Appendix B

The CMDragons Multi-Robot System

For the past eight years, we have been pursuing this domain as a research problem of teams of autonomous robots acting in adversarial, dynamic environments. Our RoboCup Small-Size League entry, CMDragons 2006 builds on the successful past entries of CMDragons and the joint team CMRoboDragons, which entered in RoboCup 2004 and 2005 and placed fourth overall both years. It also builds on the prior experience of CMDragons entries in 1998-99 and 2001-03 [16, 22, 70]. 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. Each robot has four drive wheels, a kicker, and a dribbler. The robot can also send status data back to the offboard control computer to augment the overhead vision information.

Motivated by this challenging multi-robot scenario, we have focused on investigating general multi-robot path planning. Effective multi-robot path planning is a very complex problem especially in highly dynamic environments, such as our robot soccer task. Robots need to rapidly navigate avoiding each other and obstacles while aiming at reaching specific objectives. We decompose the multi-robot path planning problem into two parts: (i) the definition of target points that each robot needs to achieve, and (ii) the multi-robot planning and safe navigation towards the assigned target points. This article briefly addresses the setting of objectives, and focuses in detail on safe multi-robot navigation. Although the systems described here are motivated by the specific requirements of the RoboCup domain, by looking at the common elements across many versions we realize these parts are of wider applicability in other single-robot or multi-robot domains, especially where high speed and safety are desired.

Throughout our design and algorithms, we assume a system where the robotic agents are distributed, but decisions are made in a centralized fashion. In particular, all agents share a common world state with only pure Gaussian noise in state estimates, and we assume perfect communication of actions between all agents. Though centralized, the system is still distinct from a completely monolithic design where all agents share a single world state vector and joint action space. First, the execution time of our approach scales linearly with the number of agents (rather than the exponential scaling typical of a monolithic design). Second, the state required to be communicated between agents is of fixed bounded size, and is not directly related to the complexity of per-agent local calculations. Thus, although centralized, our approach is closer to meeting the restrictions of a distributed algorithm, and we expect it would serve as a good starting point for such a future development.

Offboard communication and computation is allowed, leading nearly every team to use a centralized approach for most of the robot control. The data flow in our system, typical of most teams, is shown in Figure B.1 [16, 22]. Sensing is provided by two or more overhead cameras, feeding into a central computer to process the image and locate the 10 robots and the ball on the field 30-60 times per second. These locations are fed into an extended Kalman filter for tracking and velocity estimation, and then sent to a "soccer" module which implements the team strategy using various techniques. The three major parts of the soccer system are: (1) world state evaluation, (2) tactics and skills, and (3) navigation. World state evaluation involves determining high level states about the world, such as whether the team is on offense or defense. It also involves finding and ranking possible subgoals, such as evaluating a good location to receive a pass. 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. Given these parameters, along with the world state, the tactic generates a navigation target either directly or by invoking lower level skills with specific parameters. Finally, these navigation targets are passed to a navigation module, which employs randomized path planning, motion control, and dynamic obstacle avoidance. The resulting velocity commands are sent to the robots via a serial radio link. Due to its competitive nature, over the years teams have pushed the limits of robotic technology, with the some small robots travelling over $2m/s$, with accelerations between $3-6m/s^2$, and kicking the ball used in the game at up to $10m/s$. Their speeds require every module to run in real-time to minimize latency, all while leaving enough computing resources for all the other modules to operate. In addition, all the included algorithms involved must operate robustly due to the full autonomy requirement.

Two parts of our soccer system which have proven stable in design over the several iterations of our team are the world state evaluation for position determination and navigation module for real-time motion planning. First, determining supporting roles through objective

