begin{cases} 0 & \text{if } m(t) < 0, \\ 1 & \text{if } m(t) > \beta(g_1(t) - g_0(t))/2, \\ 0 & \text{if } m(t) > m(t-1), \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

This method has worked extremely well in practice, as it appears to strike a good balance between the conflicting options of shooting as early as possible (to capture short-lived opportunities) and waiting to improve the aim (thus lower the probability of missing the target.) Though simple to compute, the method captures all of the following qualitative properties:

- Try to aim for the center of the target angular range.
- If an angular range is widening, delay shooting since the miss probability is dropping with time.
- If an angular range is narrowing, take the shot since the miss probability is increasing with time.
- If aiming past a moving object (such as a goalkeeper), delay shooting iff our goal probability is improving faster than the opponent is decreasing it.

Two examples of the shooting method executing on a real robot are shown in Figures 9 and 10, with the relevant variables plotted over time until the kick is taken. The experiment setup was a single robot 1.5m from an open goal. In the first example, the margin increases to the maximum, and the kick is taken due to the zero crossing of the margin delta. In the second example, the margin stops improving so the shot is taken before the maximum (the ball was rolling away from the robot causing its aim to stop improving).

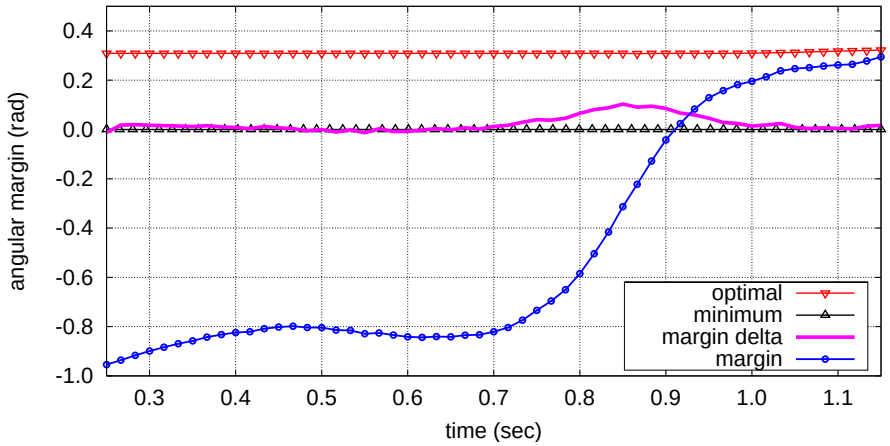

Fig. 9. An example plot of the delta-margin shooting metric reaching a maximum margin. Shot is taken at $t = 1.15$.

One-touch Aiming The one-touch pass-and-shoot skill is a method for intercepting a moving ball to shoot it at the goal, and corresponds to a "one-touch" strike in human soccer. This skill combines our existing ball interception and target selection routines with a method for determining the proper angle to aim the robot to accurately redirect the ball to the target. In order to calculate the proper aiming angle, a model of how interaction with the kicker will affect the speed of the ball is needed. In particular, while the kicker adds a large forward component to the ball velocity, effects from the ball's original (incoming) velocity are still present and non-negligible.

The system model is shown in Figure 11. R_h and R_p represent the normalized robot heading and perpendicular, respectively. After exploring numerous options to determine the final ball velocity (v_1), the model chosen was a weighted sum of three components:

- Initial ball velocity v_0 damped by the dribbling device. This is a tangential velocity along R_p, or $(R_p \cdot v_1)R_p$, which is scaled by a damping factor $\beta \in [0, 1]$.
- Initial ball velocity v_0 reflected by the robot heading R_h. This is expressed as vector reflection of v_0 by R_h, scaled by a constant $\gamma \in [0, 1]$.
- Additive velocity imparted by the kicker along R_h. The kicker provides an impulse that would propel a ball at rest to speed k (i.e. $\|v_0\| = 0 \rightarrow \|v_1\| = k$). Because the kicker is attached to the moving robot, k is the sum of the kicking device speed and the speed of the robot along R_h.

