INVITED PAPER

Safe Multirobot Navigation Within Dynamics Constraints

In fast robot soccer games, teams play without any human input, avoiding collisions and obstacles and coordinating action to implement team strategy and tactics.

By James R. Bruce and Manuela M. Veloso

ABSTRACT | This paper introduces a refinement of the classical sense–plan–act objective maximization method for setting agent goals, a real-time randomized path planner, a bounded acceleration motion control system, and a randomized velocity-space search for collision avoidance of multiple moving robotic agents. We have found this approach to work well for dynamic and unpredictable domains requiring real-time response and flexible coordination of multiple agents. First, the approach employs randomized search for objective maximization and motion planning, allowing real-time or any-time performance. Next, a novel cooperative safety algorithm is employed which respects agent dynamics limitations while also preventing collisions with static obstacles or other participating agents. An implementation of our multilayer approach has been tested and validated on real robots, forming the basis for an autonomous robotic soccer team.

KEYWORDS | Mobile robots; motion-planning; multirobot; navigation; robot dynamics

I. INTRODUCTION

For the past eight years, we have been pursuing research into teams of autonomous robots acting in adversarial, dynamic environments. We have used robot soccer as the underlying testbed and research platform, and have participated in several leagues in the RoboCup [1] robot soccer competition. In each league, teams of robots play a competitive match modeled after soccer, and during the match the teams must operate fully autonomously without any human input. This work is particularly motivated by our team in the small-size robot soccer league, which offers very interesting multirobot coordination opportunities. Briefly, the team of robots can be observed from one or more external vision cameras, usually hung above the field. This external sensing provides a global view of the state of the complete playing field. Specifically, this includes position of the the ball, as well as the locations of teammate and opponent robots. The global state information is used by offboard computing for effective team coordination and control. Within this domain, several teams have reached extremely impressive levels of performance in multiple aspects of multirobot systems, including real-time vision processing, mechanical robot design and control, and coordination. These accomplishments led to very fast games where robots can move at over 2 m/s and the ball moves at speeds of up to 10 m/s.

Motivated by this challenging multirobot scenario, we have focused on investigating the general navigation problems that it poses. In particular, the offboard sensing removes the world-state estimation bottleneck present on many robotics systems, allowing the navigation problem itself to come to the fore. Effective multirobot path navigation can be 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 to reach specific objectives. At the same time, the speeds reached mean that limitations on agent dynamics cannot be ignored. Finally, the navigation approach has to fit within an overall system, as it must be controllable by a behavior system as its input and provide meaningful commands as output to the local robot control loops in order to carry out the objective goals.

In searching for a solution to the navigation problem posed by this domain, we have looked for general solutions which can be used beyond robot soccer. In particular, we have taken a modular, layered approach, where the interfaces between modules are of relatively wide applicability,

Manuscript received June 1, 2005; revised June 1, 2006. This research was supported in part by United States Grant DABT63-99-1-0013, Grant F30602-98-2-0135, Grant F30602-97-2-0250, and Grant NBCH-1040007; and in part by Rockwell Scientific Co. LLC under Subcontract B4U528968 and Prime Contract W911W6-04-C-0058 with the U.S. Army. The views and conclusions contained herein are those of the authors, and do not necessarily reflect the position or policy of the sponsoring institutions, and no official endorsement should be inferred.

The authors are with the Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213-3891 USA (e-mail: jbruce@cs.cmu.edu; mmv@cs.cmu.edu).

Digital Object Identifier: 10.1109/JPROC.2006.876915

and within each module relatively general models of a mobile agent are assumed. In our approach, the planning aspect of the sense–plan–act loop is expanded into several layers. The first layer is a task-oriented evaluation of the sensed location, defining target points that each robot needs to achieve through objective functions. The next layer is a factored motion planning approach for the multiple agents. This is followed by motion control to define a velocity target for each of the agents. Finally, the last layer employs a cooperative safety algorithm to prevent collisions between the multiple agents, while respecting their dynamics constraints.

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 at worst quadratically with the number of agents (rather than the exponential scaling typical of a monolithic design). Second, the state required to be broadcast by each agent during a control cycle is of fixed size, and is therefore not a function of the the complexity of multiagent interactions or the per-agent local calculations. Thus, although centralized, the approach presented in this paper is closer to meeting the restrictions of a practical distributed algorithm, and we expect it would serve as a good starting point for such a future development.

This paper briefly addresses the setting of objectives and motion planning, and then focuses in detail on our novel multirobot safety method. The presentation is organized as follows. Section II presents the small-size robot soccer league in more detail. Section III discusses our approach for role selection and setting of objectives for the multirobot team. Section IV describes the algorithms for multirobot path planning, followed by motion control in Section V. Section VI then describes how we have achieved safe navigation as a postprocess to planning and motion control. Section VII shows empirical results and Section VIII concludes the paper. Related work is discussed throughout the presentation.

II. SMALL-SIZE MULTIROBOT SOCCER

The RoboCup small-size league involves teams of five small robots, each up to 18 cm in diameter and 15 cm in height. The field of play is a carpet measuring 4.9 m by 3.8 m, with a 30-cm border around the field for positioning outside the field of play (such as for free kicks). A game played on an earlier (half-size) version of the field is pictured in Fig. 1. Since its start in 1997, the league has seen rapid advances in both algorithms and hardware, and has emerged with an emphasis on speed and flexible coordination. Four generations of our small-size robots are shown in Fig. 2.

Fig. 1. *Two teams are shown playing soccer in the RoboCup small-size league.*

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 Fig. 3 [2]. Sensing is provided by two or more overhead cameras, feeding into a central computer to process the image and locate the ten 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,

Fig. 2. *Four generations of Carnegie Mellon robots: (from left) 1997, 1998–99, 2001, 2002–03.*

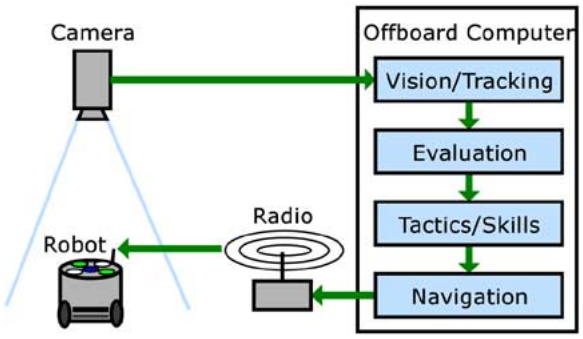

Fig. 3. *The overall system architecture for CMUnited/CMDragons. (Thanks in particular to Brett Browning and Michael Bowling for their contributions to the overall team architecture.)*

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 traveling over 2 m/s, with accelerations between 3 and 6 m/s^2, and kicking the ball used in the game at up to 10 m/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. Thus, these two modules will be the focus of this paper. First, determining supporting roles through objective 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 [3], [4].

