Integrating Range and Object Data for Robot Navigation *

David Franklin and **R. James Firby**
Department of Computer Science
University of Chicago
{franklin, firby}@cs.uchicago.edu

Abstract

Most sensors used for robot navigation fall into one of two broad categories: range sensors that give approximate distances to obstacles, and object-based sensors that detect and locate objects of specific types. Each type of sensor does a good job of detecting certain kinds of obstacles, but fails to detect others. A robust robot navigation system must be able to integrate data from both sources. We present an implemented system that combines information from range and object-based sensors into a single representation that is used to reliably navigate a robot through an office environment. The high level nature of the representation allows easy integration of task-specific navigation constraints. Results are given demonstrating how the system is utilized on our mobile robot, Chip.

1 Introduction

Navigating an autonomous robot can be viewed as a combination of three tasks: sensing the world, controlling the motor actuators, and doing some computation to determine how the results of the first task should affect the second. The first task involves activities like reading sonar or infrared range responses, or examining camera images to get information such as free space or the locations of specific objects. One problem with any method for accomplishing this task is that all sensing techniques are affected by noise and can easily be fooled (for example, sonar echoes or visual texture on the floor). As a result, an autonomous robot must utilize multiple sensing techniques so that the fallibility of one sensor can be compensated for by another. The second task involves reliably and precisely controlling the motion of the robot. That is to say, if the control system is told to move the robot with a specific translational or rotational velocity, it should be able to do so with reasonable accuracy.

*This work was supported in part by ARPA grant N00014-93-1-1394, ONR grant N00014-93-1-0332 and ONR grant N00014-93-1-1183.

The third task is the topic of this paper: how to use sensor data to decide how the robot should move. Many techniques have been proposed to accomplish this task and most fall into one of two classes:

- Reactive systems that maintain little or no state. Action is derived (almost) directly from the sensor readings. Examples of this include: using sonar readings to define potential fields to be summed with task specific potential fields to determine what direction to move (Arkin 1990), and using sonar readings to determine that a desired path is blocked and which way around the blocking obstacle is free (Firby, Christianson, & McDougal 1992).
- Deliberative systems that rely on *a priori* knowledge of the environment in the form of some sort of map. Sensors are used both to detect unexpected obstacles and to keep the robot's perceived position accurate. Having *a priori* knowledge of the environment allows a great deal of computation to be done in advance, but if the knowledge is incorrect or if the robot's actual position becomes sufficiently different from its perceived position, much of that computation is wasted, and in the worst case, out of date information can cause the robot to collide with an obstacle. One example of a deliberative system is (Payton 1990) where *a priori* knowledge is used to create a gradient field which is used in conjunction with a simple reactive obstacle avoidance technique to choose how to move the robot.

Neither of these approaches are adequate for what we want to do with our robot: we want our robot to be able to build and use rich representations of its environment and still be able to react to dynamic changes.

1.1 Goals for the navigation system

Our goal is to provide a means for moving our robot quickly and reliably from one point to another using

information from numerous sensors and to create a representation that can be easily utilized in accomplishing other navigation tasks such as entering doorways. The algorithm we have developed is designed to accomplish this goal and incorporate the following characteristics:

- The system should utilize multiple, redundant sensor types and averaging of sensor data when appropriate to make it less susceptable to problems caused by sensor fallibility.
- It should be easy to incorporate a wide range of *a priori* knowledge of the environment into the representation such as map information and how objects are expected to move.
- Objects being tracked should be reasoned about as objects, enabling us to use conventions like staying to the right of approaching people.
- The system should be sufficiently reactive to move safely in a non-menacing environment; one containing slowly moving objects, and subject to reasonable sensor and actuator error.

1.2 Overview of the paper

This paper describes algorithms we have created for providing a safe and reliable means of navigating our robot from point to point in a somewhat cluttered environment. The sensing systems we use are sonar, visual free-space, a trash finder, and a trash can finder. The sonar and visual free-space sensing systems update occupancy grids and the other systems update object-based representations. The navigation system utilizes information from all these systems to determine how to move the robot.

The information from all the systems is combined into a set of geometric shapes that enclose all regions that the robot should *not* travel in. These shapes are considered obstacles, and each is given a clockwise or counter-clockwise spin that determines which side they must be passed on. At any moment, at most two of the obstacles are relevant to the robot: the two obstacles that the robot will have to pass between next (Slack 1990). The positions of these two obstacles (relative to the robot) determine how the robot moves.