Using this model, we can estimate v_1 as:

$$\hat{v_1} = \beta(R_p \cdot v_0)R_p + \gamma(v_0 - 2(v_0 \cdot R_h) \cdot R_h) + kR_h \tag{5}$$

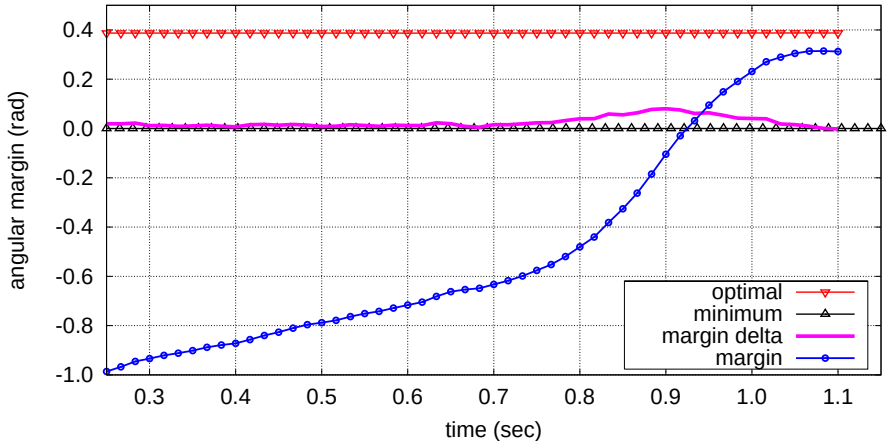

Fig. 10. An example plot of the delta-margin shooting metric reaching a zero-crossing of the margin delta. Shot is taken at $t = 1.1$.

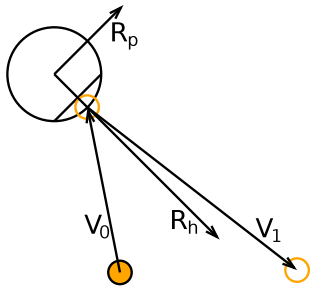

Fig. 11. The model for one-touch pass-and-shoot. The final velocity of the ball v_1 contains a component of the initial velocity v_0, and thus is not parallel to the robot heading R_h.

We determined the model parameter values experimentally, by recording the incoming and outgoing velocity at a variety of angles and kick speeds, and calculating the most likely values. We found that $\beta = 0.1$ and $\gamma = 0.5$ best modelled the data, but these values are likely to be dependent on the exact robot design, and can even be affected by the field surface. The calibration procedure is straightforward however, so we have not found this to be a major limitation.

Of course, the forward model alone is not sufficient, as the control problem requires solving for the robot angle given some relative target vector g. To invert the model, we use bisection search. The bounding angles for the search are the original incoming ball angle (where the residual velocity component would be zero) and the angle of target vector g (where we would aim for an ideal kicker with total damping ($\beta = 0, \gamma = 0$). The actual solution lies between these limits,

and we can calculate an error metric e by setting up Equation 5 as a function of the robot angle α.

$$\begin{aligned} R_h(\alpha) &= \langle \cos\alpha, \sin\alpha \rangle \\ R_p(\alpha) &= \langle -\sin\alpha, \cos\alpha \rangle \\ \hat{v_1}(\alpha) &= \beta(R_p(\alpha) \cdot v_0)R_p(\alpha) + kR_h(\alpha) + \\ &\quad \gamma(v_0 - 2(v_0 \cdot R_h(\alpha)) \cdot R_h(\alpha)) \qquad (6) \\ e(\alpha) &= \hat{v_1}(\alpha) \cdot g \qquad (7) \end{aligned}$$

Thus when $e(\alpha) > 0$ the solution lies with α closer to g, while if $e(\alpha) < 0$ the solution is closer to v_0. A solution at the value of α where $e(\alpha) = 0$, so bisection search is simply terminated whenever $\|e(\alpha)\| < \epsilon$. While it is possible to invert the model so that search is not required, using a numerical method for determining α allowed rapid testing of different models and parameters. The complexity of the approximation is $O(\log(1/\epsilon))$, which has proven adequately fast in practice.

We have found the one-touch aiming method to work with passes between $2m/s$ and $4m/s$, and has proven quite a bit faster than a more explicit receive-turn-shoot approach. The main limitation of the approach is that the angle between the pass and the shot on goal should generally lie between 30 and 90 degrees. The receive-turn-shoot approach can be used in cases where the angle is not amenable to the pass-and-shoot approach.

We also adapted the 2D version of one-touch aiming to the 3D problem of soccer "headers". The chip kicker is used to kick the ball in the air, and a dynamics model of the ball fit a parabolic trajectory to the observed ball position. This allows a robot to intercept a ball while still in the air to deflect it into a goal. Because the kicker is not used, the model for aiming is pure reflection ($\beta = 0, \gamma = 1.0$). The interception method used is to drive to the point where the ball will reach a particular height above the ground (halfway up the flat part of the robot's protective cover). Due to the decreased accuracy of chip kicks, this type of pass normally does not allow the receiving robot to remain at a fixed location, and depends heavily of the receiving robot adjusting to intercept the ball. Despite the low probability of a successful pass compared to other methods $P[\text{success}] = 0.3$, when it succeeds it has a high chance of scoring as it leaves little time for the receiving team to react.