III. OBJECTIVE ASSIGNMENT FOR MULTIROBOT PLANNING

Multirobot 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, multirobot 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 work exists for task assignment and execution in multiagent systems. Gerkey and Mataric [5] provide an overview and taxonomy of task assignment methods for multirobot systems. Uchibe [6] 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 *et al.* [7] 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.* [3], while more recent methods are described by Browning *et al.* [2], [8]. Beyond assignment of tasks to agents, their still lies the problem of describing how each agent should implement its local behavior. Brooks [9] presents a layered architecture using subsumptive rules, while Tivoli [10] presents an artificial potential field for local obstacle avoidance. Arkin [11] 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 [12] point out the limitations of direct local execution of potential functions. In Latombe [13], 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 [14], [15] 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.* [3], 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.* [16] 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* [17] search can be directly applied. Laue *et al.* [18] 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.* [8]. Our system 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 signaling between the active and supporting behaviors as in D'Angelo *et al.* [7]. In this paper we briefly present the potential function maximization used by support roles, and then describe how the resulting navigational target is achieved.

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 modeled as general real-valued functions. An example of the general form of the pass position evaluation function is given in Fig. 4. 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

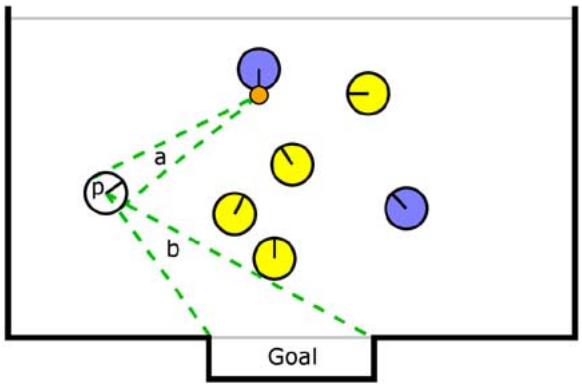

Fig. 4. *An example demonstrating the passing evaluation function inputs. The input values for evaluating a point p are the circular radius and subtended angle of the arcs a and b, and the angle between the center line segments of the two arcs. These are combined using simple weighted functions to achieve the desired behavior.*

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 Fig. 5.

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 may 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 the 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

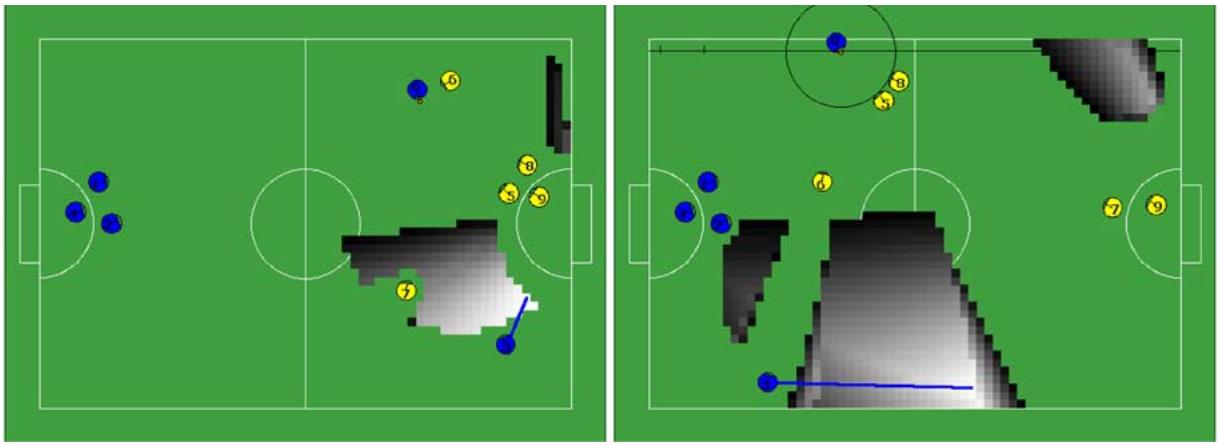

Fig. 5. *Two example situations passing situations are shown with, the passing evaluation metric sampled on a regular grid. The values are shown in grayscale where black is zero and the maximum value is white. Values below 0.5% of the maximum are not drawn. The same evaluation function is used in both the short and long pass situations, and the maximum value is indicated by the bold line extending from the supporting robot.*

maximum. The best of the three is taken as the maximum, and is used as the robot's 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.

IV. PATH PLANNING

Path planning is one of the most studied problems in mobile robotics. Latombe [13] gives a thorough overview of early approaches, while Reif [19] establishes the exponential complexity of the general path planning problem. This complexity has inspired many approximation methods, such as local minima free grid-based potentials [20], and common application of the A* algorithm [17] on cost grid representations of the robot's state space. Stentz's D* algorithm [21] builds on A* to create a variant which only recalculates portions of the problem where costs change, achieving significant speedup for domains where the environment changes slowly over time. Much recent work has centered on the idea of randomized sampling for approximation, such as LaValle and Kuffner's RRT algorithm [22], [23], and planners based on Kavraki *et al.*'s Probabilistic Roadmap (PRM) framework [24], [25]. RRT grows random trees in configuration space to solve single-query problems efficiently, and was extended by Bruce [26] to work efficiently in unpredictably changing domains by using continuous replanning with a bias from past plans. PRM separates planning into a learning and query phase. In the learning phase, a random subgraph of the configuration space is build by sampling points and connections between points to find free locations and paths, respectively. In the query phase, this graph can be used with ordinary graph search methods such as A* to solve the path planning problem. It relies on largely static domains to achieve efficiency, since, in an unchanging workspace, the learning phase need only be computed once. Boor *et al.* [27] changed the sampling method of PRM to focus on the boundaries of free space in their Gaussian PRM approach. Amato and Wu [28], [29] create another modified sampling approach called Obstacle PRM, and Hsu [30] created a bridge test to bias sampling to difficult narrow passages. Isto [31] looked at applying complex local planners to PRM, replacing the commonly used "straight-line" method for local planning to connect two points. It was found to significantly improve graph connectivity for difficult problems, resulting in more reliable planning.

