Fast Goal Navigation with Obstacle Avoidance using a Dynamic Local Visual Model

Juan Fasola, Paul E. Rybski, and Manuela M. Veloso

School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA, 15213
jfasola@andrew.cmu.edu,{prybski,mmv}@cs.cmu.edu

Abstract. We introduce an algorithm for navigation to a goal with obstacle avoidance for the Sony AIBO mobile robot. The algorithm makes use of the robot's single monocular camera for both localization and obstacle detection. The algorithm alternates between two different navigation modes. When the area in front of the robot is unobstructed, the robot navigates straight towards the goal. When the path is obstructed, the robot will follow the contours of the obstacles until the way is clear. We show how the algorithm operates in several different experimental environments and provide an analysis of its performance.

1 Introduction

Navigating to goals while avoiding both static and dynamic obstacles is a challenging problem for a mobile robot. This problem is even more difficult when the robot is unable to generate an accurate global models the obstacles in its environment. Determining an optimal navigation policy without this information can be difficult or impossible. If placed in such a situation, a robot will have to rely on local sensor information and navigation heuristics to direct it from one location to another. The quality of this sensor information is extremely important as well. Poor sensor and odometry estimates will greatly compound the errors in the robot's freespace estimates and will make navigation decisions very difficult. We are interested in developing global navigation algorithms for robots with these perceptual limitations.

In the RoboCup [1] domain, teams of robots play soccer against one another. The goal behind this annual competition is to encourage research in the areas of artificial intelligence and robotics. Robots must contend with both static obstacles such as wells as well as dynamic obstacles (such as other robots) as they attempt to put the ball into the opponent's goal. In the RoboCup legged league [2], the only robots that are allowed are Sony AIBOs. These robots are equipped with a single monocular camera which serves as their only exteroceptive sensor. This paper describes a technique by which an AIBO robot can visually navigate to globally-defined goal points on the Soccer field while avoiding obstacles.

Our approach to the problem of navigating to goal points is a two step process. When the robot's path is unobstructed, it navigates straight towards the goal, using its global position estimate to guide it. When the robot encounters an obstacle, it follows the contours of the obstacles to avoid making contact with them while still attempting to make forward progress towards the goal. This algorithm evaluates the robot's position in relation to the obstacles and goal and determines whether it should continue following the obstacle or whether it is safe to walk directly towards the goal. Because of the uncertainty in the robot's position and the difficulty of determining whether an obstacle is static or dynamic, this algorithm does not involve any form of global memory of the robot's position. This means that in some pathological situations, the robot may return to the same location in its pursuit of the desired goal. A degree of randomness is used to help perturb the robot out of these kinds of situations.

It is interesting to note that even though the algorithm is not necessarily complete, such as in cases where the random aspects fail to jostle the robot out of a trap, the algorithm is successful in the majority of the cases that present themselves to the robot while in execution. To compute a globally consistent map of its environment that will allow the robot to compute a globally optimal path to its goal would likely require a great deal more computation [3] and time than AIBOs will have in a typical soccer game. Striking a balance between computing highly accurate maps and maintaining a rapid response time is a challenge faced by all competitors in the RoboCup domain. Finally, while our research focuses on algorithms that will be used for RoboCup, the techniques described in this paper can be used outside of the soccer arena in any environment where robots need to navigate to a goal but cannot compute a globally optimal plan due to limited or noisy sensor information.

2 Related Work

Many different methods for obstacle avoidance have been proposed, ranging from completely reactive behavior-based schemes [4] to more deliberative systems that construct maps and plan over freespace [5].

One method is motor schemas [6], which uses a method similar to the attractors and repellers found in potential fields approaches to direct a robot's motion. In this approach, several different navigation vectors are computed and the sum of their velocities at any given point in the environment describes the robot's current motion. In our approach, the algorithm either heads towards the goal, or follows the contours of obstacles. In either case, there is no blending of multiple control policies at any point.

The notion of alternating between goal pursuing and obstacle avoidance was illustrated by the TangentBug/WedgeBug algorithms [7]. In these algorithms, the robots are assumed to have accurate information about the distances to obstacles from sensors such as stereo cameras or laser range finders. Additionally, the range of the sensors is assumed to be much larger than what we have available on the AIBOs.

MORE!!!