3.6 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

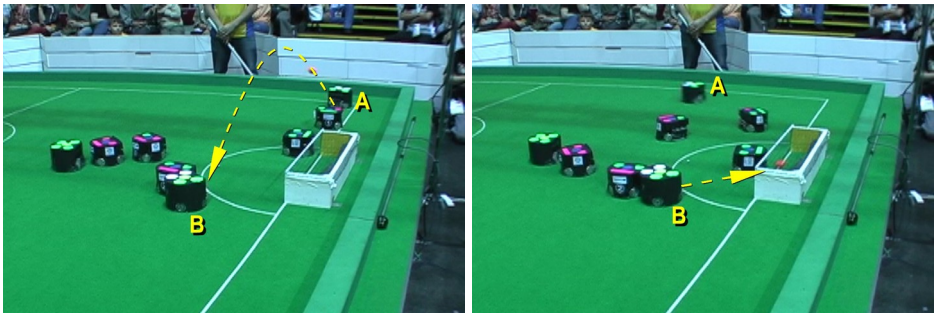

Fig. 12. An example header from the RoboCup 2006 semi-final match. Robot A passes to robot B, which then deflects the ball into the goal. Robot B chooses its angle using one-touch aiming in order to deflect into the gap left by the goalkeeper.

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 [15] 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* [16], adding the simulation of rigid body interactions to traditional second order navigation planning [17].

Domain Model and Parameters A rigid body system is composed of n rigid bodies $r_1 \ldots r_n$. 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

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

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 $\langle a_1, \ldots, a_n \rangle$, 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 $d = \langle G, \bar{r}_1 \ldots \bar{r}_n, M \rangle$ where

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

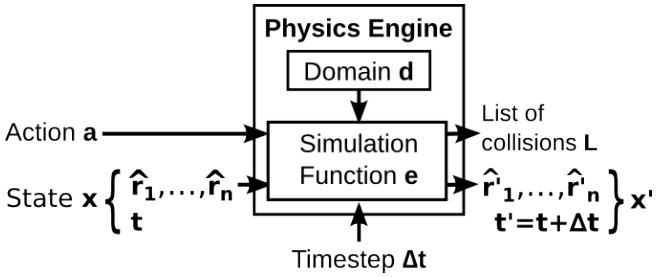

Fig. 13. A physics engine computes state transitions.

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*, these simulators are then used as a "black box" by the planner to simulate state

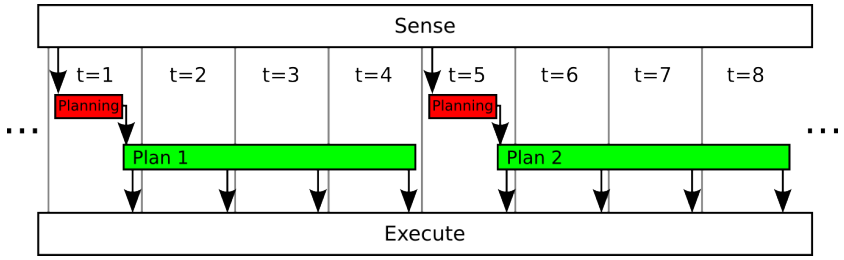

Fig. 14. Physics-based planner integration into the CMDragons execution environment.

transitions in the physics space (see Figure 13). 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.

Planning and Execution To efficiently plan in this physics-based space, we use the *Behavioral Kinodynamic Balanced Growth Trees (BK-BGT)* algorithm [18, 17, 19]. 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 14 shows the integration of the physics-based planner ("Planning" indicates planner invokation, "Plan" represents the solution generated by the physics-based planner). A complete description and evaluation of this physics-based planning approach is available in [19].

