CMDragons 2007 Team Description

James Bruce **Stefan Zickler** **Mike Licitra**
Manuela Veloso

December 2007
CMU-CS-07-173

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We provide an overview of the robotic soccer team CMDragons 2007, as well as some technical highlights represented in this year's system. CMDragons 2007 is the Carnegie Mellon entry for the RoboCup Small-Size League, and builds on our competitive teams fielded in the last several years. The overview describes both the robot hardware and the general software architecture of our team. Significant technical improvements include one-touch shot aiming, and a new navigational planner.

Keywords: RoboCup, robotic soccer, planning, TDP

1 Introduction

Our RoboCup Small-Size League entry, CMDragons 2007, builds on the world champion team CMDragons 2006, as well as the ongoing research used to create the CMDragons team (1997-2003,2006) 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 linked to frame-grabbers on the offboard computer. The software then sends driving commands to the individual robots. The first section describes the robot hardware and the offboard control software required to implement a robot soccer team. The second section highlights some advances represented in this year's team. The paper then finishes with concluding remarks.

2 System Overview

Our team consists of seven homogeneous robot agents, with five being used in a game at any one time. In Figure 2, an example robot is shown with and without a protective plastic cover. For this year, we do not expect to make significant changes to the hardware, and intend to focus on improving the software to make full use of the existing robots. Nevertheless, there was considerable interest in our hardware by other teams at RoboCup 2006, so we describe it here.

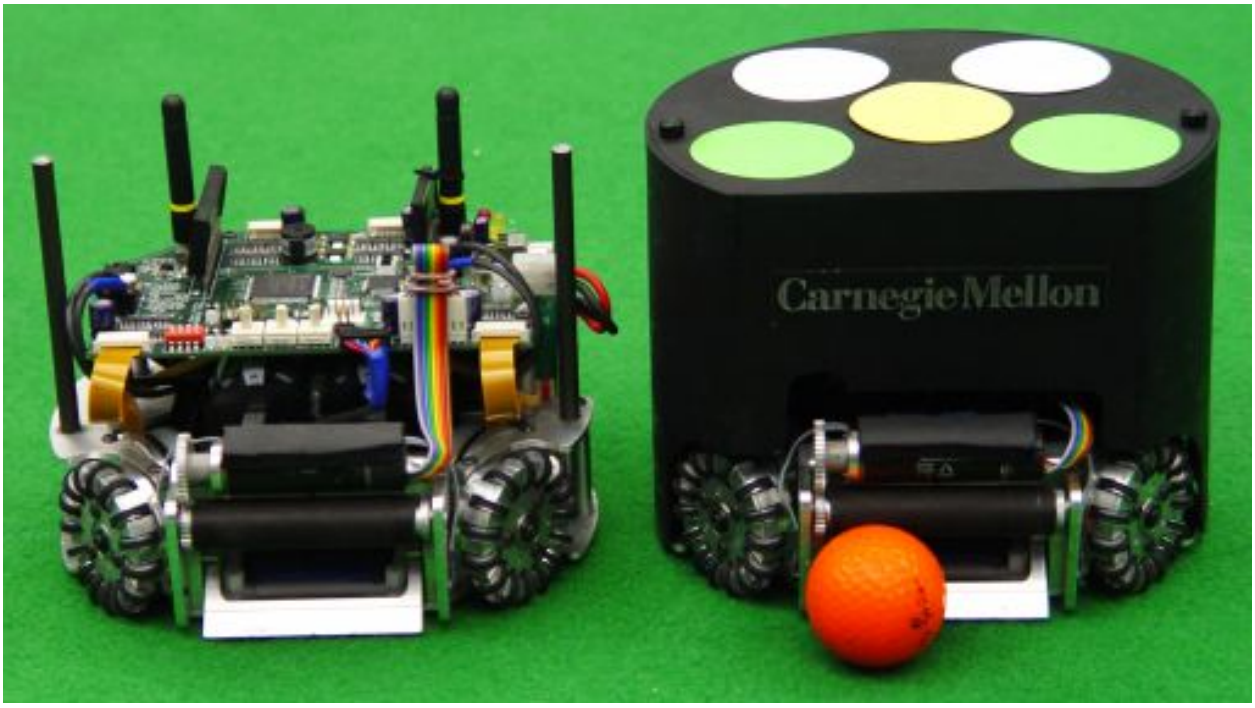

Figure 1: A CMDragons robot shown with and without protective cover.

