REAL-TIME MULTI-ROBOT MOTION PLANNING WITH SAFE DYNAMICS *

James Bruce, Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213, USA
{jbruce,mmv}@cs.cmu.edu

Abstract This paper introduces a motion planning system for real-time control of multiple high performance robots in dynamic and unpredictable domains. It consists of a randomized realtime path planner, a bounded acceleration motion control system, and a randomized velocity-space search for collision avoidance of multiple moving robots. The realtime planner ignores dynamics, simplifying planning, while the motion control ignores obstacles, allowing a closed form solution. This allows up to five robots to be controlled 60 times per second, but collisions can arise due to dynamics. Thus a randomized search is performed in the robot's velocity space to find a safe action which satisfies both obstacle and dynamics constraints. The system has been fully implemented, and empirical results are presented.

Keywords: realtime path planning, multirobot navigation

1. Introduction

All mobile robots share the need to navigate, creating the problem of motion planning. When multiple robots are involved, the environment becomes dynamic, and when noise or external agents are present, the environment also becomes unpredictable. Thus the motion planning system must be able to cope with dynamic, unpredictable domains. To take advantage of high performance robots, and respond quickly to

*This work was supported by United States Department of the Interior under Grant No. NBCH-1040007, and by Rockwell Scientific Co., LLC under subcontract No. B4U528968 and prime contract No. W911W6-04-C-0058 with the US 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.

external changes in the domain, the system must also run at real-time rates. Finally, navigation at high speeds means respecting dynamics constraints in the robot motion to avoid collisions while staying within the operational bounds of the robot. When multiple robots are introduced, the system must find solutions where no robots collide while satisfying each robot's motion constraints. This paper describes a domain with these properties, and a motion planning system which satisfies the requirements for it. In the remainder of this section, the domain will be presented, and following sections will describe the three major parts of the system mentioned above. The system will then be evaluated in its entirety as to how well it solves navigation tasks.

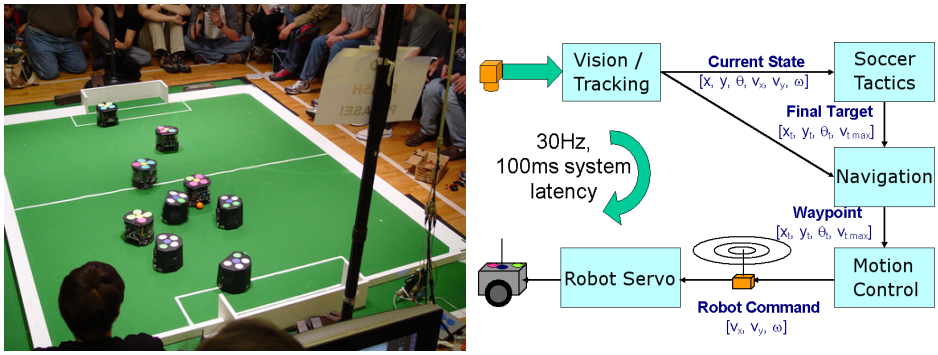

Figure 1. Two teams are shown playing soccer in the RoboCup small size league (left), and the the overall system architecture for CMDragons (right).

The motivating domain for this work is the RoboCup F180 "small size" league (Kitano et al., 1995). It involves teams of five small robots, each up to 18cm in diameter and 15cm height. The robot teams are entered into a competition to play soccer against opponent teams fielded by other research groups. During the game no human input is allowed, thus the two robot teams must compete using full autonomy in every aspect of the system. The field of play is a carpet measuring 4.9m by 3.8m, with a 30cm border around the field for positioning outside the field of play (such as for free kicks). An earlier (half size) version of the field is pictured on the left of Figure 1. 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 on the right of Figure 1 (Bruce et al., 2003). 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 soccer system implements most of its actions through a navigation module, which provides path planning, obstacle avoidance, and motion control. Finally, velocity commands are sent to the robots via a serial radio link. Due to its competitive nature, teams have pushed robotic technology to its limits, with the small robots travelling over $2m/s$, accelerations between $3 - 6m/s^2$, and kicking the golf ball used in the game at $4-6m/s$. Their speeds require every module to run in realtime to minimize latency, while leaving enough computing resources for the other modules to operate. In addition, the algorithms must operate robustly due to the full autonomy requirement.