3.7 Navigation Planning

The motion planner adopted for our navigational system is based on the RRT family of randomized path planners. Our latest variant adopts additional modified features from RRT-Connect, and is called *Bidirectional Multi-Bridge ERRT* [20]. A basic RRT planner search for a path from an initial state to a goal state by expanding a search tree. Table 1 shows the pseudo-code for a basic RRT motion planner. It relies on three primitives which are defined for the domain. First, the *Extend* function calculates a new state that can be reached from the target state by some incremental distance (usually a constant distance or time), which in general makes progress toward the goal. If a collision with an obstacle in the environment would occur by moving to that new state, then a default value, EmptyState, of type state is returned to capture the fact that there is no "successor" state due to the obstacle. Next, the function *Distance* needs to provide an estimate of the time or distance (or any other objective that the algorithm is trying to minimize) that estimates how long repeated application of *Extend* would take to reach the goal. Finally, *RandomState* returns a state drawn uniformly from the state space of the environment. For a simple example, a holonomic point robot with no acceleration constraints can implement *Extend* simply as a step along the line from the current state to the target, and *Distance* as the Euclidean distance between the two states.

The RRT-Connect variant of RRT has demonstrated high performance for one-shot queries when compared to other motion planners [21]. It combines bidirectional search with an iterated extension step. In RRT-Connect, a random target is chosen just as with a base RRT planner. However, instead of calling the extend operator once, the RRT-Connect repeats the extension until either the target point is reached, or the extension fails (such as when it would hit an obstacle). The search is performed bidirectionally, with a tree growing from both the initial and goal configurations. In each step, after a random target point is chosen, one tree repeatedly extends toward that target, reaching some final configuration q'. The second tree then repeatedly extends toward q' (as opposed to the random target), in an operation referred to as *Connect*. After each iteration, the tree swaps roles for extending and connecting operations, so that both the initial and goal trees grow using both operations.

While these improvements can markedly improve RRT's one-shot planning time, they do have an associated cost. While RRT, with its single extensions, tends to grow in a fixed rate due to the step size, RRT-Connect has a much higher variance in its growth due to the repeated extensions. As a result, when planning is iterated, RRT-Connect tends to find more widely varying homotopic paths. This is not an issue for one-shot planning, but can become a problem for

```
function RRTPlan(env:Environment, initial:State, goal:State) : RRTTree
1     var nearest,extended,target : State
2     var tree : RRTTree
3     nearest ← initial
4     tree ← initial
5     while Distance(nearest,goal) < threshold do
6         target ← ChooseTarget(goal)
7         nearest ← Nearest(tree,target)
8         extended ← Extend(env,nearest,target)
9         if extended ≠ EmptyState
10            then AddNode(tree,extended)
11    return tree

function ChooseTarget(goal:State) : State
1     var p : ℝ
2     p ← UniformRandom(0,1)
3     if p ∈ [0, GoalProb]
4         then return goal
5     else if p ∈ [GoalProb, 1]
6         then return RandomState()

function Nearest(tree:RRTTree,target:State) : State
1     var nearest : State;
2     nearest ← EmptyState;
3     foreach State s ∈ tree do
4         if Distance(s,target) < Distance(nearest,target)
5             then nearest ← s;
6     return nearest;
```

Table 1. The basic RRT planner stochastically expands its search tree to the goal or to a random state.

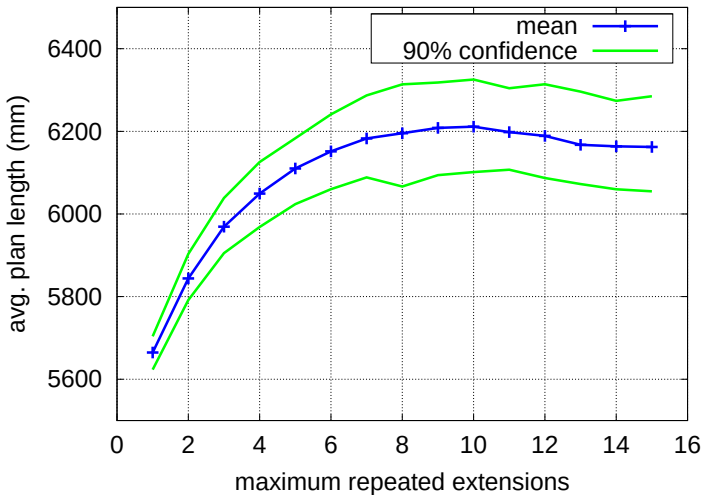

Fig. 15. The effect of repeated extensions in ERRT on plan length.

iterated planning with interleaved execution. Thus ERRT adopts the improvements of RRT-Connect, but modified somewhat. First, ERRT supports repeated extensions, but only up to some maximum constant, which can be tuned for the domain. Figure 15 shows the effect of this parameter on the average plan length for a domain. Each datapoint is includes the mean and confidence interval for 300 test runs, where each run represents 240 iterated plans in a small-size domain with 64 random rectangular obstacles. The initial and goal positions were varied with sinusoidal motion. As the number of extensions increases, the plan average length grows. While applications can apply smoothing to remove some of the inefficiency of less optimal plans, a shorter plan is more likely to be in the same homotopy class as the optimal plan. Thus plan length is at least one indicator of the reachable length even after smoothing, and stability in the homotopic path is important for interleaved execution of the plan. Thus repeated extensions, while they may speed planning, may come at the expense of average plan length. ERRT, by using a tunable constant to set the maximum number of extensions, allows the system designer to trade off between the two. In most applications, we have used a value of 4, and found it to represent a reasonable compromise.