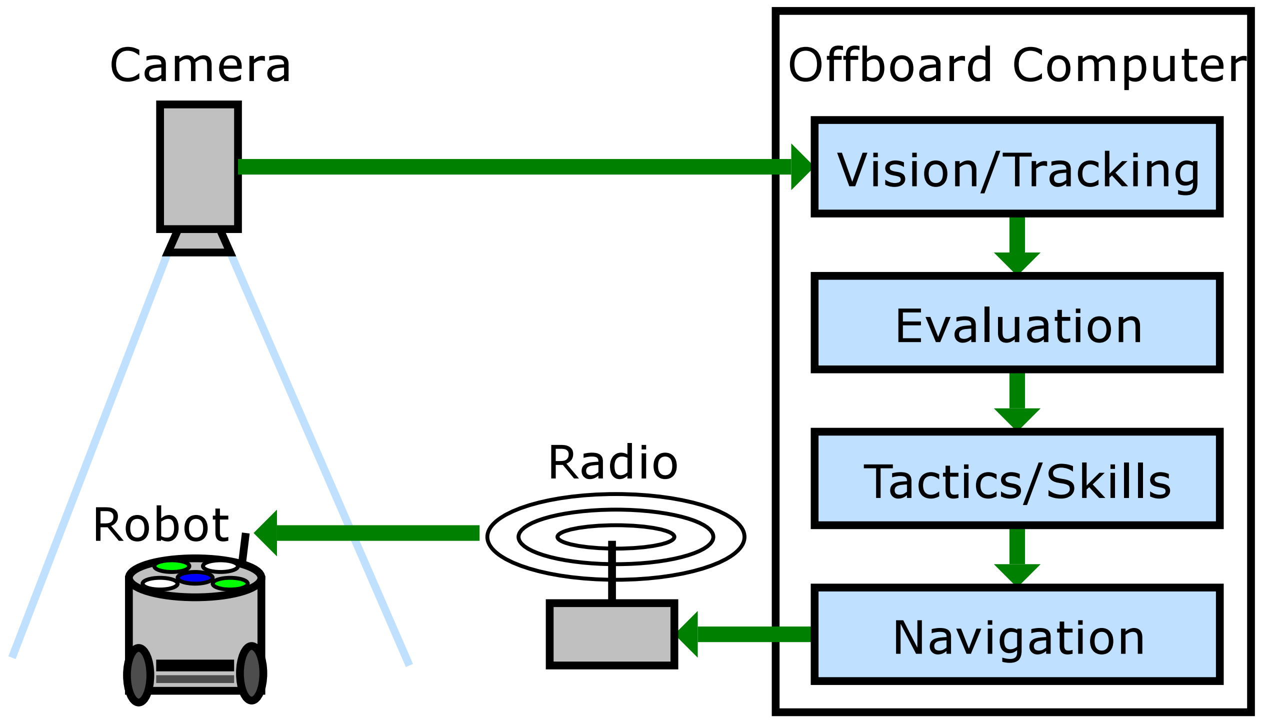


Figure B.1: The overall system architecture for CMUnited/CMDragons. Thanks in particular to Brett Browning and Michael Bowling for their contributions to the overall team.

functions and constraints has proven a very natural and flexible way of allowing multiple robots to support an active player on offense, and to implement defensive strategies on defense. This paper will describe the common elements of this module which have been present throughout its evolution. Second, navigation has always been a critical component in every version of the system. Parts of the system described here were first present in 2002, with the latest dynamic safety module debuting in 2005. The navigation system's design is built on experience gained since 1997 working on fast navigation for small high performance robots [11,86].

In the RoboCup domain, we found the navigation system to work well in practice, helping our team consistently place within the top four teams, with comparatively few penalties for aggressive play. In testing, the planner in the RoboCup achieved execution times below 1ms to meet the tight timing requirement of the small-size system (in practice the planner is aborted early if too much time is used, with the associated degradation in navigation performance).

B.1 Software Overview

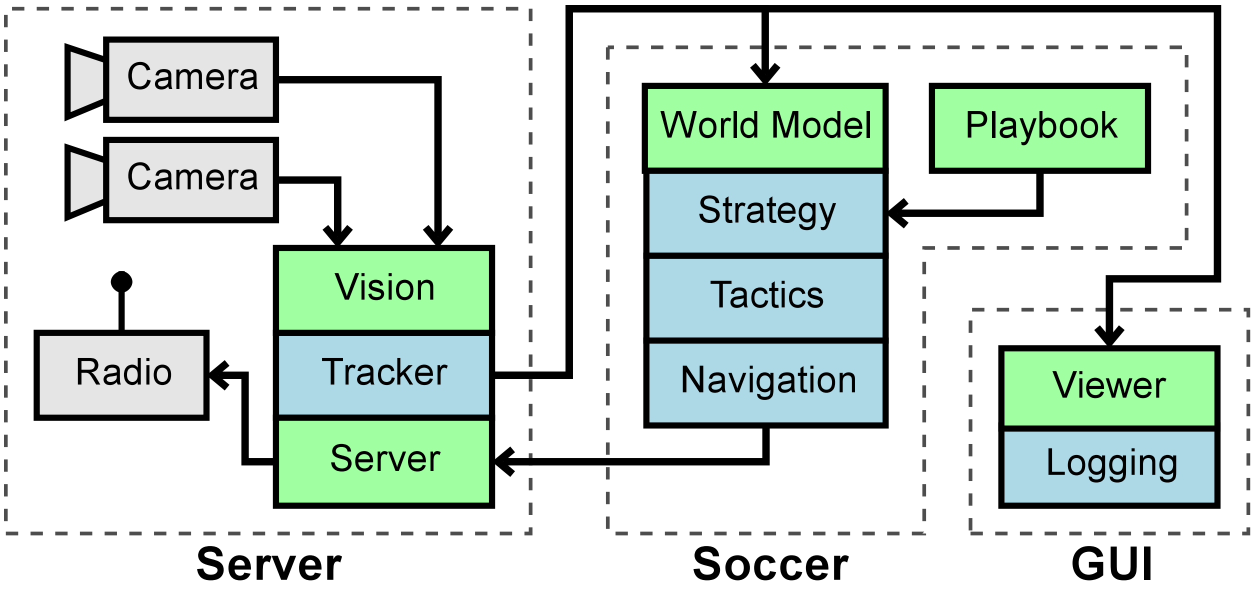


Figure B.2: The general architecture of the CMDragons offboard control software.

The software architecture for our offboard control system is shown in Figure B.2. The major organizational components of the system are a server program which performs vision and manages communication with the robots. Two other major client programs connect to the server via UDP sockets. The first is a soccer program, which implements the soccer playing strategy and robot navigation and control, and the second 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 [17, 18]. 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 [20]. 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 [16]. The radio system sends short commands to each robot over a 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 the soccer system is running.