Our particular domain contains multiple moving objects, which must be treated as obstacles for safe navigation. Erdmann and Lozano-Perez [32] look at the effects of multiple moving objects on the planning problem and offer some early solutions by adding time to the configuration space. Latombe [13] provides background and investigates the effects of moving obstacles on the planning problem. Fiorini and Shiller [33] focus on using relative velocity to simplify the planning problem from each agent's point of view, leading to a more tractable problem for mobile robots. Their work was extended to construct explicit velocity obstacles, first for linear paths and then for arbitrary nonlinear paths [34], [35]. The approach assumes that the complete obstacle paths are known in advance. More recently, Hsu [36] applied randomized planning to domains with moving obstacles, and tested the

system on physical robots. This approach assumes constant velocity obstacles, but recovers via replanning when a change in velocity is detected.

In the RoboCup small-size domain, the environment is truly dynamic, with up to ten agents moving at a given time. In addition, it is unpredictable; opponent robots may move arbitrarily, and the motion of the ball cannot be predicted well enough that we can even foresee our own team's commands. The result is that we cannot apply a system which assumes a known future trajectory for moving obstacles, but instead one that allows new decisions to be made each control cycle. The resulting approach may be less optimal from an efficiency point of view, but we would like to provide for collision-free navigation to the greatest extent possible. Since half of the moving obstacles are agents controlled by our own centralized system, we can take advantage of this to achieve safety for among our own robotic agents. Opponent robots can be incorporated into the calculation as well, but without any safety guarantees, since safe motion may not even be possible in the general case of agents that can move as fast as those that we control. For the remainder of this section, we will describe our approach to the path planning problem as part of the larger navigation system.

For path planning in our system, we model the environment in two dimensions (2-D), ignoring any dynamics constraints of the robot. These constraints are instead handled by a later module, allowing the planner to focus on generating a path valid for only static obstacles. The algorithm used in our system is the ERRT extension of the RRT-GoalBias planner [22], [23]. Due to the speed of the algorithm, a new plan can be constructed each control cycle, allowing the plan to track changes in the dynamic environment without the need for replanning heuristics. A more thorough description of our previous work on ERRT can be found in [26]. Since that work, only a new more efficient implementation has been completed, but the underlying algorithm is the same. It is described here in enough detail to be understood for the evaluations later in the paper.

Rapidly exploring random trees (RRTs) [22] employ randomization to explore large state spaces efficiently, and form the basis for a family of probabilistically complete, though nonoptimal, kinodynamic path planners [23]. Their strength lies in that they can efficiently find plans in relatively open or high dimensional spaces because they avoid the state explosion that discretization faces. A basic planning algorithm using RRTs is shown in Fig. 6, and the steps are as follows. Start with a trivial tree consisting only of the initial configuration. Then iterate: pick a random location q in the configuration space, then find the nearest point in the current tree using some distance metric, and extend that point toward q. Extending is defined as adding a new point to the tree that extends from a point in the existing tree toward some point q, while maintaining whatever motion constraints exist. The algorithm leaves the choice of the distribution of q as a free parameter, and this is the key to efficient search. A basic RRT tree picks q uniformly throughout the environment. This will build a tree which tends toward uniform convergence of the environment [23], but does not focus its search on any particular point. For path planning, we want to focus on a specific goal point g, so RRT-GoalBias modifies the basic RRT algorithm's distribution of q by making it bimodal: with probability p, choose the goal location g, while with probability $1-p$, pick a point x uniformly from the configuration space. Using a typical value of p between 0.05 and 0.1, the algorithm combines random exploration with biased search toward the goal, and the tree quickly reaches the goal for typical environments [22].

To convert the RRT algorithm into a planner, we need two simple additions. One is to restrict the tree to free space, where it will not collide with any obstacles. This can be accomplished easily by only adding nodes for extensions that will not hit obstacles. To make the tree into a planner, we only need to stop once the tree has reached a point sufficiently close to the goal location. Because the root of the tree is the initial position of the robot, tracing up from any leaf gives a valid path through free space between that

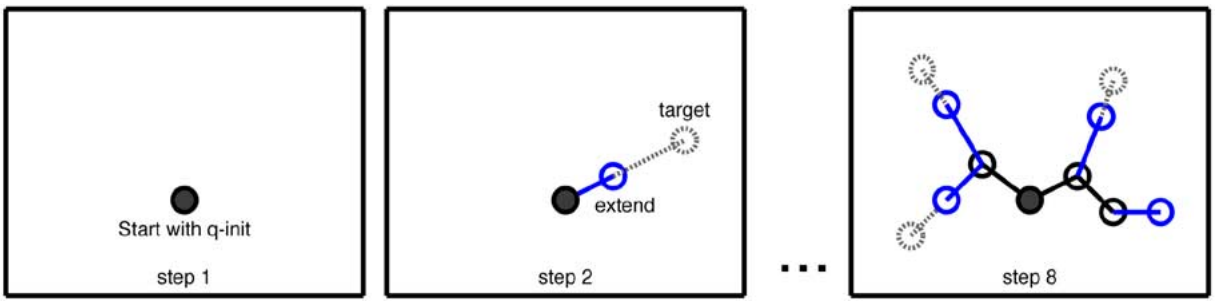

Fig. 6. *Example growth of an RRT tree for several steps. Each iteration, a random target is chosen and the closest node in the tree is "extended" toward the target, adding another node to the tree.*

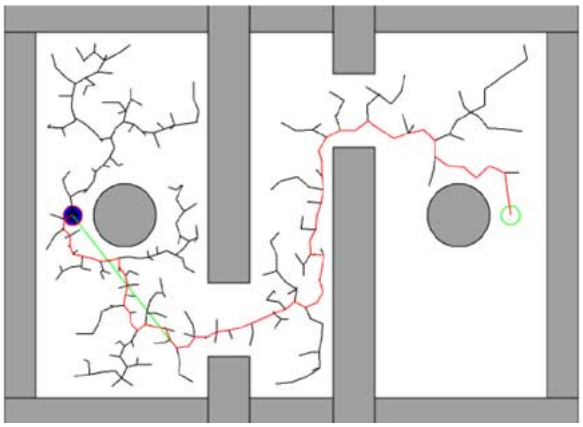

Fig. 7. *A robot on the left finds a path to a goal on the right using ERRT. The current location is shown as a filled circle, while the goal location is shown as an outlined circle.*

leaf and the initial position. Thus, a leaf near the goal can be used as a path which solves the planning problem.