ERRT also adopts the notion of the connect operation from RRT-Connect, however this can also cause plan length to suffer. ERRT again offers a way to mitigate this with a tunable parameter, which is the number of connections between the initial and goal trees before planning is terminated. Thus, ERRT allows multiple connection points, and can choose the shortest path of those available when planning terminates. Implementing such a feature represents an important departure from RRT however; with multiple connections the connected planning

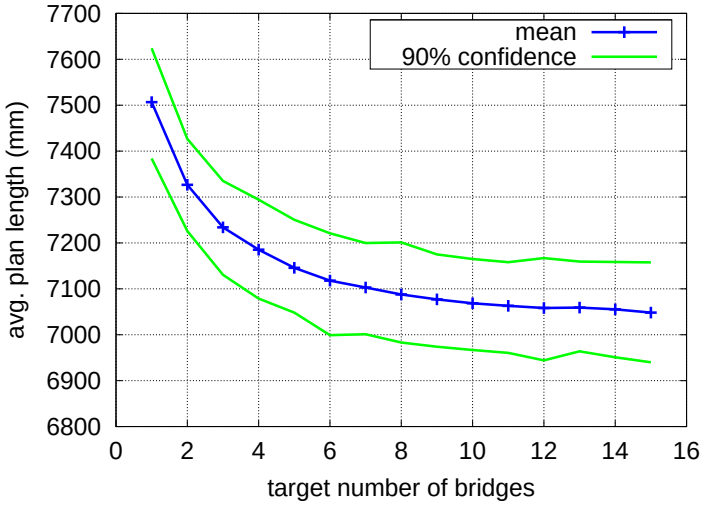

Fig. 16. The effect of multiple connection points in ERRT on plan length.

"tree" is actually now a graph. This does not pose any significant theoretical problems to RRT, but requires the implementation to support graph operations efficiently instead of the normally faster tree operations. When planning terminates, A* search is used over the resulting graph. The effect of varying the number of connection points is shown in Figure 16. The methodology is the same as used for the maximum repeated extensions, but using the domain *RandCircle*. As can be seen, increasing the number of connections improves the plan length, although the effect decreases after 6-8 connection points. Multiple connection points by definition increase planning execution time, since ERRT continues to search even after a path has been found. However, after the first connection, ERRT can operate in a purely any-time fashion, using up idle time in the control cycle to improve the solution, but able to terminate and return a valid solution whenever it is needed. Again, by offering the number of connections as a parameter, the tradeoff can be set by the system designer.

Supporting multiple extensions and multiple connection points give ERRT the benefits of RRT-Connect, as well as the ability to tune the system as needed. By supporting a graph representation instead of search trees, as well as planning over the resulting roadmap graph, ERRT takes a step toward unification with the PRM family of planners. The only differences that remain are in the sampling strategies for building and maintaining the search graph or roadmap structure.

3.8 GUI

Using the Skills, Tactics, and Plays infrastructure, it is possible to model complex multi-robot behaviors. However, such complexity can also make it challenging to

analyze and debug the many components and parameters of the system. To overcome such challenges, CMDragons features a logging infrastructure that is able to record complete sequences of world states during games. Besides containing the world state (which consists of the ball's and all the robots' state parameters), the system also records the internal state of the Play-selection system and the current hierarchy of selected Tactics. Additionally, each Tactic is able to write further debugging information into the log, thus providing detailed information about the Tactics' internal reasoning at any given point in time.

To make use of such rich data, CMDragons features a *Viewer* program that allows easy browsing through logs. Figure 17 shows a screenshot of this viewer. The internal state of the Play system and Tactics is shown as a browsable hierarchy of text (on the right). The world state is drawn onto the virtual soccer field, and allows toggling of several visualization options. Furthermore, the viewer can display the current velocity of the robots or balls as a graph (bottom).

