FAST GOAL NAVIGATION WITH OBSTACLE AVOIDANCE USING A DYNAMIC LOCAL VISUAL MODEL

Juan Fasola[*]
jfasola@andrew.cmu.edu

Paul E. Rybski[*]
prybski@cs.cmu.edu

Manuela M. Veloso[*]
veloso@cs.cmu.edu

[*]School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA, 15213

ABSTRACT

We introduce an algorithm for navigating to a goal while avoiding obstacles for an autonomous robot, in particular the Sony AIBO robot. The algorithm makes use of the robot's single monocular camera for both localization and obstacle detection. The algorithm builds upon a novel method for representing freespace around the robot that was previously developed for use on the AIBO robot. 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 follows 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.

KEYWORDS: Mobile Robotics, Navigation and Self-Localization.

1 INTRODUCTION

Navigating to goals while avoiding obstacles is a challenging problem for a mobile robot. This problem is even more difficult when the robot is unable to generate accurate global models of 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 (Veloso et al., 2000) 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. In the RoboCup legged league (Lenser et al., 2001), 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. In the 2003 RoboCup competition, one of the challenge competitions was to have an AIBO navigate from one side of the field to the other while not hitting any obstacles. Our algorithm was developed in response to this challenge.

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 obstacles, 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. The algorithm includes a degree of randomness to help perturb the robot out of these kinds of situations.

The algorithm is very careful not to have the robot collide with static obstacles. While the algorithm is not guaranteed to find an optimal path to the goal, because the robot's perceptual field is very limited, the random aspects of the algorithm provide enough disturbances to jostle the robot out of potential obstacle traps. 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 computational (Simmons e Koenig, 1995) power than is reasonable to expect with an AIBO. 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 can 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 (Brooks, 1986) to more deliberative systems that construct maps and plan over freespace (Thrun et al., 1999). Our method falls somewhere in between those two extremes. One method that is similar to ours is motor schemas (Arkin, 1989), which uses a method similar to the attracting and repelling forces 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. Another class of methods that is similar in flavor to ours is the TangentBug/WedgeBug algorithms (Laubach e Burdick, 1999) algorithm. This algorithm also uses the notion of alternating between goal pursuing and obstacle avoidance. 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.

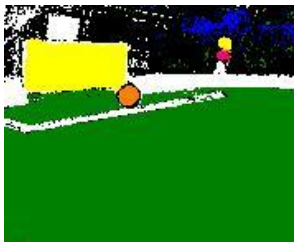
3 THE ROBOT PLATFORM

The robots used in this research are the commercially-available AIBOs, 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.

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 2 (Bruce e Veloso, 2003), 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.

Figure 1: Sony AIBO ERS-210 with a soccer ball.



Yellow goal and ball

AIBO in red uniform

Figure 2: Sample color segmented images.

4 LOCAL ENVIRONMENTAL MODEL

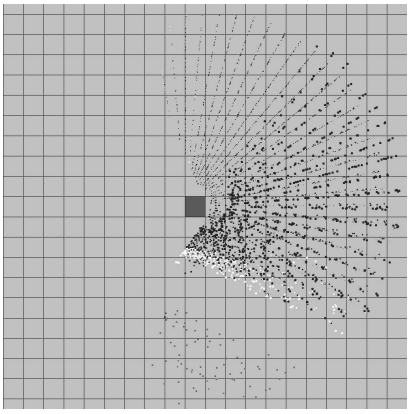
Two different environmental modeling systems are used for this algorithm. The first is a local obstacle model which uses readings from the robot's sensors to determine the distances of the nearest obstacles in any given direction. The second method is a global localization scheme which uses markers on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