The soccer program is based on the STP framework [16]. 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 [10, 22]. 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 [16]. One primitive skill used by almost all behaviors is the navigation module, which uses the RRT-based ERRT randomized path planner [23, 51, 65] combined with a dynamics-aware safety method to ensure safe navigation when desired [24]. It is an extension of the Dynamic Window method [12, 38]. The robot motion control uses trapezoidal velocity profiles (bang-bang acceleration) as described in [16, 22].

B.2 Robot Hardware

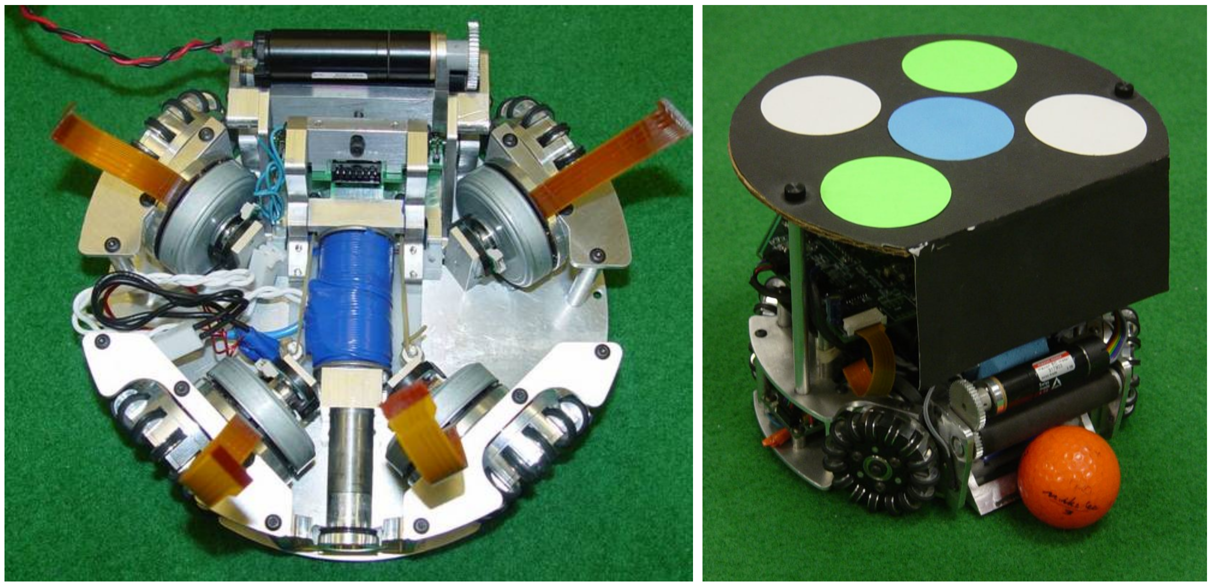


Figure B.3: View of the robot drive system, kicker, and dribbler (left), and an assembled robot without a protective cover (right).

Our team consists of five homogeneous robot agents. 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 the previous design which used a standard D-frame solenoid pushing a kicking plate with guide rods. Ball catching and handling is performed by an actuated, rubber-coated dribbling bar which is mounted on a hinged damper for improved pass reception. The robot drive system and kicker are shown in Figure B.3. Our 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.