Navigation is a critical component in the overall system described above, and the one we will focus on in this paper. The system described here is meant as a drop-in replacement for the navigation module used successfully by our team since 2002, and building on experience gained since 1997 working on fast navigation for small high performance robots (Bowling and Veloso, 1999). For our model, we will assume a centralized system, although the interaction required between robots will be minimized which should make decentralization a straightforward extension. It comprises of three critical parts; A path planner, a motion control system, and a velocity space search for safe motions. Although motivated by the specific requirements of the RoboCup domain, it should be of general applicability to domains where several robots must navigate within a closed space where both high performance and safety are desired.

2. Path Planning

For path planning, the navigation system models the environment in 2D, and also ignoring dynamics constraints, which will instead be handled by a later module. The algorithm used is the ERRT extension of the RRT-GoalBias planner (LaValle, 1998; LaValle and James J. Kuffner, 2001). Due to the speed of the algorithm, a new plan can be constructed each control cycle, allowing it 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 (Bruce and Veloso, 2002). Since then, 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) (LaValle, 1998) employ randomization to explore large state spaces efficiently, and form the basis for

a family of probabilistically complete, though non-optimal, kinodynamic path planners (LaValle and James J. Kuffner, 2001). 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 Figure 2, and the steps are as follows: Start with a trivial tree consisting only of the initial configuration. Then iterate: With probability p, find the nearest point in the current tree and extend it toward the goal g. Extending means adding a new point to the tree that extends from a point in the tree toward g while maintaining whatever motion constraints exist. Alternatively, with probability $1 - p$, pick a point x uniformly from the configuration space, find the nearest point in the current tree, and extend it toward x. Thus the tree is built up with a combination of random exploration and biased motion towards the goal configuration.

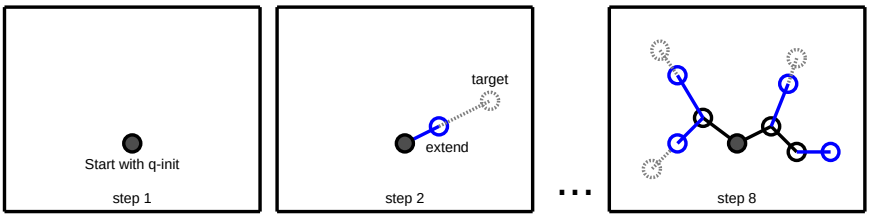

Figure 2. Example growth of an RRT tree for several steps. Each iteration, a random target is chosen and the closest node in the tree is "extended" toward the target, adding another node to the tree.

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 obstacles. This can be accomplished 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 leaf and the initial position. Thus finding a leaf near the goal is sufficient to solve 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 it 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 Figure 3, and a photograph of a real robot controlled by the planner is shown in Figure 4. 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 non-optimalities in the generated plan.

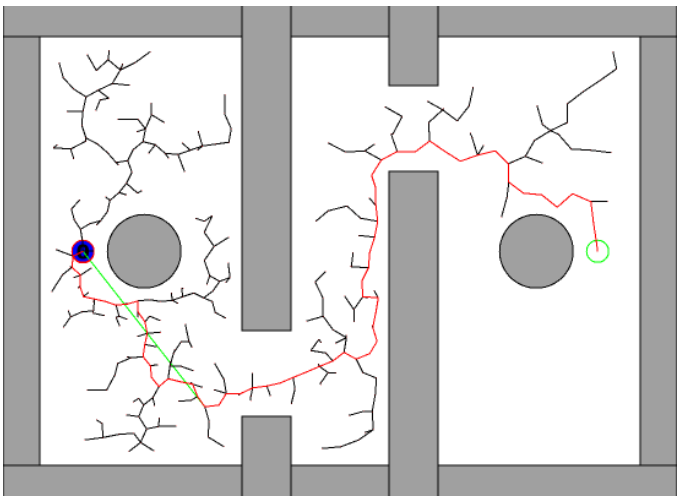

Figure 3. A robot on the left finds a path to a goal on the right using ERRT.