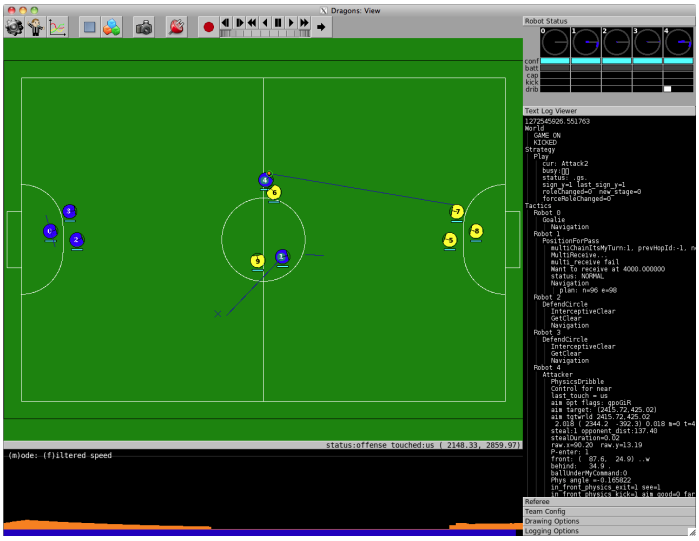

Fig. 17. A screenshot of the CMDragons Viewer.

4 Conclusion

This paper gave an 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

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

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
US Open 2009	1st
RoboCup 2009	Quarter Final

Table 2. Results of RoboCup small-size competitions for CMDragons from 2003-2009

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 eight teams of the tournament every year since 2003, and finishing 1st in 2006 and 2007. The competition results since 2003 are listed in Table 2. 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. Zickler, S., Vail, D., Levi, G., Wasserman, P., Bruce, J., Licitra, M., Veloso, M.: CMDragons 2008 Team Description. In: Proceedings of RoboCup 2008
2. 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)
3. Bruce, J., Balch, T., Veloso, M.: Fast color image segmentation for interactive robots. In: Proceedings of the IEEE Conference on Intelligent Robots and Systems, Japan (2000)
4. Bruce, J.: CMVision realtime color vision system. The CORAL Group's Color Machine Vision Project `http://www.cs.cmu.edu/~jbruce/cmvision/`.
5. Bruce, J., Veloso, M.: Fast and accurate vision-based pattern detection and identification. In: Proceedings of the IEEE International Conference on Robotics and Automation, Taiwan (May 2003)
6. 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)
7. 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

8. 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)
9. Gerkey, B.P., Mataric, M.J.: A formal analysis and taxonomy of task allocation in multi-robot systems. International Journal of Robotics Research **23**(9) (2004) 939–954
10. Uchibe, E., Kato, T., Asada, M., Hosoda, K.: Dynamic task assignment in a multiagent/multitask environment based on module conflict resolution. In: Proceedings of the IEEE International Conference on Robotics and Automation. (2001) 3987–3992
11. D'Angelo, A., Menegatti, E., Pagello, E.: How a cooperative behavior can emerge from a robot team. In: Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems. (2004)
12. Tilove, R.B.: Local obstacle avoidance for mobile robots based on the method of artificial potentials. General Motors Research Laboratories, Research Publication GMR-6650 (September 1989)
13. Koren, Y., Borenstein, J.: Potential field methods and their inherent limitations for mobile robot navigation. In: Proceedings of the IEEE International Conference on Robotics and Automation. (April 1991) 1398–1404
14. Veloso, M., Bowling, M., Achim, S., Han, K., Stone, P.: The CMUnited-98 champion small robot team. In: RoboCup-98: Robot Soccer World Cup II, Springer Verlag (1999)
15. Baraff, D.: Physically Based Modeling: Rigid Body Simulation. SIGGRAPH Course Notes, ACM SIGGRAPH (2001)
16. Donald, B., Xavier, P., Canny, J., Reif, J.: Kinodynamic motion planning. Journal of the ACM (JACM) **40**(5) (1993) 1048–1066
17. 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
18. Zickler, S., Veloso, M.: Tactics-Based Behavioural Planning for Goal-Driven Rigid-Body Control. Computer Graphics Forum **28**(8) (2009) 2302–2314
19. Zickler, S.: Physics-Based Robot Motion Planning in Dynamic Multi-Body Environments. PhD thesis, Carnegie Mellon University, Thesis Number: CMU-CS-10-115 (May 2010)
20. Bruce, J.R.: Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments. PhD thesis, Carnegie Mellon University (Dec 2006)
21. 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)