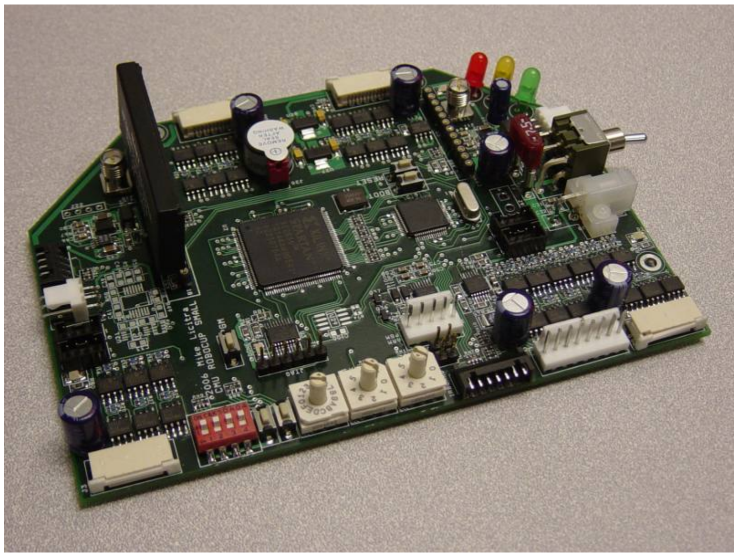


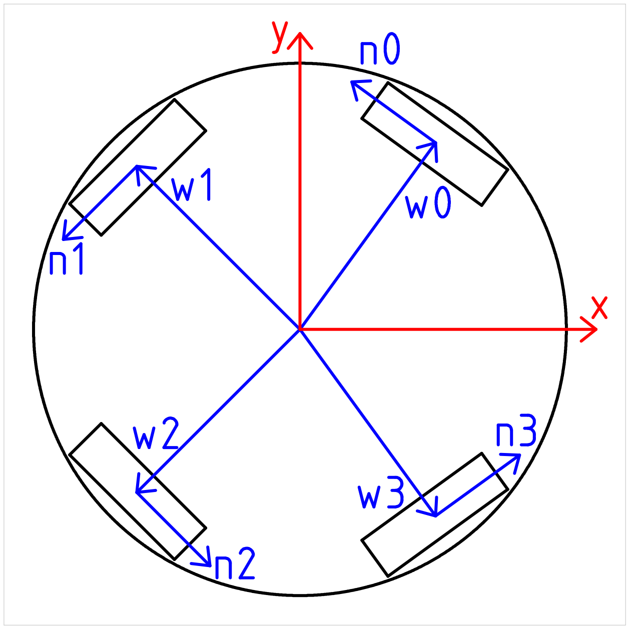
Figure B.4: The main robot electronics board for CMDragons 2006.

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 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, which integrates all electronic components except for a separate kicker board and IR ball sensors, is shown in Figure B.4. 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 small size, 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.

For improved angular control, the robot also incorporates an angular rate gyroscope. It is linked into the drive control system via a secondary proportional controller, and can be used to move to or maintain a specific heading. Whenever the robot is visible to the overhead camera, the coordinate system is updated periodically using the angle determined by the vision system. This allows the local heading on the robot to match the heading for the global coordinate system. Using the gyro and secondary control loop, the robot is able to maintain a stable heading even during periods of no vision lasting several seconds.

B.3 Robot Model

Thus we can assume wheel centers at vectors $w_i, i \in [0, 3]$ relative to the center of the robot, each pointed perpendicular to the w_i vector along unit vectors n_i.



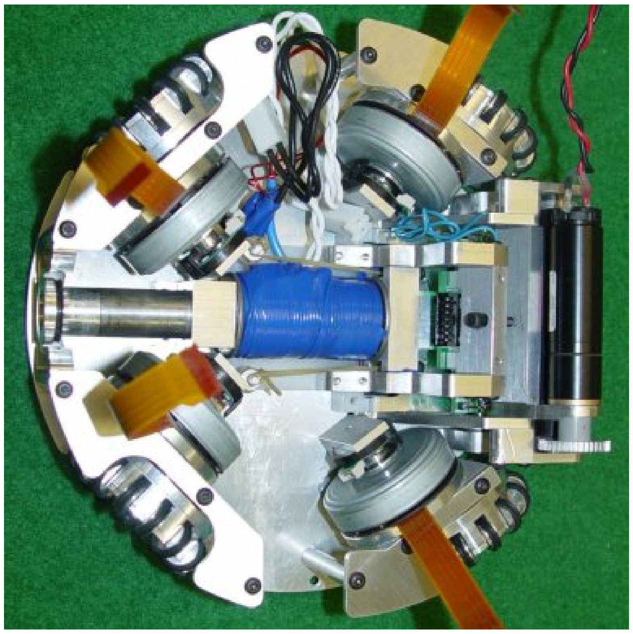


Figure B.5: Model of a CMDragons holonomic robot with four omni-directional wheels (left), and a picture of a partially assembled robot (right).

If each wheel of the robot is driven by an electric motor, we can assume the time to generate a particular force on a wheel is negligibly small, and as a simplified approximation we can treat the wheels as points which can exert force along their respective n_i vectors, up to some bounded maximum velocity. This is shown graphically in Figure B.5 The force an electric motor (and thus the wheel) can exert decreases linearly from a maximum when the wheel is stationary down to a force of zero when the wheel is rotating at the maximum no-load speed.

This leads to the following dynamics equations, where F is the total force on the robot and τ is the torque:

$$\begin{aligned} F(t) &= \sum_{i=0..3} n_i f_i(t) && \text{(B.1)} \\ \tau(t) &= \sum_{i=0..3} w_i \times n_i f_i(t) && \text{(B.2)} \\ \ddot{q}(t) &= mF(t) && \text{(B.3)} \\ \dot{\omega}(t) &= I\tau(t) && \text{(B.4)} \end{aligned}$$

For a relatively high performance robot however, the motor torque often exceeds the friction the wheel can exert on the ground, resulting in a traction-limited maximum force. Although again reality is quite a bit more complex, we will assume a fixed maximum no-slip traction force. These traction problems lead to controller stability issues with a full multiple-input multiple-output (MIMO) controller, as shown for a similar RoboCup robot in Sherback et al. [77]. As in [77], we adopt the simpler single-input single-output (SISO) controller model, with a velocity PD loop at each wheel. Unfortunately this leads to some torque imbalance during acceleration, as Equation B.2 is not guaranteed to be equal to zero. However, with a sufficiently stiff controller, the imbalance is within tolerable error.

To model the robot, we decouple translation and rotation, reserving a small amount of the total control effort for angle, and leaving the rest for translation. The robot has a small moment of inertia, and little need to turn at maximum speed, so this tradeoff is acceptable. To model the maximum traction force, we adopt a translational acceleration limit as in [52] and [77]. We add an additional ability to decelerate at a constant larger than the acceleration, leading to a bounding half-ellipse shape (See Section 5.1, and in particular Figure 5.2).