Execution Extended RRT (ERRT) adds the notion of a waypoint cache, which is a fixed-size lossy store of nodes from successful plans in previous iterations of the planner. Whenever a plan is found, all nodes along the path are added to the cache with random replacement of previous entries. Then during planning, random targets are now chosen from three sources instead of two. In other words, with probability p it picks the goal, with probability q it picks a random state from the waypoint cache, and with the remaining $1 - p - q$ it picks a random state in the environment.

In order to implement ERRT, we need an *extend* operator, a distance function between robot states, a distribution for generating random states in the environment, and a way of determining the closest point in a tree to a given target state. Our implementation uses Euclidean distance for the distance function and the uniform distribution for generating random states. The nearest state in the tree is determined using KD-Trees, a common technique for speeding up nearest neighbor queries. Finally, the *extend* operator simply steps a fixed distance along the path from the current state to the target. For a planner ignoring dynamics, this is the simplest way to guarantee the new state returned is closer to the intermediate target than the parent. Our step size is set to the minimum of the robot's radius and the distance to the randomly chosen target. An image of the planner running in simulation is shown in Fig. 7, and a photograph of a real robot controlled by the planner is shown in Fig. 8. To simplify input to the motion control, the resulting plan is reduced to a single target point, which is the furthest node along the path that can be reached with a straight line that does not hit obstacles. This simple postprocess smooths out local nonoptimalities in the generated plan.

Fig. 8. *A robot (lower left) navigating at high speed through a field of static obstacles.*

V. MOTION CONTROL

Once the planner determines a waypoint for the robot to drive to in order to move toward the goal, this target state is fed 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. 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 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 Fig. 9. The area traced out by the trapezoid is the displacement effected by the robot. For motion in 2-D, the problem is decomposed as a one-dimensional (1-D) motion problem along the axis from the robots' current position to the desired target, and another 1-D deceleration perpendicular to that axis.

While the technique is well known, the implementation focuses on robustness even in the presence of numerical

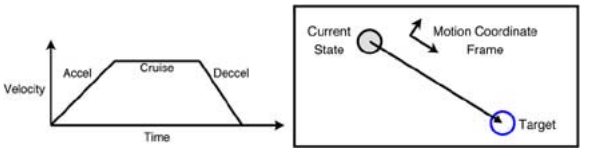

Fig. 9. *Our motion control approach uses trapezoidal velocity profiles. For the 2-D case, the problem can decomposed into two 1-D problems, one along the difference between the current state and the target state, and the other along its perpendicular.*

inaccuracies, changing velocity, or acceleration constraints, and the inability to send more than one velocity command per cycle. First, for stability in the 2-D case, if the initial and target points are close, the coordinate frame becomes degenerate. In that case the last coordinate frame above the distance threshold is used. For the 1-D 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-based servo loop if accurate tracking is desired.

VI. DYNAMICS SAFETY SEARCH

In the two previous stages in the overall system, the planner ignored dynamics while the motion control ignored obstacles, which has no safety guarantees in preventing collisions between the agent and the world, or between agents. The "Dynamic Window" approach [37] is a search method which elegantly solves the first problem of collisions between the robotic agent and the environment. It is a local method, in that only the next velocity command is determined; however, it can incorporate nonholonomic constraints, limited accelerations, maximum velocity, and the presence of obstacles into that determination, thus guaranteeing safe motion for a robot. The search space is the velocities of the robot's actuated degrees of freedom. The two developed cases are for synchro-drive robots with a linear velocity and an angular velocity, and for holonomic robots with two linear velocities [37], [38]. In both cases, a grid is created for the velocity space, reflecting an evaluation of velocities falling in each cell. First, the obstacles of the environment are considered, by assuming the robot

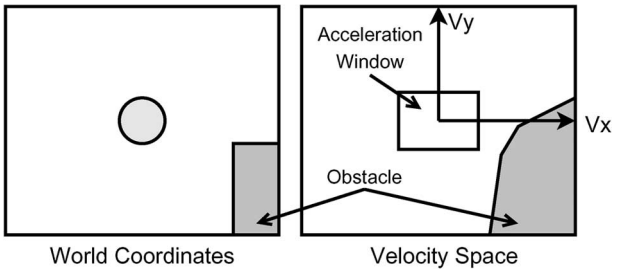

Fig. 10. *Example environment shown in world space and velocity space.*

travels at a cell's velocity for one control cycle and then attempts to brake at maximum deceleration while following that same trajectory. If the robot cannot come to a stop before hitting an obstacle along that trajectory, the cell is given an evaluation of zero. Next, due to limited accelerations, velocities are limited to a small window that can be reached within the acceleration limits over the next control cycle (for a holonomic robot this is a rectangle around the current velocities). An example is shown in Fig. 10. Finally, the remaining valid velocities are scored using a heuristic distance to the goal. It was used successfully in robots moving up to 1 m/s in cluttered office environments with dynamically placed obstacles [38].

Our approach extends the Dynamic Window approach to multiple velocity and acceleration bounded robots, and replaces the grid-based sampling with a randomized sampling approach which guarantees the preservation of safety if no sensor or action noise is present. The system model employed is n robots moving according to Newtonian physics with piecewise constant accelerations. Let a *motion segment* be defined as a time interval during which a robot is undergoing constant acceleration. Thus, the motion of each robot is defined by several motion segments. A segment for robot i starts at time t_0, where the robot is at position q_{0i} with velocity v_{0i}. The robot will execute a control acceleration u_i until time t_{1i}. At all times during the motion the robot is bounded by a circle (or a sphere in higher dimensions) of radius r_i. The equation of motion is the following:

$$q_i(t) = q_{0i} + v_{0i}(t - t_0) + \frac{1}{2}u_i(t - t_0)^2. \qquad (1)$$

For the safety search, two major checking primitives must be available. The first is to check a segment against static obstacles in the environment. This can be achieved with standard collision detection approaches using a piecewise linear approximation of the trajectory, with a margin added around the robot so that the actual trajectory lies within the linear approximation at all points. The types of trajectories generated for our approach typically have