2 Range Sensors and Occupancy Grids

Range sensors provide the navigation system with information of the following form: "The closest obstacle in this approximate direction is at this distance." Two types of range sensors are currently used by the navigation system: sonar and visual free-space. Both, to varying degrees, output noisy and unfocused data. To overcome this, the navigation system needs to average the data over time to get more stable and accurate results.

Figure 1: Range sensor readings.

We use occupancy grids to facilitate this averaging. An occupancy grid is a data structure that represents how certain it is that a specific (x,y) coordinate is occupied by an obstacle. To do this, the world is represented as a two-dimensional array; each element of the array corresponds to a specific square on the surface of the actual world, and its value represents the certainty that there is some obstacle there. When new information about the world is received, the array is adjusted based on the nature of the information.

It is important to realize that, in the grids we use, the values in the array are not actual probabilities; the properties of our sensors and robot's environment are not known well enough to create actual probabilities for sensor responses. As a result, updating an occupancy grid consists of simply increasing the occupancy grid values in the areas that the data suggest are occupied, and decreasing values in the areas that the data suggest are unoccupied. More details about the use of occupancy grids can be found in (Borenstein & Koren 1991; Elfes 1989).

The data that a range sensor uses to update an occupancy grid describes a wedge on the floor. The wedge is defined as shown in Figure 1.

One type of range sensor used on our robot is sonar. A reading from a given sonar indicates that a cone extending out from the sensor is empty out to the distance given by the reading, and that, somewhere along the semi-spherical surface at the base of the cone, there is an obstacle. To use a sonar reading in a two-dimensional occupancy grid, it is translated into a pie-wedge shape on the floor. The accuracy of the sonar readings depends greatly on what material the obstacle is made of; the obstacle must reflect the sound wave back to the sonar sensor for it to be detected. Noise and instability occur when obstacles reflect sonar elsewhere and hence become invisible at certain angles.

Figure 2: Treating free-space as a range sensor. (a) A typical situation that the robot might find itself in. (b) An image from the robot's camera. A column of free-space is lightly shaded. (c) A bird's-eye view of the scene with the free-space column translated onto the floor.

`UpdateOGrid( g, point, r, theta )`

1. Decrease the value for each square in the interior of the pie shape defined by `point`, `r` and `theta` by `DecrementStrength`.
2. Collect all the squares along the circular edge of the pie shape into a set `S`. Set `Sum` to be the sum of the values of all array elements in `S`.
3. For each square $\mathtt{s} \in \mathtt{S}$, set:
   $\mathtt{val(s)} := \mathtt{val(s)} + \lceil \mathtt{UpdateStrength} \times \mathtt{val(s)} \div \mathtt{Sum} \rceil$
4. Clip the value for each square in `S` to fit within the range of acceptable pseudo-probabilities.

Algorithm 1: The occupancy grid update algorithm

Another range sensor is provided by a visual free-space algorithm (Horswill 1993) which divides a camera image into columns and then finds the closest obstacle in each column. Each column is then treated as an individual range sensor. Figure 2 shows how a column in a camera image is translated into a wedge on the ground. The free-space algorithm used on CHIP works by finding all the edges in a camera image, thresholding the edges to reduce noise, then searching each column from the bottom until an edge is found. Heading and distance values are then calculated by translating the image coordinates of the lowest edge in each column into floor coordinates.

2.1 Updating the occupancy grid

Using range readings to update an occupancy grid requires an algorithm to map readings into corresponding changes to the grid array. The algorithm we use focuses on the following characteristics:

- Correctness: The updating that is done should be in agreement with the sensor response: regions that the sensor indicates are free should have their values reduced and regions that the sensor indicates are occupied should have their values increased.
- Stability: When the occupancy grid accurately represents the actual world, changes to the occupancy grid as a result of new sensor readings should be minimal.
- Reactivity: Sudden changes to the world that affect the safety of the robot should quickly appear in the occupancy grid.

The algorithm that we use (detailed in Algorithm 1) performs well with regard to all of these characteristics, can be computed efficiently, and keeps the number of empirical parameters to a minimum. The algorithm looks at the two regions defined by the range reading: the interior of the wedge (thought to be empty), and the area around the curved edge of the wedge (thought to contain at least one obstacle). Step 1 decreases the values in this first region and step 3 increases at least some of the values in the second. The stability of the algorithm is achieved through the proportional increases defined by step 3. This step attempts to use the range reading to confirm and strengthen what is already thought to be true in the world. Finally, the reactivity of the algorithm is achieved due to the fact that `UpdateStrength` is independent of the distance specified by the range reading. This means that, with readings close to the robot, the occupancy grid values are increased more rapidly because the pie edge contains many fewer points.