2.1 Robot Hardware

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.

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.

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. The main electronics board 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.

2.2 Software

The software architecture for our offboard control system is shown in Figure 3. The major organizational components of the system are a server program which performs vision and manages

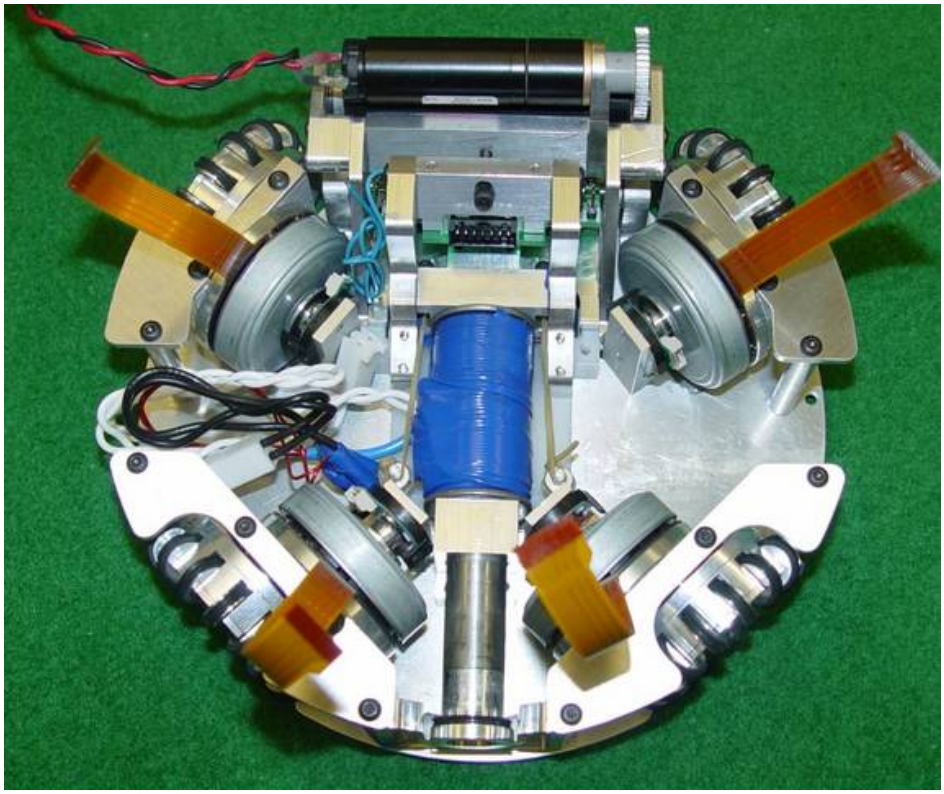

Figure 2: Top view of the robot drive system, kicker, and dribbler.

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 server program consists of vision, tracker, radio, and a multi-client server. The vision system uses CMVision2 for low-level image segmentation and connected region analysis [5, 4]. On top of this system lies a high-level vision system for detecting the ball and robot patterns. Our robot pattern detector uses an efficient and accurate algorithm for multi-dot patterns described in [6]. 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 [3]. An additional tracking system is developed for tracking chip-shots in real-time, based on the model given in [13]. The system tracks the linear constraints on the ball from the observed camera positions, and saves the data for the past second (60 frames). It then attempts to fit a 3D parabolic trajectory to the ball, and returns the trajectory if the RMS error is below a pre-set threshold. Final commands are communicated by the server program using a radio component. This subsystem sends short commands to each robot over an RS232 radio link. The system allows multiple priority levels so that different clients may control the same robots with appropriate overriding. This allows, for example, a joystick tele-operation program to override one of the soccer agents temporarily while