little curvature, so a linear approximation is well suited. The second primitive is to check if a segment from one robot is safe against another robot's segment. One need only consider the parts of the trajectories during the intersecting portion of the segments' time intervals. The segment check can be derived from the closest point of approach under constant acceleration, which is the point in time where the distance between two bodies traveling according to (1) is minimal. Its calculation will be described later in this section. In both Dynamic Window and our safety search method, each robot i can be said to have three segments. The first is an acceleration action for the next control cycle over the time interval $[t_0, t_c]$. The acceleration for this period comes directly from the trapezoidal motion control, and will either be maximum acceleration, no acceleration, or maximum deceleration depending on the state of the robot and the time during the trajectory. From the resulting position of that immediate action, the next segment is to come to a complete stop at maximum deceleration. This is over the time interval $[t_c, t_{si}]$. Note that although all robots share a common value for t_c, t_{si} is dependent on the velocity of a robot and thus can be different for each one. The final segment is simply to remain stopped permanently, resulting in a time interval of $[t_{si}, \infty]$. An example velocity profile for two robots is shown in Fig. 11. Note that although the profiles always come to a stop immediately after executing a command, this is merely to guarantee safety. In the next control cycle, each robot will try to replace the emergency stop predicted from the last frame with an action closer to the desired velocity. By always providing for the *ability* of an agent to stop, we guarantee safety, although ideally this will not be necessary. If no agents interact, then we will always execute the command from motion control. When they do interact, we will attempt a search to find an alternative action to stopping that more closely matches the desired acceleration command while still guaranteeing safety.

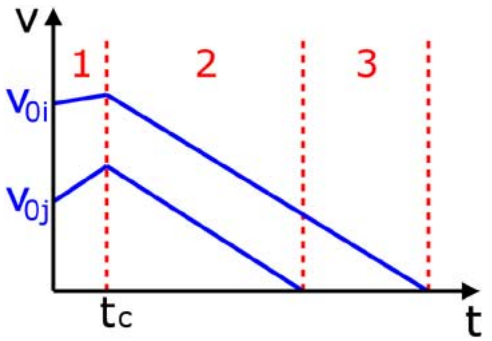

Fig. 11. *Example velocity profile of two agents i and j. Each agent starts at a distinct velocity (v_{0i} and v_{0j}), and will execute an acceleration command from the current time $t = 0$ until control cycle time t_c. Then each robot assumes that it will come to an emergency stop at maximum deceleration. This results in three segments of constant acceleration in the relative motion, denoted by the dotted lines.*

One feature of our method is that safe commands are determined purely in terms of accelerations, rather than velocity targets as in the Dynamic Window approach. This leads to a more straightforward implementation for holonomic robots powered by electric motors, since they have nearly direct control on their forces and thus their accelerations. For robots where velocity control is more appropriate, the short control interval of t_c makes it relatively easy to integrate the acceleration to create a velocity space search as in Dynamic Window. For simplicity we will only describe the acceleration-based approach.

If each robot $i \in [1, n]$ starts at t_0 with position q_{0i} and velocity v_i, and each robot has a maximum deceleration of a_d, we would like to determine a safe control acceleration u_i that most closely approximates a target acceleration a_{ti} given by motion control. In our approach we minimize the squared Euclidean distance $\|u_i - a_{ti}\|^2$ subject to u_i maintaining safety. Other metrics may be more appropriate for different robot models, and the metric need only to be able to compare two accelerations, allowing almost any metric to be applied easily to this system.

Using the two segment–obstacle and segment–segment collision checking primitives from the path planner, along with the three motion segments each robot undergoes, we can construct functions for use by a high-level search procedure. The first function we will call CheckObs(i, u_i) : *bool*, which returns true if and only if the three motion segments of robot i will not hit a world obstacle, given an acceleration of u_i during the first motion segment (the control segment where $t \in [t_0, t_c]$). Note that for an implementation, we need only check the first two motion segments, since the position during the third (stationary) segment is equal to the final position of the second segment. The second function we call CheckRobot(i, u_i, j, u_j) : bool, which checks that robot i and j do not collide during any of their respective motion segments, given accelerations of u_i and u_j, respectively, during the first controlled motion segment. This requires checking the closest point between the robots over the first segment $[t_0, t_c]$, followed by segments $[t_c, \min(t_{si}, t_{sj})]$, and finally $[\min(t_{si}, t_{sj}), \max(t_{si}, t_{sj})]$. Referring to Fig. 11, each point at which the acceleration changes for either of the robots defines a new segment. Thus, during each segment, the two robots are bodies undergoing constant acceleration. By using the starting relative position, relative velocity, and relative acceleration for each segment, we can define a quadratic relative position function as shown in (2). The minimum of the function can be solved algebraically or numerically, and the derivation is straightforward. A solution to similar formulation is given by Fiorini *et al.* [33]. If the distance between the closest points of any segment are less than the summed radius of the two robot's bounding circles, $r_i + r_j$, CheckRobot must return false, indicating that the robots could collide. Otherwise, the two

Table 1 High-Level Search Routines for Velocity-Space Safety Search

```
function CheckAccel(i:robot, a:vector):boolean
    if ¬CheckObs(i,a) then
        return False
    foreach j ∈ [1,n], j ≠ i
        if ¬CheckRobot(i,a,j,a_j) then
            return False
    end
    return True
end
procedure FindSafeAccel()
    foreach i ∈ [1,n]
        u_i ← − a_d v_i/‖v_i‖
        d_i ← ‖u_i − a_ti‖^2
    end
    foreach i ∈ [1,n]
        if CheckAccel(i,a_ti) then
            u_i ← − a_d v_i/‖v_i‖
            d_i ← 0
        else
            foreach j ∈ [1,m]
                a_r ← RandomAccel
                if CheckAccel(i,a_r) ∨ ‖a_r = v_ti‖^2 < d_i then
                    u_i ← a_r
                    d_i ← ‖a_r = v_ti‖^2
                end
            end
        end
    end
end
```

bounding volumes are disjoint at all times during the trajectories, and CheckRobot should return true. With these two functions available, we can write the high-level search routines CheckAccel and FindSafeAccel, listed in Table 1

$$\begin{aligned}\Delta q_{ij}(t) &= q_i(t) - q_j(t)\\ \Delta q_{ij}(t) &= (q_{0i} - q_{0i}) + (v_{0i} - q_{0i})(t - t_0)\\ &\quad + \frac{1}{2}(u_i - u_j)(t - t_0)^2. \end{aligned} \tag{2}$$

CheckAccel checks if an acceleration would hit a static obstacle or the trajectory of another robot. FindSafeAccel first initializes all robot commands to stopping, then it for each robot it tries to improve the command. An improvement is any safe acceleration that more closely matches the desired acceleration returned by the motion control (a_{ti} for robot i). The distance from the desired acceleration is stored in the variable d_i. If a robot is not near any obstacles or other robots, then it can execute any control acceleration safely. This optimistic case comes next, checking if the a_{ti} is safe. If it is, we can move on to the next robot. If not, we use random sampling of the of the robots' feasible accelerations. Samples are returned by a function RandomAccel, and each is checked to see if it more closely matches the target acceleration. If no better acceleration was found, the robot will execute the previously assigned default command of a maximum deceleration to a stop.