4.1 Obstacle Modeling

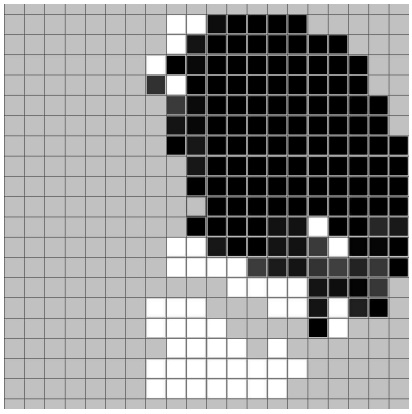
All decisions as to whether the area in front of the robot is free are made by analyzing the segmented images with an algorithm called the visual sonar (Lenser e Veloso, 2003). As its name suggests, visual sonar detects obstacles in the environment and calculates the distances from the robot to those obstacles based on the height and angle of the robot's camera.

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. The AIBO's vision system semantically labels each colored pixel as belonging to a class of freespace of obstacles. Each frame of video is scanned at small degree increments and any freespace or obstacle colors found along those scanlines (which radiate out from the robot's center) are added to the local model as points.

As this is a model of the robot's local perceptual space, the robot is always considered to be at the center of the model. The points shift around the robot based on its own odometric model, i.e., the points translate past the robot when it is walking forward and orbit



Obstacles and freespace are
represented as samples from the samples
black = freespace
white = obstacles



Occupancy grid generated
from samples
black = freespace
white = obstacles

Figure 3: The ego-centric local obstacle model where the robot is in the center of the grid. Scanlines from the visual perceptual system are parsed for colors that are freespace and obstacles (according to the color segmentation algorithm). The scanlines are sampled and a collection of points is added to the local model's database, as shown in the figure on the left. These points have a finite life time (approximately 2 seconds) before being forgotten. These points can be turned into a more traditional grid-based occupancy grid by summing the contribution of each of the freespace and obstacle points in that grid.

the robot when the robot turns in place. Because of the uncertainty in the robot's position, the points are forgotten after a few seconds to avoid using data badly corrupted by odometric error. Figure 3 illustrates how obstacles and freespace appear to the robot.

The stored points can be used to generate an occupancy grid (Elfes, 1989) (a probabilistic representation of free space). However updating the cells of a complete occupancy grid around the robot is typically too computationally expensive to be practical. Instead of generating a complete occupancy grid with fixed grid positions, the local obstacle model can be queried with a generalized rectangle of any size and orientation around the robot. This focuses the computation exclusively on the areas of interest. We make use of this feature in the robot's obstacle avoidance behavior.

4.2 Robot Localization

In order for the robot to determine the locations of goal positions, a global localization scheme using a particle filter (Thrun et al., 2000) 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 $2\,\mathrm{m}$ cannot reasonably be measured in this fashion. As a result, all of the navigation decisions must be made from very local information. Our navigation algorithm only considers obstacles that are at most $0.6\,\mathrm{m}$ away from it.

At a high-level, the algorithm switches between a goal-navigation mode and an contour-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.

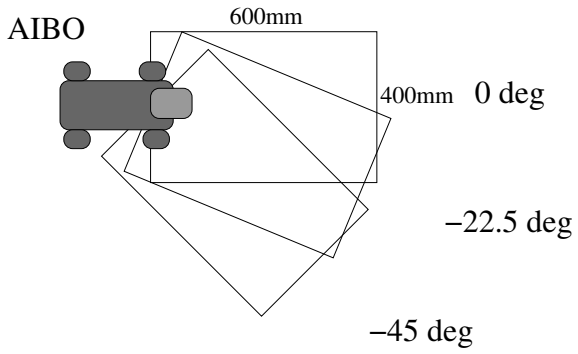


Figure 4: Generalized cells searched for obstacles when the AIBO is contour following. The size of each cell is 600mm x 400mm.