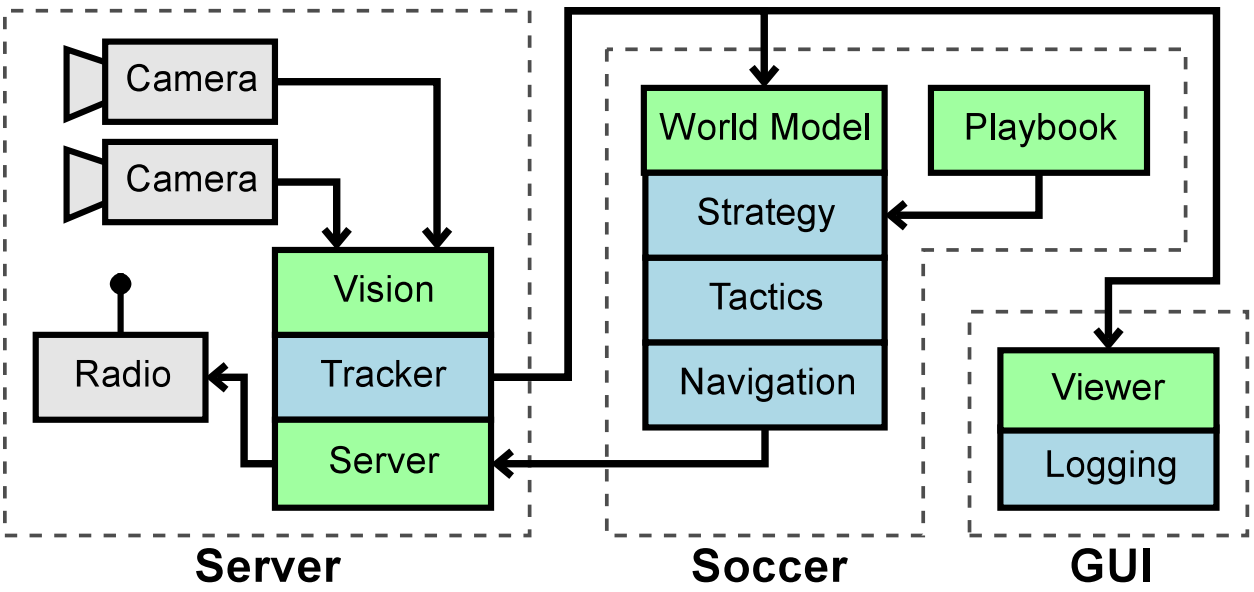

Figure 3: The general architecture of the CMDragons offboard control software.

the soccer system is running.

The soccer program is based on the STP framework [3]. 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 [1, 8]. 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 [3]. One primitive skill used by almost all behaviors is the navigation module, which uses the RRT-based ERRT randomized path planner [9, 14, 12] combined with a dynamics-aware safety method to ensure safe navigation when desired [10]. It is an extension of the Dynamic Window method [11, 2]. The robot motion control uses trapezoidal velocity profiles (bang-bang acceleration) as described in [7, 3].

3 Significant Developments

Numerous low level skills are required to implement a robot soccer team. These play a critical role in the small-size league due to both the speed of the game and the robustness required for a five robot team to operate successfully. Two areas of our team that represent improvements over our past entries are one-touch aiming, and our improved navigational planner.

3.1 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 4. 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:

$$\hat{v_1} = \beta(R_p \cdot v_0)R_p + \gamma(v_0 - 2(v_0 \cdot R_h) \cdot R_h) + kR_h \tag{1}$$

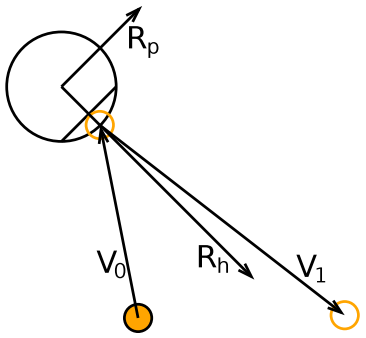

Figure 4: The model for one-touch pass-and-shoot. The final velocity of the ball v_1 contains a component of the initial velocity v_0, and thus is not parallel to the robot heading R_h.

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, however these values are likely to

be dependent on our exact robot design, and can even be affected by the field surface. As the calibration procedure is straightforward, 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 1 as a function of the robot angle α.

$$
\begin{aligned}
R_h(\alpha) &= \langle \cos\alpha, \sin\alpha \rangle \\
R_p(\alpha) &= \langle -\sin\alpha, \cos\alpha \rangle \\
\hat{v_1}(\alpha) &= \beta(R_p(\alpha) \cdot v_0)R_p(\alpha) + kR_h(\alpha) + \\
&\quad \gamma(v_0 - 2(v_0 \cdot R_h(\alpha)) \cdot R_h(\alpha)) \qquad (2) \\
e(\alpha) &= \hat{v_1}(\alpha) \cdot g \qquad (3)
\end{aligned}
$$

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 Bisection search is $O(\log(1/\epsilon))$, and 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. An example of a successful header is shown in Figure 5. 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.