3. 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 con-

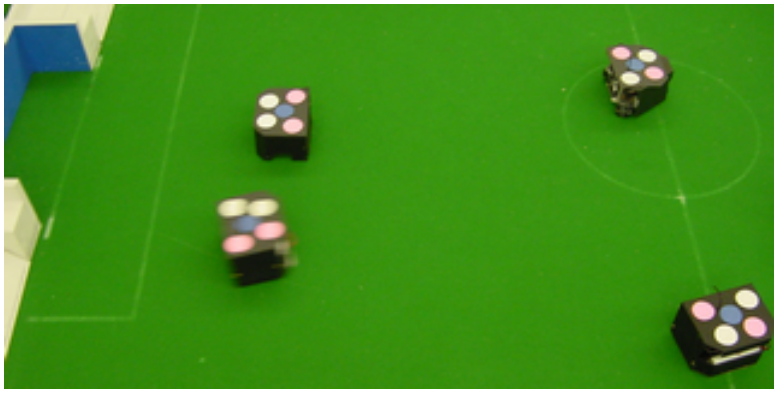

Figure 4. A robot (lower left) navigating at high speed through a field of static obstacles.

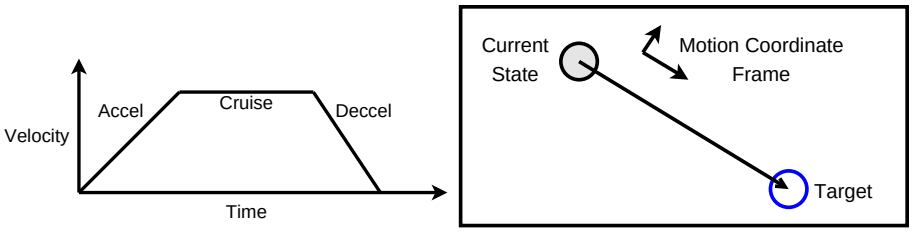

Figure 5. Our motion control approach uses trapezoidal velocity profiles. For the 2D case, the problem can decomposed into two 1D problems, one along the difference between the current state and the target state, and the other along its perpendicular.

trol 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 (Figure 5. The area traced out by the trapezoid is the displacement effected by the robot. For motion in 2D, the problem is decomposed as a 1D motion problem along the axis from the robots' current position to the desired target, and another 1D deceleration perpendicular to that axis.

While the technique 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 close, the coordinate frame becomes 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 is opposite 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 overshoot, but does mean that very small short motions will not be taken (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.

4. Velocity Space 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 (Fox et al., 1997) 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

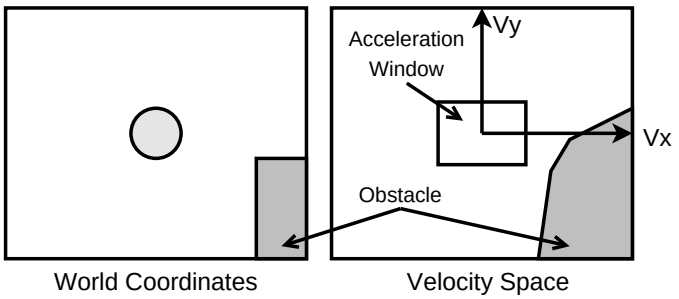

Figure 6. Example environment shown in world space and velocity space.

incorporate non-holonomic 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 (Fox et al., 1997; Brock and Khatib, 1999). 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 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 Figure 6. Finally, the remaining valid velocities are scored using a heuristic distance to the goal. It was used successfully in robots moving up to 1m/s in cluttered office environments with dynamically placed obstacles (Brock and Khatib, 1999).

Our approach first replaces the grid with randomized sampling, with memory of the acceleration chosen in the last frame. Static obstacles are handled in much the same way, by checking if braking at maximum deceleration will avoid hitting obstacles. The ranking of safe velocities is done by choosing the one with the minimum Euclidean distance to the desired velocity given by the motion control. For moving bodies (robots, in this case) we have to measure the minimum distance between two accelerating bodies. The distance can be computed by solving two fourth degree polynomials (one while both robots are decelerating, and the second while one robot has stopped and the other is still trying to stop).