When the robot is in the contour-following state, it knows on which side the obstacle is located and therefore concentrates its head camera to that side only. In order for the robot to follow the contours of the obstacle, it evaluates three different walking angles every 300ms. The different angles are evaluated by querying the local model with a rectangle that emerges from the center of the robot in the direction of the angle being evaluated. Each rectangle is 400mm wide (across) and 600mm long (forward). The angles being evaluated are [0°,22.5°,45°], where 0° is considered the angle directly in front of the robot, and 45° is half-way towards the obstacle being followed. Figure 4 illustrates how the space in front of the AIBO is searched. Each of the three angles are first checked for the existence of obstacles,

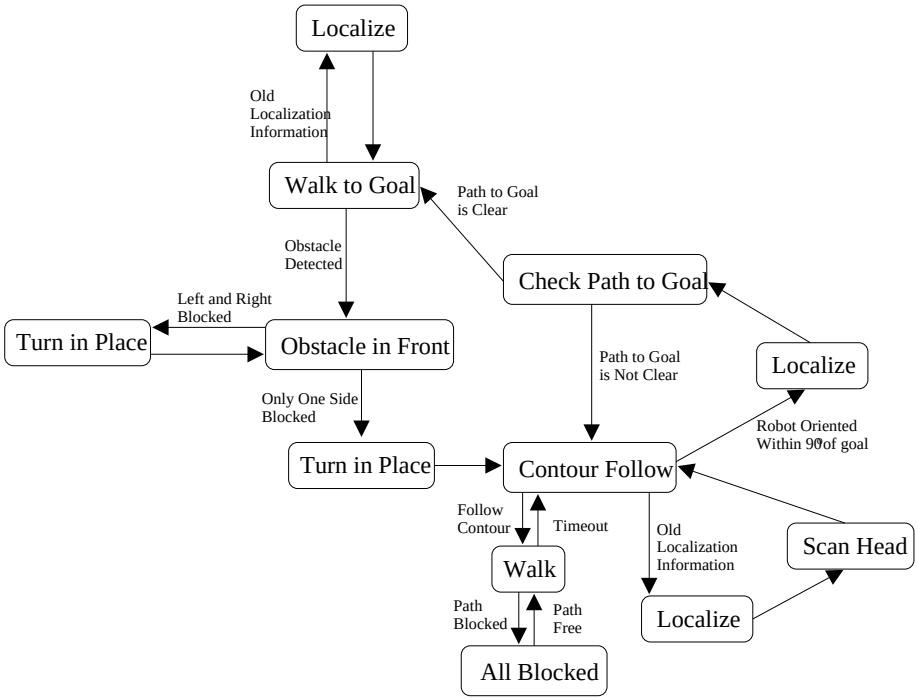
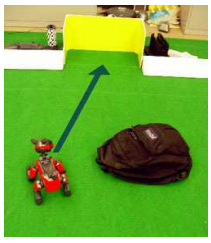


Figure 6: Finite state machine description of the navigation algorithm. The robot starts out in the **Walk to Goal** state. States such as **Localize** and **Turn in Place** may transition to multiple different states depending on the situation and so these states are duplicated in the figure for the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

(a) Contour following

(b) End of contour reached



(c) Goal pursuing

Figure 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).

those angles that are found to contain obstacles are eliminated from consideration for the walking direction. Out of the angles that are free of obstacles, the one that most closely leads the robot towards the obstacle is chosen, making 45° the angle with the highest priority, 0° with the least. Figure 5 illustrates the robot in motion.

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

- **Localize:** Halt forward motion for $4\,\mathrm{s}$, look at the various goal markers, and compute a localization estimate. Pausing for this duration allows multiple landmark readings to be taken which greatly improves the localization accuracy from a single reading. Additionally, standing still avoids unnecessary head motions that may introduce further error into the localization estimate.

- **Walk to Goal:** Check to see if a localization estimate has been taken in the last $8\,\mathrm{s}$. If not, switch to the **Localize** state to obtain a localization estimate, and then transition back to the **Walk to Goal state**. Once localized, move 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 both the right and left sides for $1.5\,\mathrm{s}$ 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 surroundings.