B.4 Motion Control

Once the planner determines a waypoint for the robot to drive to in order to move toward the goal, it feeds this target to the motion control layer. The motion control system is responsible for commanding the robot to reach the target waypoint from its current state, while subject to the physical constraints of the robot as described above. The model we will take for our robot is a three or four wheeled omnidirectional robot, with bounded acceleration and a maximum velocity. The acceleration is bounded by a constant on two independent

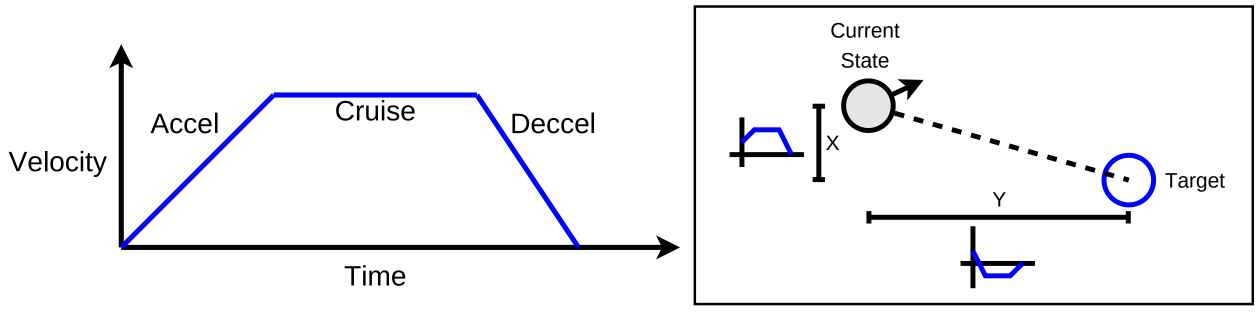


Figure B.6: Our motion control approach uses trapezoidal velocity profiles. For the 2D case, the problem can decomposed into two 1D problems, with the division of control effort balanced using binary search.

axes, which models a four-wheeled omnidirectional robot well. In addition, deceleration is a separate constant from acceleration, since braking can often be done more quickly than increasing speed. The approach taken for motion control is the well known trapezoidal velocity profile. In other words, to move along a dimension, the velocity is increased at maximum acceleration until the robot reaches its maximum speed, and then it decelerates at the maximum allowed value to stop at the final destination. An example velocity profile is shown in Figure B.6. The area traced out by the trapezoid is the displacement effected by the robot. For motion in 2D, the problem is decomposed as two 1D motion problems along x and y. For each direction, we allow some multiple of the total control effort, based on a position on the boundary of the acceleration half-ellipse (see the previous section for a description). The relative weights are altered to minimize the maximum execution time of the two directions. This balancing is achieved using binary search, always increasing the weight of the direction requiring more time to execute [77].

While the technique of trapezoidal control is well known, the implementation focuses on robustness even in the presence of numerical inaccuracies, changing velocity or acceleration constraints, and the inability to send more than one velocity command per cycle. First, for stability in the 2D case, if the initial and target points are very close, the coordinate frame can become degenerate. In that case the last coordinate frame above the distance threshold is used. For the 1D case, the entire velocity profile is constructed before calculating the command, so the behavior over the entire command period (1/60 to 1/30 of a second) can be represented. The calculation proceeds in the following stages:

- If the current velocity has a different sign than the difference in positions, decelerate to a complete stop.
- Alternatively, if the current velocity will overshoot the target, decelerate to a

complete stop.

- If the current velocity exceeds the maximum, decelerate to the maximum.
- Calculate a triangular velocity profile that will close the gap.
- If the peak of the triangular profile exceeds the maximum speed, calculate a trapezoidal velocity profile.

Although these rules construct a velocity profile that will reach the target point if nothing impedes the robot, limited bandwidth to the robot servo loop necessitates turning the full profile into a single command for each cycle. The most stable version of generating a command was to simply select the velocity in the profile after one command period has elapsed. Using this method prevents overshooting, but does mean that very small short motions will not actually occur (when the entire profile is shorter than a command period). In these cases it may be desirable to switch to a position based servo loop rather than a velocity base servo loop if accurate tracking is desired.

B.5 Objective Assignment for Multi-Robot Planning

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. Occasionally, even tighter coordination is needed with multiple robots executing joint actions in concert, such as for a passing play between robots, or joint manipulation.

Robotic soccer falls generally between parallel and serialized domains, with brief periods of joint actions. Soccer is a serialized domain mainly due to the presence of a single ball; At any given time only one robot should be actively handling the ball, even though all the teammates need to act. In these domains, multi-robot coordination algorithms 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 give rise to the parallel component of the domain, since each robot can execute different actions in a possibly loosely coupled way to support the overall team objective.