3 The Robot Platform

The robots used in this research are the commercially-available AIBOs, as shown in Figure 1, created by the Sony Digital Creatures Laboratory. The robots are fully autonomous, with a 384MHz MIPS processor, visual perception, and sophisticated head, leg, and tail motions. Information returned from the sensors includes temperature, infrared distance, 3-axis acceleration, and touch (buttons on the head, the back, chin, and legs). The robot has twenty degrees of freedom including the mouth (1 DOF), head (3 DOF), legs (3 DOF x 4 legs), ears (1 DOF x 2 ears), and tail (2 DOF). The program storage medium is a 32M memory stick.

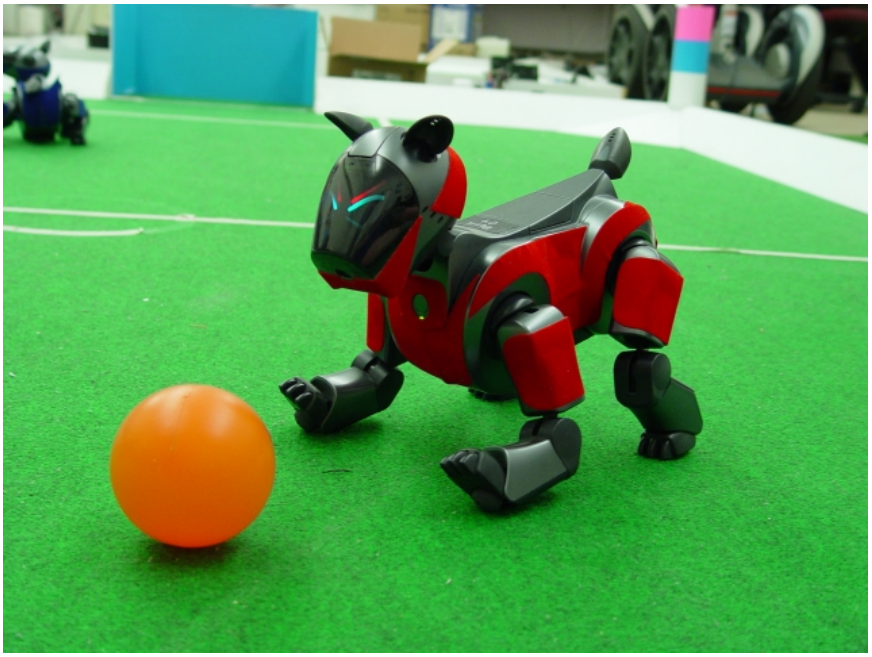
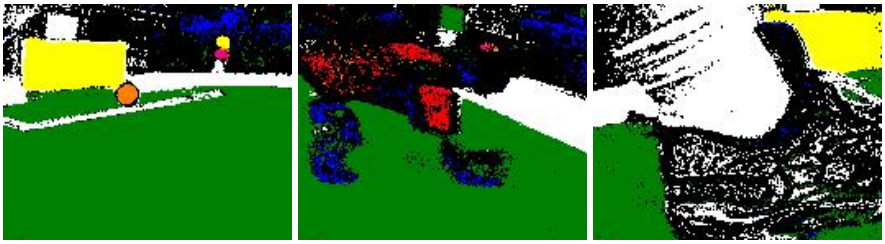


Fig. 1. Sony AIBO ERS-210 with a soccer ball.

3.1 Vision

The AIBO's primary exteroceptive sensor is a color CCD camera mounted in its nose. The pixels in the images are classified into semantically meaningful groups using CMVision [8], a fast color segmentation algorithm. Some color classes that

the robot is aware of includes the floor, the soccer ball, other robots, and walls on the field. Any color pixel that is not in the list is classified as unknown. Figure 2 shows sample images segmented by the robot.



Yellow goal and ball | Sony in red uniform | The programmer's shoe

Fig. 2. Sample color segmented images.

4 Local Environmental Model

Two different environmental modeling systems are used for this algorithm. The first a local obstacle model which uses readings from the robot's sensors to determine the distances of the nearest obstacles in any given direction. The second method is a global localization scheme which uses markers on the field to determine the location of the robot.