Overall safety is guaranteed by forward chaining of the safety guarantee across control cycles; the stopping action is put off into the future whenever possible, but always maintained as a possible action to execute (starting in the next cycle) whenever decisions are made for the current cycle. Specifically, CheckAccel returns true if and only if a given robot i can execute an acceleration command a while still maintaining a safe future stopping action. Thus, any action passing CheckAccel will allow the agent to stop safely beginning in the next control cycle. FindSafeAccel starts by initializing each command to a stopping action, which should be guaranteed safe from the previous control cycle. For this to hold, the initial conditions of the system must be safe, but this is easily satisfied by starting the system with all agents stopped. After initializing the robot acceleration commands, FindSafeAccel then tries to improve each robot's command in turn, first checking the desired command for that robot, and then sampling randomly for one which passes CheckAccel. If any such velocity is found, it must be safe due to the properties of CheckAccel, and if no action is found, then the default action of stopping remains, which was guaranteed safe from the previous cycle. In this way, safety can be maintained continuously, provided that all moving agents are under control of the centralized algorithm.

The safety system as a whole guarantees that robots starting from a safe configuration will always choose accelerations and thus velocities that maintain this safety condition. The approach always provides for a safe stopping action in future control cycles, but engages in search each cycle to find a command which better matches the desired velocity to achieve the overall navigation objective safely. Safety is defined as the ability to stop without hitting a static or moving obstacle, provided all moving obstacles are a robot participating in the algorithm. Other moving circular obstacles (such as opponent robots) can be incorporated into the algorithm's calculations, although no guarantees can be provided for safety in this more general case. Empirically observed behavior in this situation has, however, been favorable (our most recent team, CMRoboDragons, did not receive a single pushing or hitting penalty throughout the RoboCup 2005 competition).

VII. EVALUATION AND RESULTS

For evaluation, we have focused on safe navigation, as meaningful quantitative results are difficult to obtain for objective assignment and evaluation. For objective evaluation in our implemented system, we were able evaluate 100 random points each cycle using the described passing evaluation function, resulting in an execution time of only 0.29 ms per supporting robot. By maintaining the best values across control cycles, we could effectively search 6000 samples per second and quickly find a near optimal solution to the objective function. In the RoboCup 2005 competition, passing was critical to the success of our team,

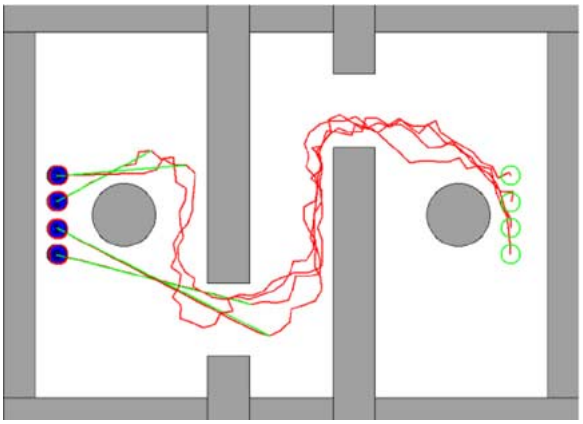

Fig. 12. *The evaluation environment consists of four robots which must alternately achieve goals on the left and right side of the environment. The state pictured here is just at the beginning of a run, with the robots represented as filled circles and their respective goals represented as outlined circles. The plans calculated from the path planner are shown, but the search trees are omitted for clarity.*

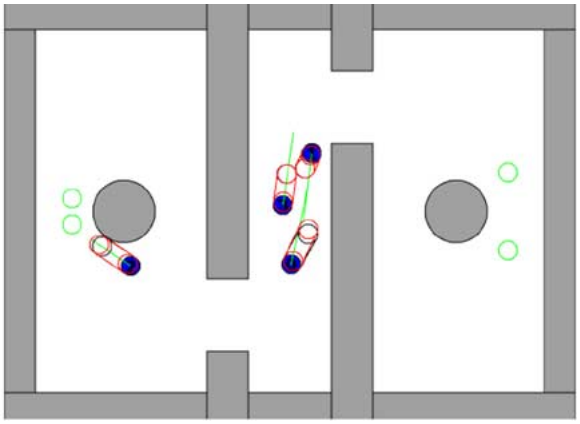

Fig. 13. *Multiple robots navigating traversals in parallel. The outlined circles and lines extending from the robots represent the chosen command followed by a maximum rate stop.*

accounting for over 75% of the 23 goals scored, leading to our fourth-place finish after five wins and two losses.

For navigation, we created an evaluation domain which is a simulation of the RoboCup small-size environment, but with additional large obstacles added in order to increase the likelihood of robot interactions. The environment is shown in Fig. 12. For the tests, four robots were given the task of traveling from the leftmost open area to the rightmost open area, and back again for four iterations. Each robot has separate goal point separated from the others by slightly more than a robot diameter. The 90-mm-radius robots represent the highest performance robots we have used in RoboCup. Each has a command cycle of 1/60 s, a maximum velocity of 2 m/s, acceleration of 3 m/s^2, and deceleration of 6 m/s^2. Because different robots have different path lengths to travel, after a few traversals robots start interacting while trying to move in opposed directions. Fig. 13 shows an example situation in the middle of a test run. On average, four full traversals by all of the robots took about 30 s of simulated time.

For the evaluation metric, we chose interpenetration depth multiplied by the time spent in those invalid states. To more closely model a real system, varying amounts of position sensor error were added, so that the robot's reported position was a Gaussian deviate of its actual position. This additive random noise represents vision error from overhead tracking systems. Velocity sensing and action error were not modeled here for simplicity; these errors depend heavily on the specifics of the robot and lack a widely applicable model. First, we compared using the planner and motion control but enabling or disabling the safety search. Each data point is the average of 40 runs (four robots, each with ten runs), representing about 20 min of simulated run time. Fig. 14 shows the results, where it is clearly evident that the safety search significantly decreases the total interpenetration time. Without the safety search, increasing the vision error makes little difference in the length and depth of collisions. Ideally the plotted curve without safety search would be smooth, but due to the random nature of the collisions it displays extremely high variance, and many more runs would be needed to demonstrate a dependence on vision noise. However, even with the noise, it is clear that the curve is significantly worse than when safety search is used. Next, we evaluated the safety search using different margins of 1–4 mm around the 90-mm robots, plotted against increasing vision error (see Fig. 15). As one would expect, with little or no vision error even small margins suffice for no collisions, but as the error increases there is a benefit to higher margins for the safety search, reflecting the uncertainty in the actual position of the robot.