A great body of existing work exists for task assignment and execution in multi-agent systems. Gerkey and Mataric [43] provide an overview and taxonomy of task assignment methods for multi-robot systems. Uchibe [85] 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. A module selection and assignment method with conflict resolution based on priorities was presented. D'Angelo [29] et al. present a cooperation method that handles tight coordination in a soccer domain via messaging between behaviors executing the cooperating agents. Task assignment for our early robot soccer system is given in Veloso et al. [86], while more recent methods are described by Browning et al. [16, 22]. Beyond assignment of tasks to agents, their still lies to problem of describing how each agent should implement its local behavior. Brooks [14] presents a layered architecture using subsumptive rules, while Tivoli [84] presents an artificial potential field for local obstacle avoidance. Arkin [3] presents the method of motor schema, which extends the idea of potential functions to include weighted multiple objectives so that more complex tasks can be carried out. All three of these approaches calculate behaviors based on local sensor views. Koren and Borenstein [58] point out the limitations of direct local execution of potential functions. In Latombe [62], potentials are defined over the entire workspace and used as guidance to a planner, alleviating most of the problems with local minima. The MAPS system [5, 83] described by Tews et al. uses workspace potential functions which are combined to define behaviors in a robotic soccer domain. The potentials are sampled on an evenly spaced grid, and different primitives are combined with weights to define more complex evaluations. The potential is used as input to a grid-based planner and to determine targets for local obstacle avoidance. The method of strategic positioning via attraction and repulsion (SPAR) is presented in Veloso et al. [86], and describes the first method used in our system. It combines binary constraints with linear objective functions to define a potential over the workspace. The system could be solved using linear programming or sampling on a regular grid defined in the workspace. It was successfully applied in the RoboCup small size environment using grid sampling. More recently, Weigel et al. [88] uses a potential approach similar to SPAR but with purely continuous functions defined on a grid. Continuous functions guarantee all locations have a well defined value so that A* [72] search can be directly applied. Laue et al. [63] describe a workspace potential field system with a tree-based continuous planner to calculate a path to the location of maximum potential.

Task allocation in our system is described in Browning et al. [16]. 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 D'Angelo et al. [29]. In this paper we briefly present the potential function maximization used by support roles, and then describe how the resulting navigational target is achieved.

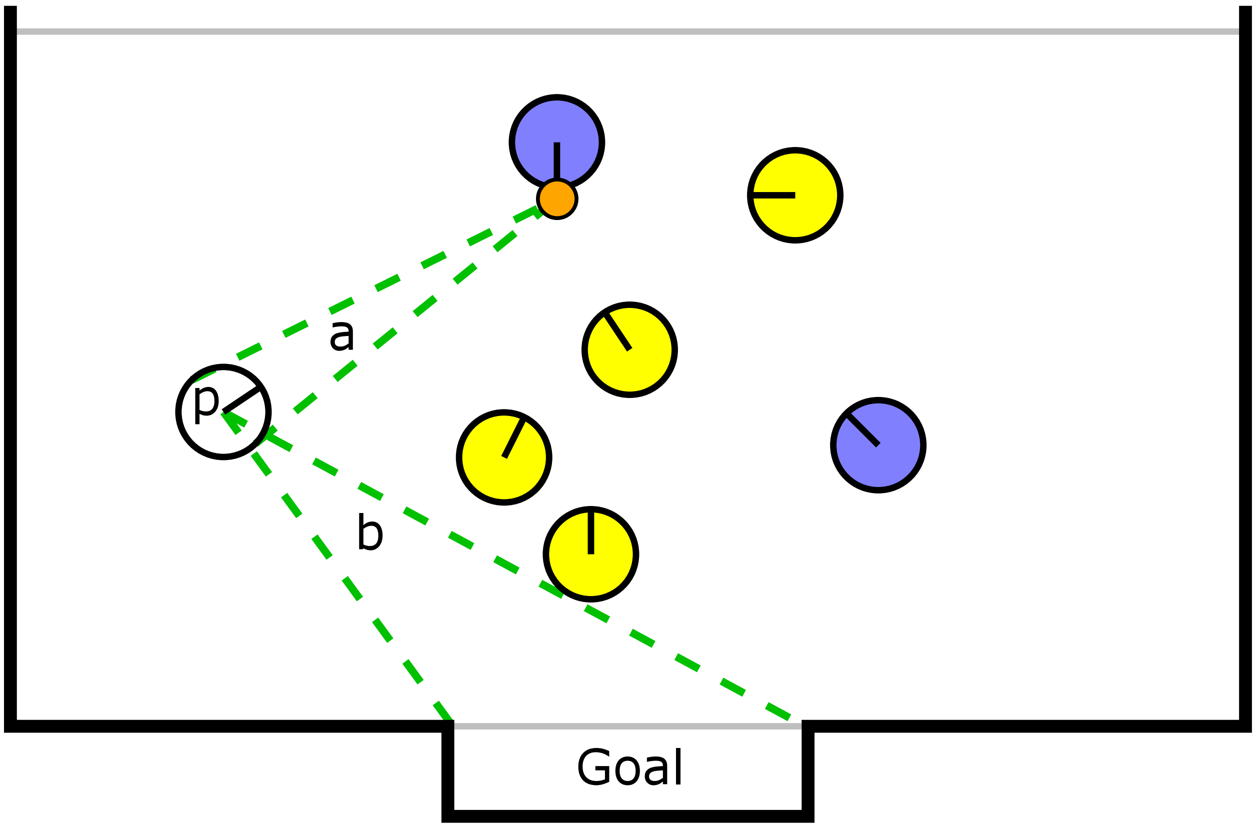


Figure B.7: A example situation demonstrating the passing evaluation function inputs. The input values for evaluating a point p are the circular radius and subtended angle of the arcs a and b, and the angle between the center line segments of the two arcs. These are combined using simple weighted functions to achieve the desired behavior.