3.2 Navigational 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

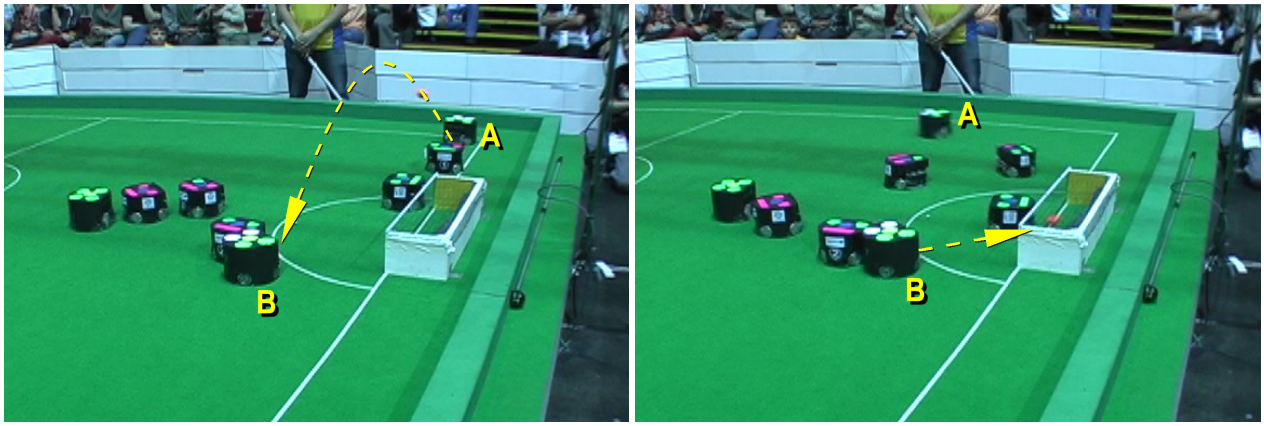

Figure 5: An example header against Plasma-Z in the RoboCup 2006 semi-final match. Robot A passes to robot B, which then deflects the ball into the goal. Robot B chooses its angle using one-touch aiming in order to deflect into the gap left by the goalkeeper.