For simplicity however we solved the polynomials numerically. Using this primitive, the velocity calculations proceeded as follows. First, all the commands were initialized to be maximum deceleration for stopping. Then as the robots chose velocities in order, they consider only velocities that don't hit an environment obstacle or hit the other robots based on the latest velocity command they have chosen. The first velocity tried is the exact one calculated by motion control. If that velocity is valid (which it normally is), then the sampling stage can be skipped. If it is not found, first the best solution from the last cycle is tried, followed by uniform sampling of accelerations for the next frame. Because the velocities are chosen for one cycle, followed by maximum deceleration, in the next cycle the robots should be safe to default to their maximum deceleration commands. Since this "chaining" assumption always guarantees that deceleration is an option, robots should always be able to stop safely. In reality, numerical issues, the limitations of sampling, and the presence of noise make this perfect-world assumption fail. To deal with this, we can add small margins around the robot, and also rank invalid commands. In the case where no safe velocity can be found, the velocity chosen is the one which minimizes the interpenetration depth with other robots or obstacles. This approach often prevents the robot from hitting anything at all, though it depends on the chance that the other robots involved *can* choose commands to avoid the collision.

Taken together, the three parts of the navigation system consisting of planner, motion control, and safety search, solve the navigation problem in a highly factored way. That is, they depend only on the current state and expected next command of the other robots. Factored solutions can be preferable to global solutions including all robots degrees of freedom as one planning problem for two reasons. One is that factored solutions tend to be much more efficient, and the second is that they also have bounded communication requirements if the algorithm needs to be distributed among agents. Each robot must know the position and velocity of the other robots, and communicate any command it decides, but it does not need to coordinate at any higher level beyond that.

5. Evaluation and Results

The evaluation environment the same as shown in Figure 3, which matches the size of the RoboCup small size field, but has additional large obstacles with narrow passages in order to increase the likelihood of robot interactions. The $90mm$ radius robots represent the highest performance robots we have used in RoboCup. Each has a command cycle of 1/60 sec, a maximum velocity of $2m/s$, acceleration of $3m/s^2$,

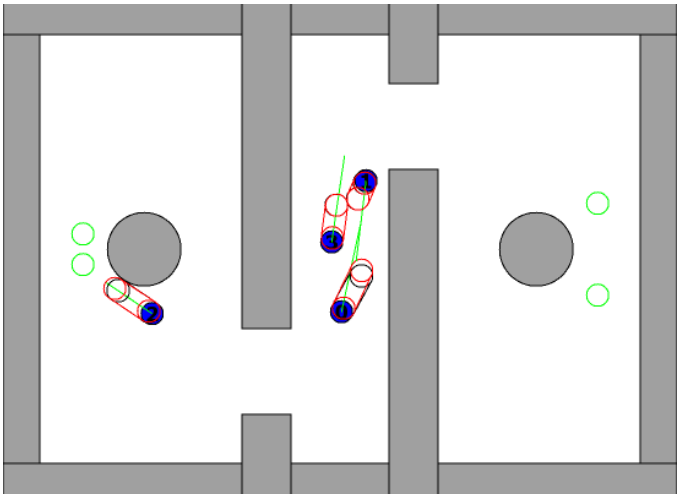

Figure 7. Multiple robots navigating traversals in parallel. The outlined circles and lines extending from the robots represent the chosen command followed by a maximum rate stop.

and deceleration of $6m/s^2$. For tests, four robots were given the task of traveling from left to right and back again four times, with each robot having separate goal points separated in the y axis. Because different robots have different path lengths to travel, after a two traversals robots start interacting while trying to move in opposed directions. Figure 7 shows an example situation. The four full traversals of all the robots took about 30 seconds 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 modelled here for simplicity; These errors depend heavily on the specifics of the robot and lack a widely applicable model. First, we compared using planner and motion control but enabling or disabling the safety search. Each data point is the average of 40 runs, representing about 20 minutes of simulated run time. Figure 8 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. Next, we evaluated the safety search using different margins of $1 - 4mm$ around the $90mm$ robots, plotted against increasing vision error (Figure 9. 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. As far as running times, the realtime operation of ERRT has been maintained, with a mean time of execution is $0.70ms$ without the velocity safety search and $0.76ms$ with it. These reflect the fact that the planning problem is usually easy, and interaction with other robots that require sampling are rare. Focusing on the longer times, the 95% percentiles are $1.96ms$ and $2.04ms$, respectively.