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. An example of the general form of the pass position evaluation function is given in Figure B.7. Passing in our system is optimized for a single pass-and-shoot behavior, and the parameters are set to value states where this can execute. First, the angular span of a receiving robot's area at a workspace position p, and the angular span of the open goal area aiming from p are introduced as linear functions with positive weights. Next the relative lengths of the passes are combined to encourage a pass and shot of equal length, maximizing the minimum speed of the ball at any point due to friction. This is introduced using a Gaussian function of the difference in lengths of the pass (a) and shot (b). Finally, interception for a shot is easiest at right angles, so a Gaussian function is applied to the dot product of the pass and shot relative vectors ($a \cdot b$). The weights were set experimentally to achieve the desired behavior. The resulting plots from two example passing situations are shown in Figure B.8.

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.

In order to generate concrete navigation targets for each robot, we must find maximal values of each robot's assigned evaluation function. We would like to do so as efficiently as possible, so that the complexity of potential functions is not a limiting factor. We achieve the necessary efficiency through a combination of hill climbing and randomized sampling. First, a fixed number of samples is evaluated randomly over the domain of the evaluation function, and the sampled point with the best evaluation is recorded. We call this point the *sampled-best*. Two other interesting points are the point of maximum evaluation recorded from the last control cycle, called *previous-best*, and the current location of the robot *current-loc*. For each of these three points, we apply hill climbing to the point with a limited number of steps to find a local maximum. The best of the three is taken as the maximum, and is used as the robots navigation target. It is also recorded for use in the next control cycle as *previous-best*. Using this approach, few random samples need to be evaluated each frame, lending itself to real-time performance. However, because previous maximal values are recorded, the calculations of several control cycles are leveraged to quickly find the global maximum for a sufficiently smooth and slowly changing function. The *current-loc* point is added to provide more consistent output in situations where the maximum of the function changes rapidly. This is normally due to rapid changes in the environment itself.