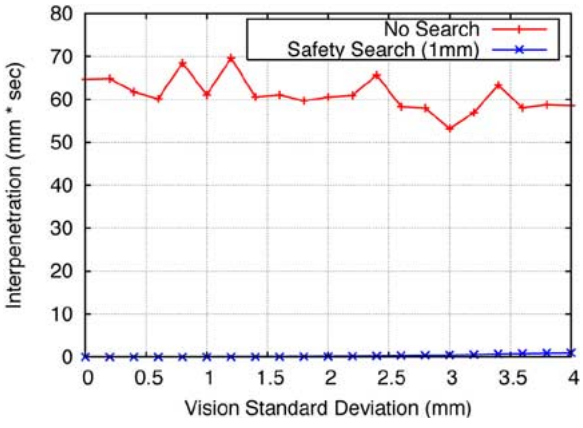

Fig. 14. *Comparison of navigation with and without safety search. Safety search significantly decreases the metric of interpenetration depth multiplied by time of interpenetration.*

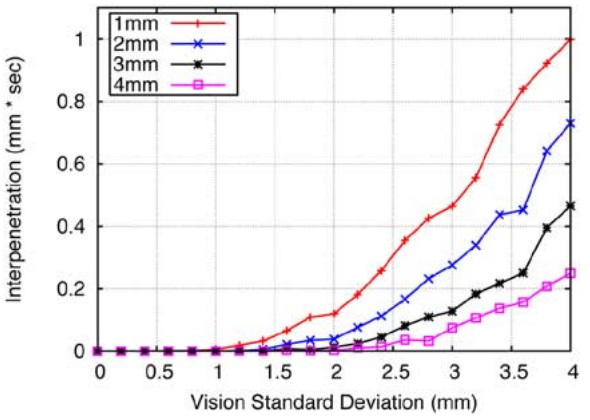

Fig. 15. *Comparison of several margins under increasing vision error. The four different margins used are listed in the key, while increasing vision standard deviation is plotted against the collision metric of interpenetration depth multiplied by time of interpenetration.*

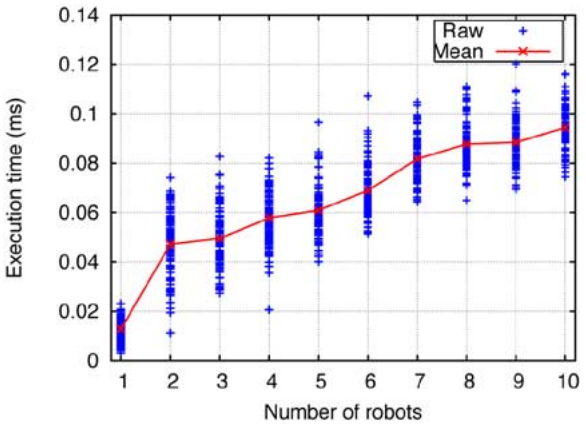

Fig. 16. *Average execution time of safety search for each agent, as the total number of agents increases. For each robot count, 100 trials of the left-right traversal task were run. Both the raw data and means are shown.*

The other variable of interest is the cost in running time of planning and the safety search. In the tests above, the planner was limited to 1000 nodes, and the safety search was limited to 500 randomly sampled velocities. The system executed with an average run time of 0.70 ms per control cycle without the velocity safety search, and 0.76 ms with it. Thus, safety search does not add a noticeable overhead to the navigation. Since this is a real-time system, however, we are most interested in times near the worst case. Looking at the entire distribution of running times, the 95th percentiles are 1.96 ms without safety search and 2.04 ms with it. In other words, for 95% of the control cycles, runtime was less than 2.04 ms with safety search, and the execution time was only 4% longer than without safety search. Next, to measure the scalability of the safety search approach, the same traversal task was repeated while varying the number of robots from one to ten. With increasing numbers of agents in a fixed-size environment, we can hope to gauge how well the algorithm performs under increasing amounts of clutter due to moving objects. The timing results for safety search are shown in Fig. 16. The function appears to scale in a roughly linear fashion for more than one agent, though it is too noisy to determine with any certainty. The most important observation is that it does not scale in a superlinear fashion, which would cause difficulties for moderately large teams.

When applied in a more realistic RoboCup environment, the lack of cluttering obstacles makes the additional execution time of safety search contribute less than 1% to overall navigation time. This is because the already efficient safety search is only needed in the rare cases when two agents are likely to collide. On the physical robots, we have qualitatively noted that the frequency of collisions between teammates goes from several times a minute to once every several minutes, even for tasks with highly conflicting navigation goals. The remaining collisions appear to be a result of our imperfect model of the robots while operating at high speed, and the resulting errors in tracking. In the future, we hope to explore more objective measurements of collisions for the real agents. In both the simulated and real RoboCup domain, the safety search system has been shown to significantly decrease cases where robots collide, while preserving the real-time performance of a navigation system which uses the ERRT planner alone.

VIII. CONCLUSION

This paper described a navigation system for the real-time control of multiple high performance robots in a dynamic, unpredictable domain. It also gave a brief overview of how the navigation system was employed in the overall system. Specifically, navigation targets for multiple supporting roles were determined by using a numerical sampling technique to find maximums in real-valued evaluation functions. This approach can operate easily with tight real-time constraints and scales easily to moderate size teams. The navigation module addresses the issue of carrying out such targets, allowing multiple robots to operate safely without collisions, even though agents are free to change their desired velocity commands each control cycle. The current solution is centralized, relying on perfect communication of world state and actions. However, the system purposely does not rely on any additional communication; in particular, no form of deliberative communication of intents is used. Compared to a purely monolithic design which plans in parallel for all agents in a joint state and action space, the approach in this paper minimizes communicated state to a known bounded size. It is expected

that this would make decentralization easier to achieve compared to a system without bounded communication. In its current form, however, the system does make demands of sensor accuracy that may not yet be practical for multiple distributed robots which use only local sensing. We feel the primary contribution is to serve as a successful model and example for similar problem domains, and as a starting point for future extensions which relax some of the assumptions. ■

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments, which led to significant improvements in the final version of this paper. The authors would also like to thank all past and current members of our RoboCup teams, specifically the small-size teams CMUnited, CMDragons, and CMRoboDragons. Without the work from team members, none of this research would have been possible.