4.1 Obstacle Modeling

All decisions as to whether the area in front of the robot is free are made by analyzing the images returned by the camera with an algorithm called the visual sonar [9]. As its name suggests, visual sonar detects obstacles in the environment and calculates the distances from the robot to those obstacles based on the height and angle of the robot's camera, as illustrated in Figure 3.

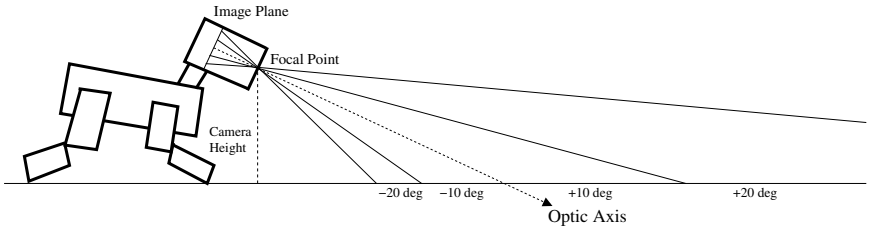
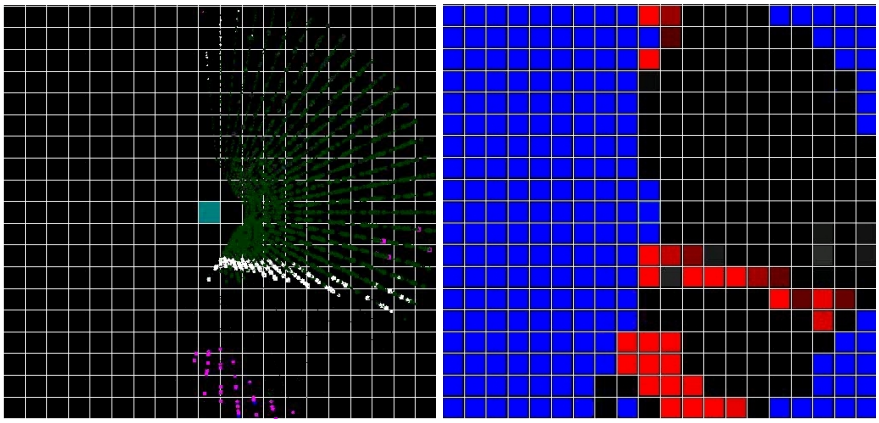


Fig. 3. Measuring distance with the AIBO's camera.

The locations of the open areas and the obstacles are all stored in an ego-centric local model. The data stored in this local model depends a great deal on the accuracy of the vision information. Figure 4 illustrates how obstacles and freespace appear to the robot. Contours of obstacles and freespace in the environment are stored in the model as collections of points that shift around the robot as it moves. Because of the uncertainty in the robot's position, the points are forgotten after a certain length of time due to the accumulated errors in the robot's position. The stored points are used to generate an occupancy grid [10], a probabilistic representation of free space.



Obstacles and freespace are represented as samples
green = field
non-green=obstacles

Occupancy grid generated from the samples
black=freespace
red=obstacles
blue=unknown

Fig. 4. Graphical representation of the ego-centric local model.

4.2 Robot Localization

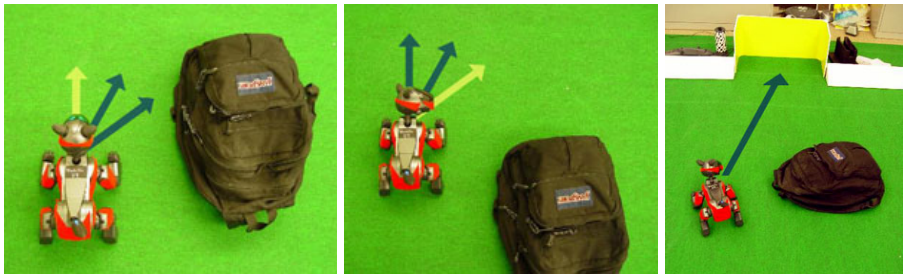
In order for the robot to determine the locations of goal positions, a global localization scheme using a particle filter [11] is employed. The particle filter is not used to track the positions of obstacles because the visual sonar does not return an accurate enough estimate of the shape of the obstacle. In addition, the drift associated with the odometry and localization uncertainty makes it difficult to correlate the local readings on a global scale.

The robot's goal points are stored in a global coordinate frame. A set of six unique markers are placed around the perimeter of the field and are used as landmarks for localization. The robot must occasionally look around to determine the positions of these landmarks so that it can localize itself.