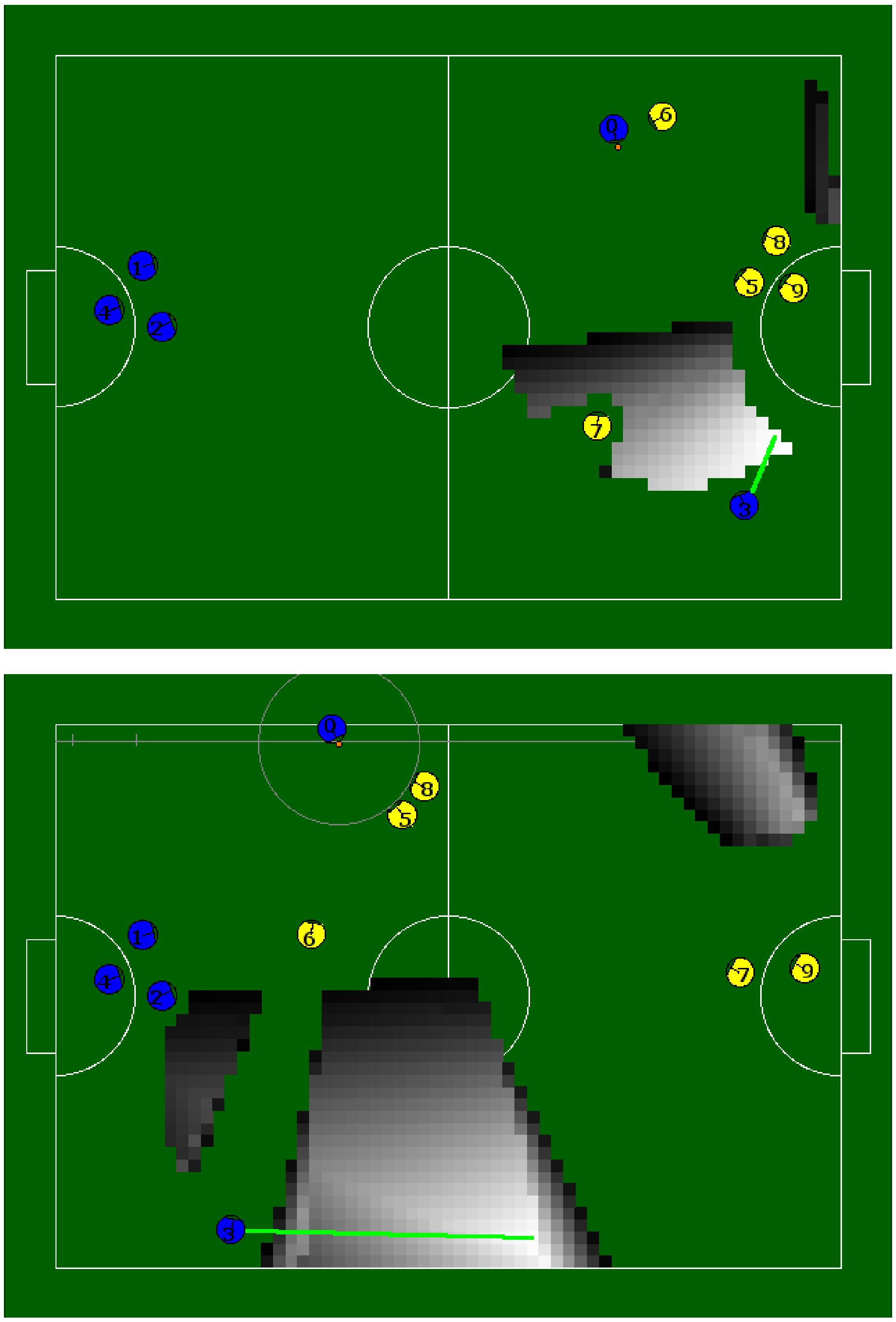


Figure B.8: Two example situations passing situations are shown with, the passing evaluation metric sampled on a regular grid. The values are shown in grayscale where black is zero and the maximum value is white. Values below 0.5% of the maximum are not drawn. The same evaluation function is used in both the short and long pass situations, and the maximum value is indicated by the bold line extending from the supporting robot.

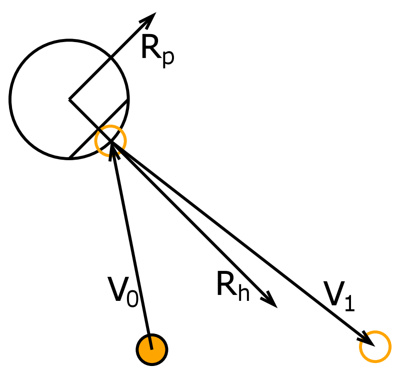


Figure B.9: System model for one-touch ball control. The final velocity of the ball v_1 still contains a component of the initial velocity v_0, rather than running parallel to the robot heading R_h.

B.6 One-touch Ball Control

One significant contribution developed shortly before the 2005 competition was CMRoboDragons' ability to accurately redirect a moving ball. The system we have developed 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 kick an incoming ball in a different direction without explicitly receiving the ball, the most important necessary new component was a model of how interaction with the kicker will affect the speed of the ball. 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.

After measurement and testing of several models, we ended up using a simple linear damping model. The system model is shown in Figure B.9. The initial (incoming) velocity of the ball is denoted as v_0, while the final velocity after kicking is denoted v_1. Normalized heading and perpendicular vectors for the robot are R_h and R_p, respectively. The kicker provides an impulse to the ball in the direction of R_h, propelling a ball initially at rest to speed k (i.e. $\|v_0\| = 0 \rightarrow \|v_1\| = k$). Using this model, we can estimate the final velocity using the following equation:

$$\hat{v_1} = kR_h + \beta(R_p \cdot v_1)R_p \tag{B.5}$$

In our testing, we found values of β ranging from 0.8 with no dribbler present, down to 0.3 for a robot with a dribbler spinning at maximum speed. Of course, simply having a forward model is not sufficient, as the control problem requires solving for the robot angle given some relative target vector g. However, it is easy to calculate this using a 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 where $\beta = 0$. The actual solution lies somewhere in between, and we can calculate an error metric e by setting up the equation above as a function of the robot angle α.

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

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 many models so that search is not required, using a numerical method for determining α allowed rapid testing of different models, since only the forward calculation needed to be made. Bisection search has proven quite fast in practice since the calculations for a forward model are relatively simple, and only a logarithmic number of evaluations need to be made to achieve the desired accuracy.

Overall, the one touch ball control proved quite useful, allowing our team in 2005 year to score most of its goals via passing, even with a relatively slow kicker $3.75m/s$. In 2006, combined with the kicker capable of $15m/s$ and accuracy improvements allowing passes as speeds ranging from $2m/s$ to $4m/s$, it allowed our team to be quite dangerous on offense. We also adapted the 2D version of ball deflection to the 3D problem of soccer “headers”. The chip kicker was 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 allowed us to intercept a ball still in the air to deflect it into a goal. The ground pass deflection and air deflection together greatly contributed to our scoring record and eventual championship.

B.7 Summary and Results

This chapter gives an overview of the CMDragons system, covering both the robot hardware and the overall 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 many improvements to the individual modules. The CMDragons software system has been used at part of two 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 results are listed in Table B.7. The competition and the resulting tournament placing are listed, along with the author's contribution to the total team effort for the software system, based on source code and hours spent. The author's contributions to the robot hardware were generally limited to design input and testing.

Competition	Result	%Contribution
RoboCup 2001	RR	30%
RoboCup 2002	QF	35%
US Open 2003	1st	40%
RoboCup 2003	4th	40%
RoboCup 2004	4th [1]	40%
RoboCup 2005	4th [1]	95%
US Open 2006	1st	85%
RoboCup 2006	1st	85%

Table B.1: Results of RoboCup small-size competitions for CMDragons from 2001-2006

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