REFERENCES

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "RoboCup: The robot World Cup initiative," in *Proc. IJCAI-95 Workshop Entertainment and AI/ALife*, 1995, pp. 340–347.

[2] J. Bruce, M. Bowling, B. Browning, and M. Veloso, "Multi-robot team response to a multi-robot opponent team," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2003, vol. 2, pp. 2281–2286.

[3] M. Veloso, M. Bowling, S. Achim, K. Han, and P. Stone, "The CMUnited-98 champion small robot team," in *RoboCup-98: Robot Soccer World Cup II*. Berlin, Germany: Springer-Verlag, 1999.

[4] M. Bowling and M. Veloso, "Motion control in dynamic multi-robot environments," in *Int. Symp. Computational Intelligence in Robotics and Automation (CIRA'99)*, 1999, pp. 168–173.

[5] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, 2004.

[6] E. Uchibe, T. Kato, M. Asada, and K. Hosoda, "Dynamic task assignment in a multiagent/multitask environment based on module conflict resolution," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2001, pp. 3987–3992.

[7] A. D'Angelo, E. Menegatti, and E. Pagello, "How a cooperative behavior can emerge from a robot team," in *Proc. 7th Int. Symp. Distributed Autonomous Robotic Systems*, 2004, pp. 71–80.

[8] B. Browning, J. Bruce, M. Bowling, and M. Veloso, "STP: Skills, tactics and plays for multi-robot control in adversarial environments," *J. Syst. Control Eng.*, vol. 219, no. I1, pp. 33–52, Feb. 2005.

[9] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. RA-2, no. 1, pp. 14–23, Mar. 1986.

[10] R. B. Tilove, "Local obstacle avoidance for mobile robots based on the method of artificial potentials," General Motors Res. Labs., Res. Pub. GMR-6650, Sep. 1989.

[11] R. C. Arkin, "Motor schema-based mobile robot navigation," *Int. J. Robot. Res.*, vol. 8, no. 4, pp. 92–112, Aug. 1989.

[12] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1991, pp. 1398–1404.

[13] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[14] A. Tews and G. Wyeth, "MAPS: A system for multi-agent coordination," *Adv. Robot.*, vol. 14, no. 1, pp. 37–50, 2000.

[15] D. Ball and G. Wyeth, "Multi-robot control in highly dynamic, competitive environments," in *RoboCup 2003: Robot Soccer World Cup VII*, vol. 3020, *Lecture Notes in Computer Science*. Heidelberg, Germany: Springer, 2003, pp. 385–396.

[16] T. Weigel, J. S. Gutmann, M. Dietl, A. Kleiner, and B. Nebel, "CS Freiburg: Coordinating robots for successful soccer playing," *IEEE Trans. Robot. Autom.*, vol. 18, no. 5, pp. 685–699, Oct. 2002.

[17] N. J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

[18] T. Laue and T. Rofer, "A behavior architecture for autonomous mobile robots based on potential fields," in *RoboCup 2004: Robot Soccer World Cup VIII*, vol. 3276, *Lecture Notes in Computer Science*. Heidelberg, Germany: Springer-Verlag, 2005, pp. 122–133.

[19] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. 20th IEEE Symp. Foundations of Computer Science*, 1979, pp. 421–427.

[20] D. E. Koditschek, "Exact robot navigation by means of potential functions: Some topological considerations," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987, pp. 1–6.

[21] A. Stentz. (2006, Jul.). Optimal and efficient path planning for unknown and dynamic environments. *Int. J. Robot. Autom.* [Online]. *10(3). Available: http://www.frc.ri.cmu.edu/axs/index.html*

[22] S. M. LaValle, *Rapidly-exploring random trees: A new tool for path planning*, Tech. Rep. 98-11, Oct. 1998.

[23] S. M. LaValle, J. James, and J. Kuffner, "Randomized kinodynamic planning," Int. J. Robot. Res., vol. 20, no. 5, pp. 378–400, May 2001.

[24] L. E. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1994, pp. 2138–2145.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[26] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. IEEE Conf. Intelligent Robots and Systems (IROS)*, 2002, pp. 2383–2388.

[27] V. Boor, M. H. Overmars, and F. van der Stappen, "The Gaussian sampling strategy for for probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1999, pp. 1018–1023.

[28] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1998, pp. 113–120.

[29] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Proc. Workshop Algorithmic Foundations of Robotics*, 1998, pp. 155–168.

[30] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2003, pp. 4420–4426.

[31] P. Isto, "Constructing probabilistic roadmaps with powerful local planning and path optimization," in *Proc. IEEE Conf. Intelligent Robots and Systems (IROS)*, 2002, pp. 2323–2328.

[32] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 4, pp. 477–521, 1987.

[33] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1993, pp. 560–565.

[34] ——, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, Jul. 1998.

[35] F. Large, Z. Shiller, S. Sekhavat, and C. Laugier, "Towards real-time global motion planning in a dynamic environment using the NLVO concept," in *Proc. IEEE Conf. Intelligent Robots and Systems (IROS)*, 2002, pp. 607–612.

[36] D. Hsu, R. Kindel, J. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[37] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[38] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1999, pp. 341–346.

ABOUT THE AUTHORS

James R. Bruce received the B.S. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 2000. He is currently working toward the Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA. His thesis research is on real-time motion planning for dynamic unpredictable environments.

His research interests include randomized algorithms for motion planning in continuous domains, high speed color machine vision, and behavior and control systems for high performance robots. He has participated in the RoboCup competition for each of the last seven years.

Manuela M. Veloso received the B.S. degree in electrical engineering and the M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico, Lisbon, Portugal, in 1980 and 1984, respectively, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1992.

She is Professor of Computer Science at Carnegie Mellon University. She researches in the area of artificial intelligence with focus on planning, control learning, and execution for single and multirobot teams. Her algorithms address uncertain, dynamic, and adversarial environments. She has developed teams of robot soccer agents, which have been RoboCup world champions several times. She investigates learning approaches to a variety of control problems, in particular, the performance optimization of algorithm implementations, and plan recognition in complex data sets.

Prof. Veloso is a Fellow of the American Association of Artificial Intelligence. She is Vice President of the RoboCup International Federation. She was awarded an NSF Career Award in 1995 and the Allen Newell Medal for Excellence in Research in 1997. She is Program Cochair of 2005 National Conference on Artificial Intelligence and the Program Chair of the 2007 International Joint Conference on Artificial Intelligence.