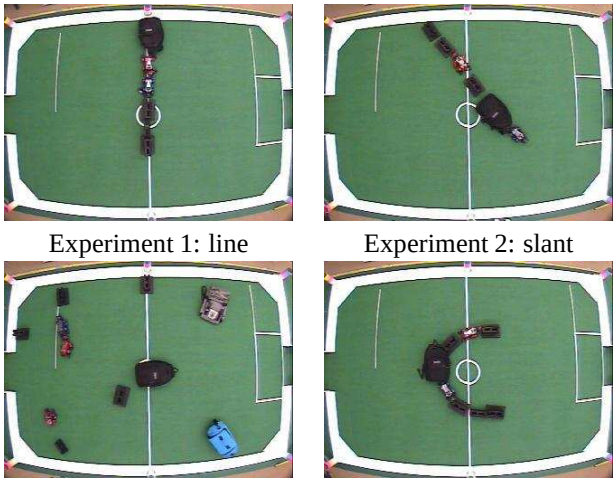
- **Turn in Place:** Rotate in place in the specified direction for $1.5\,\mathrm{s}$ (roughly corresponding to a 90 turn).

- **Contour Follow:** If a localization estimate hasn't been taken in the last $20\,\mathrm{s}$, 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. If the robot is oriented roughly within 90^o of the goal, query the local model to see if the way is clear, 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 $2\,\mathrm{s}$ to fill the local model with current data and then transition to **Contour Follow**.

- **Walk:** Walk along the obstacle contour for $300\,\mathrm{ms}$ and then transition back to **Contour Follow**. If all visible directions are blocked, and have been blocked for longer than $1.5\,\mathrm{s}$, transition to the **All Blocked** state.
- **Check Path to Goal:** Look towards the goal direction. If the path is open, transition to **Walk to Goal**. If the 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 the walk vector and transition to **Walk**. Otherwise, continue to rotate in place.

6 EXPERIMENTAL RESULTS



Experiment 1: line Experiment 2: slant

Experiment 3: spread Experiment 4: concave

Figure 7: Top-down view of the four experimental environments used in the paper. The robot started from the right side of the field (as seen from this overhead view) and had to walk to the left.

To evaluate how well this algorithm can 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.

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 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.

In order to provide a better description of how the algorithm operates, two individual trials from each of the four experimental environments are shown in Figure 8. These figures were chosen

Experimental setup	Mean (seconds)	Std Dev (seconds)	Max	Min
Line	91.32	31.46	124.83	43.92
Slant	117.57	59.51	91.31	53.68
Spread	76.65	32.31	242.01	44.65
Concave	65.38	12.09	141.76	46.00

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

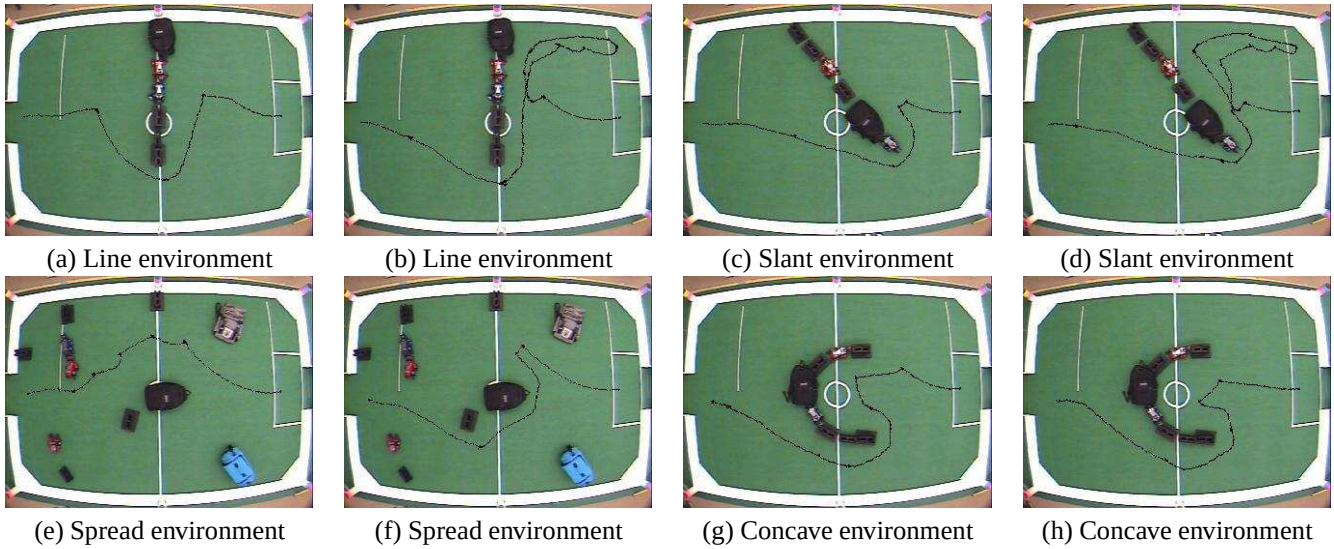
to try to illustrate some of the different ways that the algorithm operated in those environments.