5 Obstacle Avoidance Algorithm

Because of the AIBO's proximity to the ground, the error of the visual sonar increases significantly with distance. Anything past 2 m cannot reasonably be measured in this fashion. As a result, all of the navigation decisions must be made from very local information. The algorithm only considers obstacles that are at most 0.6 m away from it.

At a high-level, the algorithm switches between a goal-navigation mode and an obstacle-following mode. In the goal-navigation mode, the robot has not encountered an obstacle directly in front of it and moves toward the global coordinate that is the goal. In obstacle-following mode, the robot follows the contours of an obstacle that it has encountered in an attempt to move around it.



(a) Contour following (b) End of contour reached (c) Goal pursuing

Fig. 5. A high-level description of the algorithm. The robot follows the contours of an obstacle (a) until it has reached the end of it (b) and can move towards the goal (c).

Figure 6 shows the algorithm's finite state machine. The individual states of the algorithm are described below:

Localize: Stop the robot for 4 s, look at the various goal markers, and compute an estimate for the robot's location. Gathering readings for this duration allows multiple landmark readings to be taken which improves the localization accuracy. Additionally, standing still avoids unnecessary head motions that may introduce further error into the localization estimate.

Walk to Goal: Check to see if the robot has localized in the last 8 s. If not, switch to the **Localize** state to obtain a good estimate for the robot's position, and then transition back to the **Walk to Goal state**. Once localized, move the robot directly towards the goal location. If an obstacle is encountered, transition to the **Obstacle in Front** state.

Obstacle in Front: Gather local model information on right and left sides for 1.5 s each. Choose the direction that is the most open (choosing randomly if both are equally open) and transition to the **Turn in Place** state, followed by the **Contour Follow** state. If both sides contain obstacles, transition to **Turn in Place** and then back to **Obstacle in Front** to get a new perspective on the robot's surroundings.

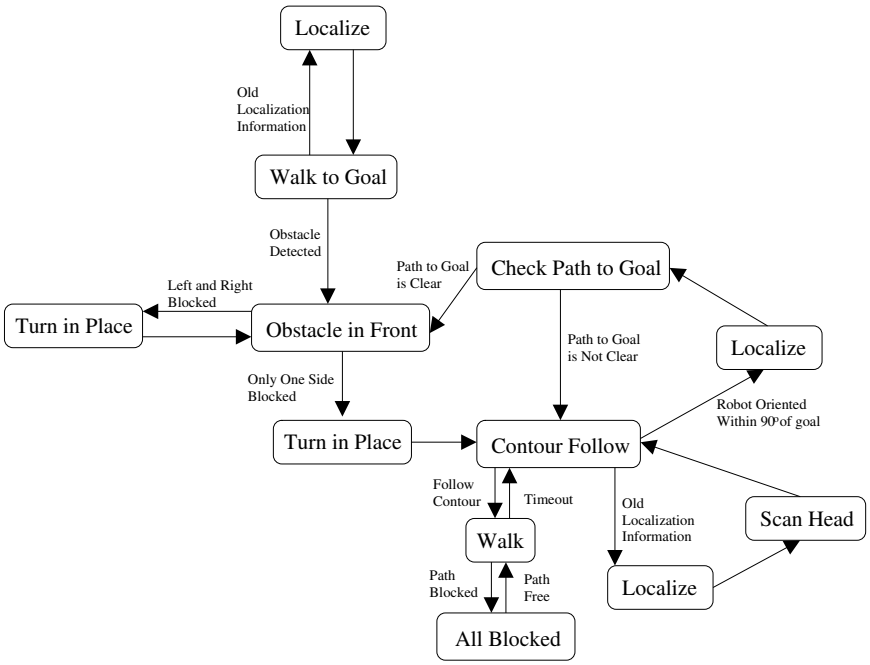


Fig. 6. Finite state machine description of the navigation algorithm. The robot starts out in the **Walk to Goal** state. States such as **Localize** and **Turn in Place** may transition to multiple different states depending on the situation and so these states are duplicated in the figure for the sake of clarity.

Turn in Place: Rotate the robot in the specified direction for 1.5 s (roughly corresponding to 90 degrees).