called *Bidirectional Multi-Bridge ERRT* [7]. 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 [12]. 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

```
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.

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 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 6 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.

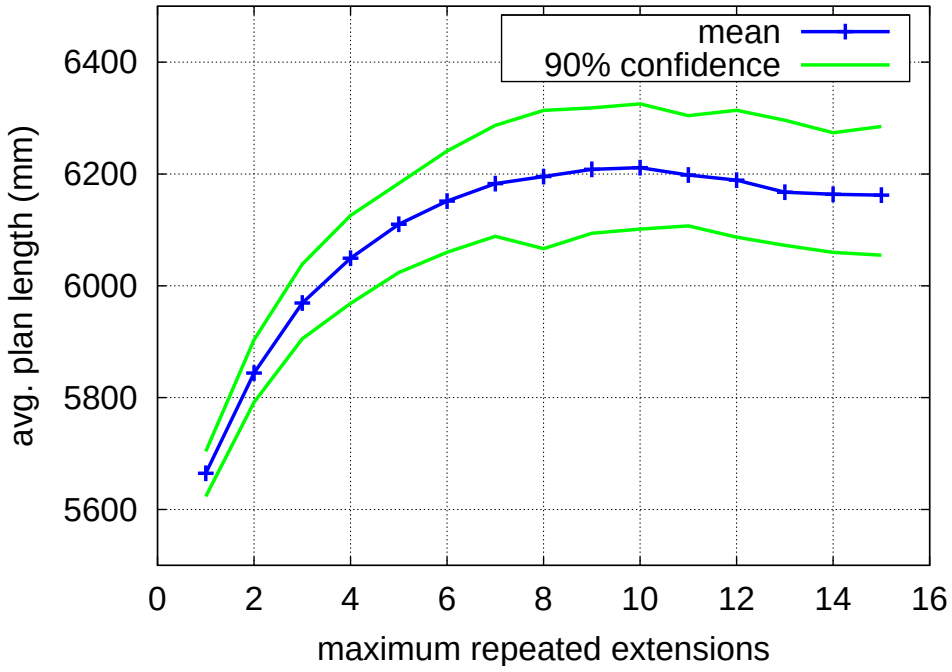

Figure 6: The effect of repeated extensions in ERRT on plan length.

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

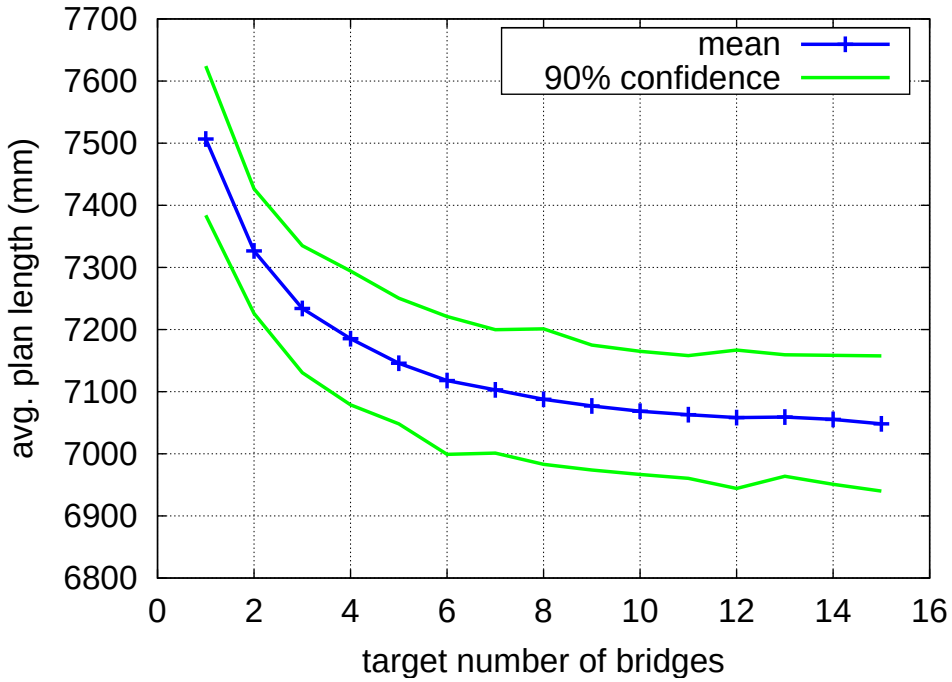

Figure 7: The effect of multiple connection points in ERRT on plan length.

available when planning terminates. Implementing such a feature represents an important departure from RRT however; with multiple connections the connected planning "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 7. 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. This planner represents ongoing research by the authors.

4 Conclusion

This paper gave a brief overview of CMDragons 2007, 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 six international RoboCup competitions, placing within the top four teams of the tournament every year since 2003, and finishing 1st in 2006. The competition results since 2003 are listed in Table 4. 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.

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

Table 2: Results of RoboCup small-size competitions for CMDragons from 2003-06

References

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

[2] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999.

[3] Brett Browning, James R. Bruce, Michael Bowling, and Manuela Veloso. STP: Skills tactics and plans for multi-robot control in adversarial environments. In *Journal of System and Control Engineering*, 2005.

[4] James Bruce. CMVision realtime color vision system. The CORAL Group's Color Machine Vision Project. `http://www.cs.cmu.edu/~jbruce/cmvision/`.

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

[5] James Bruce, Tucker Balch, and Manuela Veloso. Fast color image segmentation for interactive robots. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems*, Japan, 2000.

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

[7] James R Bruce. *Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments*. PhD thesis, Carnegie Mellon University, Dec 2006.

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

[9] James R. Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems*, 2002.

[10] James R. Bruce and Manuela Veloso. Safe multi-robot navigation within dynamics constraints. *Proceedings of the IEEE*, 94:1398–1411, July 2006.

[11] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4, March 1997.

[12] Jr. James J. Kuffner and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[13] Taeone Kim, Yongduek Seo, and Ki-Sang Hong. Physics-based 3d position analysis of a soccer ball from monocular image sequences. *Sixth International Conference on Computer Vision*, pages 721–726, 1998.

[14] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. In *International Journal of Robotics Research, Vol. 20, No. 5*, pages 378–400, May 2001.