In 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. The left direction was chosen without knowing that there was an opening there. The sensors could not see the open area. 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. This decision point was chosen randomly because the robot's sensors could not see the opening to the left. The robot 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 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 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



(a) Line environment (b) Line environment (c) Slant environment (d) Slant environment

(e) Spread environment (f) Spread environment (g) Concave environment (h) Concave environment

Figure 8: Example robot paths in each of the different experiments. The robot started on the right side of the field (as seen from overhead) and moved to the left, following the black lines superimposed on the images. The robots were automatically tracked by the overhead camera which captured their paths across the field. The AIBOs did all of their own local vision processing and did not have access to the overhead video information.

it to go off and explore the inner contours of the obstacle.

7 CONCLUSION

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.

The robot navigates towards the goal in open space and directly towards the obstacle switching into contour-following mode to carefully get around it. The robot uses a local vision model to detect the obstacles close to it. The efficiency of the contour-following can be adapted to different timing requirements of the task by increasing the lookahead for obstacles. If facing a pathological environment, the algorithm can be easily extended to include detection of possible state loops.

We have shown results from our fully implemented and well used algorithm in a variety of different environments with challenging patterns of obstacles, which the robot successfully and effectively handles.

ACKNOWLEDGEMENTS

Thanks to Sony for providing the robust and autonomous AIBO robots. Thanks also to all the CMPack robot soccer team, in particular to Scott Lenser for the visual sonar local obstacle model and the particle-filter sensor-resetting localization algorithm.

REFERENCES

Arkin, R. C. (1989). Motor schema-based robot navigation, *International Journal of Robotics Research* **8**(4): 92–112.

Brooks, R. A. (1986). A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **RA-2**(1): 14–23.

Bruce, J. e Veloso, M. (2003). Fast and accurate vision-based pattern detection and identification, *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan.

Elfes, A. (1989). *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*, PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University.

Laubach, S. L. e Burdick, J. W. (1999). An autonomous sensor-based path-planner for planetary microrovers, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 347 – 354.

Lenser, S., Bruce, J. e Veloso, M. (2001). CMPack: A complete software system for autonomous legged soccer robots, *Proceedings of the Fifth International Conference on Autonomous Agents*. Best Paper Award in the Software Prototypes Track, Honorary Mention.

Lenser, S. e Veloso, M. (2003). Visual sonar: Fast obstacle avoidance using monocular vision, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, pp. 886–891.

Simmons, R. e Koenig, S. (1995). Probabilistic robot navigation in partially observable environments, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San mateo, CA, pp. 1080–1087.

Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N., Schulte, J. e Schulz, D. (1999). MINERVA: A second generation mobile tour-guide robot, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1999–2005.

Thrun, S., Fox, D., Burgard, W. e Dellaert, F. (2000). Robust monte carlo localization for mobile robots, *Artificial Intelligence* **101**: 99–141.

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