Contour Follow: If the robot hasn't localized in the last 20 s, transition to the **Localize** state and then go to the **Scan Head** state. Otherwise, use the local model to choose a direction vector to travel towards. If the robot is oriented roughly within 90^o of the goal, query the local model, and then transition to **Localize** and then to **Check Path to Goal** if the path is open. If none of these are true, transition to **Walk** with the direction vector that will have the robot follow the contour.

Scan Head: Stand still and scan the obstacle with the camera for 2 s to fill the local model with data. Once done, transition back to **Contour Follow**.

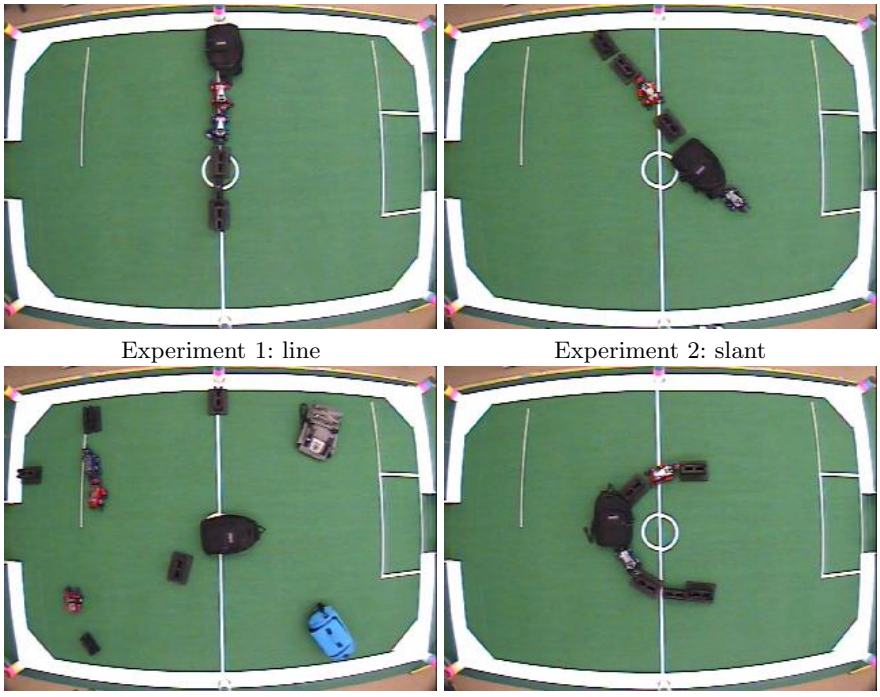
Walk: Walk along the obstacle contour for 300 ms and then transition back to **Contour Follow**. If all directions in front of the robot are blocked, and have been blocked for longer than 1.5 s, transition to the **All Blocked** state.

Check Path to Goal: Look along the vector from the robot to the goal. If the path is open, transition to **Walk to Goal**. If path is not open, transition back to **Contour Follow**.

All Blocked: Turn the robot's head to 60^o on the opposite side of the obstacle to see if the path is open. If so, set walk vector and transition to **Walk**. Otherwise, continue to rotate in place.

6 Experimental Results

To evaluate the ability of this algorithm to navigate around obstacles of various types, several experiments were run in different environmental setups. For each of the experiments, the robot started out at one end of the field and worked its way to the other end. An overhead camera tracked the robot's position and recorded the starting and ending times for each of the runs.



Experiment 1: line Experiment 2: slant

Experiment 3: spread Experiment 4: concave

Fig. 7. The four experimental environments used in the paper.

The four different environments are shown in Figure 7. The first experimental environment was a straight line of obstacles that stretched across the middle of the field. The second environment was similar to the first environment, but instead of having the line stretch straight across the field, the line slanted towards the robot's starting point and created a concave trap with only one way around. The third experiment consisted of a series of obstacles that were spread uniformly

around the field. The fourth experiment had a concave obstacle directly in the center of the field with open paths to the left and right of it.

Ten different trials were run for each experimental setup. The robot's position in the field was were recorded from an overhead camera. This was also used to record the time it took to reach the goal from the starting location. The means and standard deviations across each of the experiments is shown in Table 1.