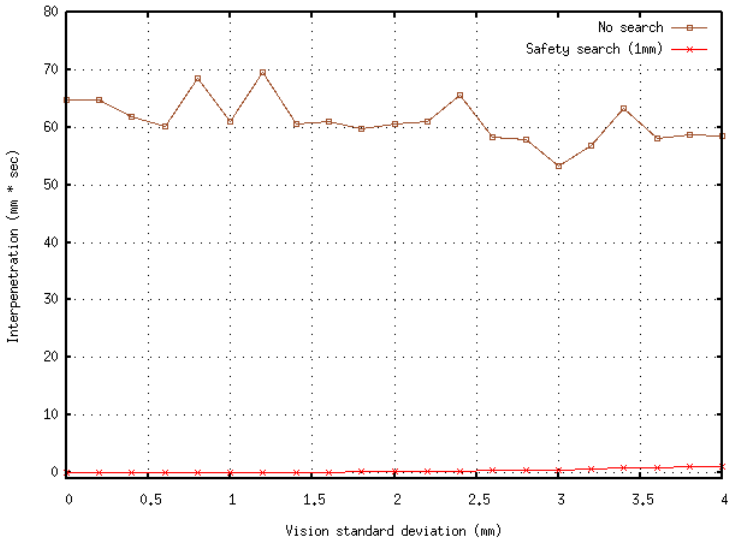

Figure 8. Comparison of navigation with and without safety search. Safety search significantly decreases the metric of interpenetration depth multiplied by time of interpenetration.

6. Conclusion

This paper described a navigation system for the real-time control of multiple high performance robots in dynamic domains. Specifically, it addressed the issue of multiple robots operating safely without collisions in a domain with acceleration and velocity constraints. While the current solution is centralized, it does not rely on complex interactions and would rely on minimal communication of relative positions, velocities, and actions to distribute the algorithm. Like most navigation systems however, it makes demands of sensor accuracy that may not yet be practical for multiple distributed robots with only local sensing. Its primary

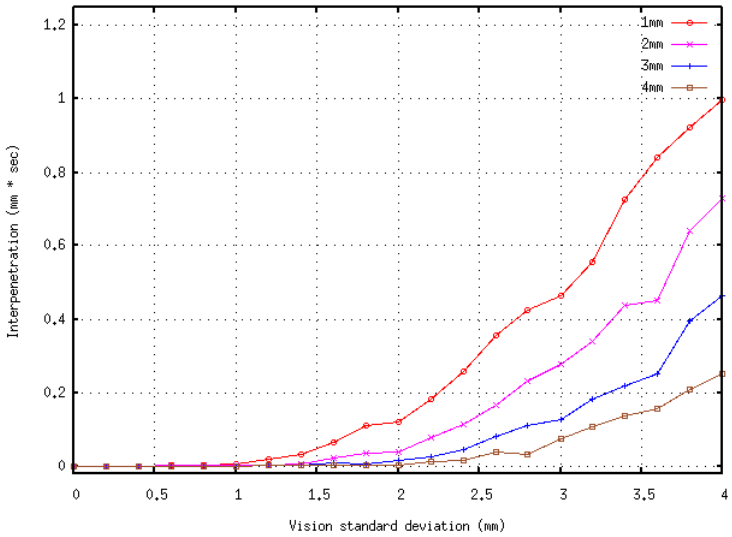

Figure 9. Comparison of several margins under increasing vision error.

contribution is to serve as an example and model of a complete system for similar problem domains.

References

Bowling, M. and Veloso, M. (1999). Motion control in dynamic multi-robot environments. In *International Symposium on Computational Intelligence in Robotics and Automation (CIRA'99)*.

Brock, O. and Khatib, O. (1999). High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Bruce, J., Bowling, M., Browning, B., and Veloso, M. (2003). Multi-robot team response to a multi-robot opponent team. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Bruce, J. and Veloso, M. (2002). Real-time randomized path planning for robot navigation. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems (IROS)*.

Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1995). Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife*.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. In *Technical Report No. 98-11*.

LaValle, S. M. and James J. Kuffner, J. (2001). Randomized kinodynamic planning. In *International Journal of Robotics Research, Vol. 20, No. 5*, pages 378–400.