Experimental setup	Mean (seconds)	Standard Deviation (seconds)
Line	91.32	31.46
Slant	117.57	59.51
Spread	76.65	32.31
Concave	65.38	12.09

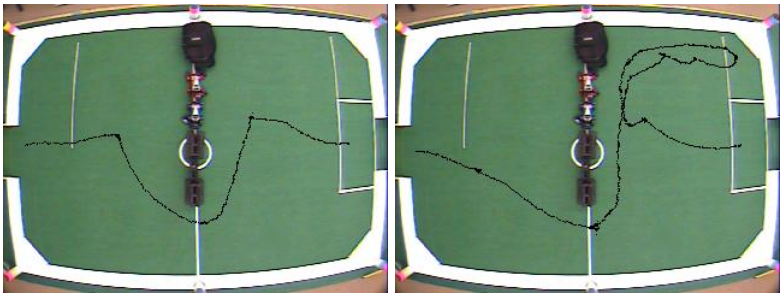
Table 1. Means and standard deviations from each of the 10 different experimental environments.

In order to provide a better description of how the algorithm operates, two individual trials from each of the four experimental environments is shown in Figure 8. These figures were chosen to try to illustrate some of the different ways that the algorithm operated in those environments.

In the Figure 8(a), the robot starts off by walking towards the goal and then stops once it reaches the line obstacle. After determining that the left and right sides are unblocked, the robot randomly chooses to turn to the left and starts to follow the contour of the obstacle until the end of the obstacle is reached. Once the robot moves around the obstacle, it localizes itself and determines that the goal is to its left. Seeing that it no longer needs to follow the contour, it walks towards the goal. In Figure 8(b), the robot decided to explore the right side of the obstacle instead. It reaches the end of the obstacle and starts following the contour of the wall. Eventually, the robot turns towards the goal, which causes it to follow the contour of the obstacle again until it is able to move past it and continue on to the goal.

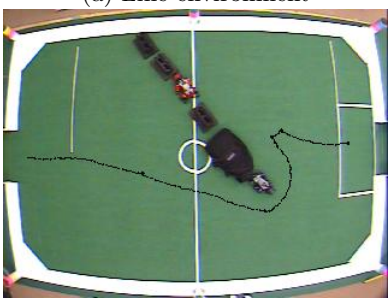
The slant environment differs from the line environment in the sense that when the robot encounters obstacle, it is more likely to find that the left side contains obstacles and and the right is free. This typically causes the robot to turn right and spend more time following the contour of the obstacle and the wall until it is able to turn around and reach the opening, as shown in Figure 8(d). The increased likelihood of turning the wrong direction resulted in this experiment to have the highest mean completion time (it also had the highest variance since once the robot became trapped, it would tend to stay trapped).

In the spread environment, the obstacles were arranged more uniformly across the field. This created more than one path for the robot to explore. Though there are more obstacles that the robot is forced to avoid its decisions on which side to turn to do not affect it as much as in the previous environments. Therefore, the mean completion time of the trials is less than in the two line obstacle

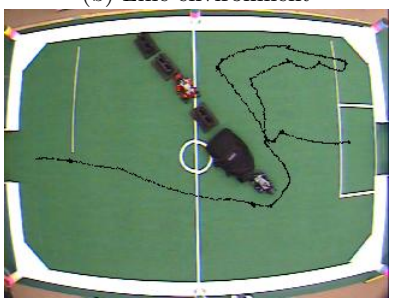


(a) Line environment

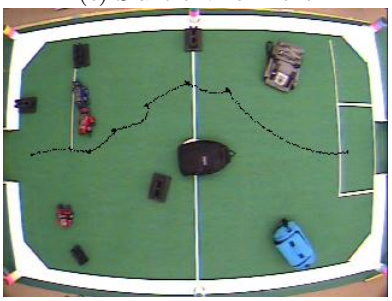
(b) Line environment



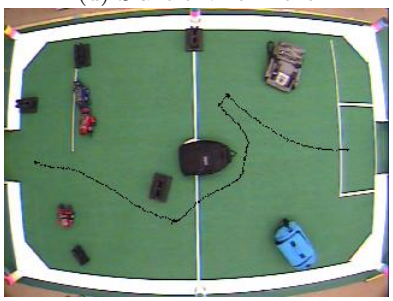
(c) Slant environment



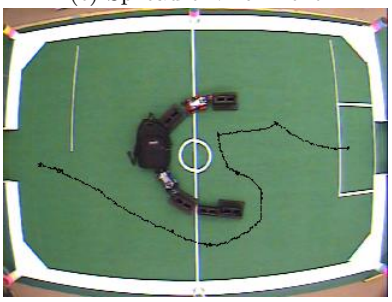
(d) Slant environment



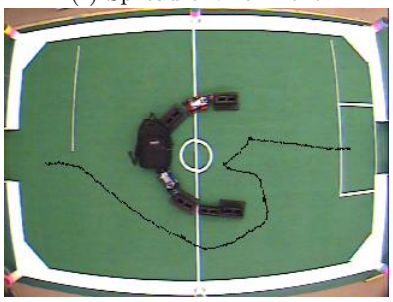
(e) Spread environment



(f) Spread environment



(g) Concave environment



(h) Concave environment

Fig. 8. Example robot paths in each of the different experiments as captured by the overhead camera.

environments. However, as can be seen in Figure 8(f), the robot would still decide to take the long way around obstacles. The angle towards which the robot approached the center obstacle still determined which direction it took, regardless of which way around was shorter.

The mean completion time for trial runs in the concave environment, along with the standard deviation time is the lowest of all the environments tested. The reason for this is that no matter at what angle the robot detects the concave obstacle, and no matter what side it chooses to explore, there is a minimal amount of contour following that it that it must do before finding an open path directly towards the goal. The robot can also see far enough to notice that the obstacle is concave and that there is no reason for it to go off and explore the inner contours of the obstacle.

7 Summary & Future Work

We have presented an algorithm that allows a mobile robot with a monocular camera system and noisy odometric estimates to navigate to a goal while avoiding obstacles. We have shown that the described algorithm works well in a variety of different environments. However, there are several areas where this algorithm could be improved.

First, the algorithm makes no attempt to look further along the robot's path to determine whether there may be obstacles that should be avoided before it reaches them. Instead, the robot navigates directly towards the obstacle and switches into contour-following mode to get around it. This works fine for large obstacles, but for small obstacles, the mode switching forces the robot to spend more time navigating than it needs to. By detecting the presence of obstacles further down the path, and making an attempt to move past them, the robot could spend less time in the contour-following mode.

Finally, because the robot keeps no memory of its history, it is still possible for it to become caught in infinite loops where the environment is particularly pathological. Incorporating some notion of where the robot entered into the contour-following state, and which direction was chosen at that point, could be useful to help the robot avoid the same situation over and over again. Further experimentation is necessary to determine how useful this would be.

Acknowledgement

Thanks to Scott Lenser, Jim Bruce, and Doug Vail for all of their help and support with the AIBOs. *Who do we acknowledge for funding?*

References

1. Veloso, M., Pagello, E., Kitano, H., eds.: RoboCup-99: Robot Soccer World Cup III. Springer-Verlag Press, Berlin (2000)

2. Lenser, S., Bruce, J., Veloso, M.: CMPack: A complete software system for autonomous legged soccer robots. In: Proceedings of the Fifth International Conference on Autonomous Agents. (2001) Best Paper Award in the Software Prototypes Track, Honorary Mention.
3. Simmons, R., Koenig, S.: Probabilistic robot navigation in partially observable environments. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, San mateo, CA, Morgan Kaufmann (1995) 1080–1087
4. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation **RA-2** (1986) 14–23
5. Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N., Schulte, J., Schulz, D.: MINERVA: A second generation mobile tour-guide robot. In: Proceedings of the IEEE International Conference on Robotics and Automation. (1999) 1999–2005
6. Arkin, R.C.: Motor schema-based robot navigation. International Journal of Robotics Research **8** (1989) 92–112
7. Laubach, S.L., Burdick, J.W.: An autonomous sensor-based path-planner for planetary microrovers. In: Proceedings of the IEEE International Conference on Robotics and Automation. (1999) 347 – 354
8. Bruce, J., Veloso, M.: Fast and accurate vision-based pattern detection and identification. In: Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation, Taiwan (2003, to appear)
9. Lenser, S., Veloso, M.: Visual sonar: Fast obstacle avoidance using monocular vision. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, Nevada (2003) 886–891
10. Elfes, A.: Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University (1989)
11. Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust monte carlo localization for mobile robots. Artificial Intelligence **101** (2000) 99–141