For our implementation, we set `DecrementStrength` to be 1, `UpdateStrength` to be 10, and the range of acceptable pseudo-probabilities to be [0,9].

3 Object-based sensor representation

Our robot also uses sensing algorithms that are quite different from the range sensors described in the previous section. We use vision-based sensing algorithms that detect instances of specific classes of objects. One algorithm finds trash bins in an image using Hausdorff edge matching (Huttenlocher & Rucklidge 1992) and the other finds and identifies cups, cans and crumpled pieces of paper using statistics of connected components in an edge image (Firby *et al.* 1995). This kind of sensing is fundamentally different from the range sensors in that:

- It identifies specific types of objects (for instance, cans, cups or bins.)

- It provides no evidence for a space being unoccupied; it only says that a particular location is free of objects of the specific type it is looking for.

- The locations of the objects found can be computed quite accurately.

3.1 Maintaining object-based representations

The object-based representations used with our robot maintain lists of the objects that have been sensed and the world coordinates of where they were found. When new information comes from an object sensor, it is compared with previously sensed objects of the same type to see what objects are new and which have probably already been seen. Each type of object has a coreference distance `d` associated with it. So, if an object is sighted within a distance `d` of a previously sensed object of the same type, it is assumed that the sighting is actually of that known object. If no match is found, it is considered to be a new object and is added to the set of objects.

When a new object measurement matches an old one, the position for that object is updated using a Kalman filtering approach of averaging:

$$\mathtt{x}_i := \mathtt{c} \times \mathtt{newx} + (1 - \mathtt{c}) \times \mathtt{x}_{i-1}$$

By choosing different values for `c` (in the range [0,1]) for different types of objects, information about the characteristics of objects can be used to help track them better. For example: a piece of furniture should be expected to remain motionless and so a small value of `c` should be used, while with a very mobile object, a value of `c` near 1 should be used. This simple tracking method keeps positions stable in the face of small sensor error, allows slowly moving objects to be tracked, and compensates for accumulated odometry errors.

4 Navigation

Given the representations outlined in the previous sections, navigation becomes a task of planning a path through the obstacles and then actually following the path. To do this, an object-based representation that incorporates the information from all the individual representations is constructed. Then path planning is done by searching for a sequence of line segments that connect the robot's current location to the goal location while avoiding obstacles. Finally, as the robot is moving, it is sent pairs of points to go between. The sequence of pairs of points loosely define a path for the robot to follow.

Figure 3: A simple example of circular paths. The arcs show the bounding paths.

4.1 Navigation Templates (NaTs)

(Slack 1990) presents a method for controlling a robot as it navigates through an environment cluttered with multiple obstacles. The technique assigns a "spin" to each obstacle in the environment, specifying on which side the robot should pass [1]. A path through the environment is defined by how the obstacles are spun. These paths are sketchy, telling the robot to do things like "pass this obstacle on the left, then go between these two obstacles ..." allowing greater flexibility in the face of sensor and actuator noise.

While the robot is navigating through the environment, it need only consider at most two obstacles (called local navigation objectives) that constrain where the robot can safely move. One, the clockwise bound, is the next obstacle that the robot must pass to the left of, and the other, the counter-clockwise bound, is the next obstacle for the robot to pass to the right of. After a bound is passed, it is replaced by the next appropriate obstacle. As the robot continues from bound to bound, it follows the sketchy path defined by the obstacle spins.

4.2 Circular paths

The primary way that our method differs from Slack's work is that, due to the way our robot moves, we use circular segments (rather than straight lines) to describe the paths the robot travels on. Our robot is controlled by setting translational and rotational velocities over intervals of time, so, in order to travel on a path of line segments, the robot would have to follow a repeated cycle of stopping, rotating to align with the next segment and then moving forward to the end of that segment. This poses two problems for

[1] These are called modifier navigation templates (or m-NaTs) in Slack's research.

`OGrid->Objects( ogrids, start-pt, end-pt )`

1. Choose a region of interest. This region should contain both the current location and the desired destination. All path planning will be restricted to this region. For our robot, we choose a circle where the `start-pt` and `end-pt` are towards opposite sides of the circle.
2. Combine all the occupancy grids into a single binary occupancy grid, where 1 represents unsafe and 0 represents safe. This occupancy grid just contains the region of interest. For our robot, we mark a square as unsafe if any of the individual occupancy grids square's values are above the default value. Note: if more specific knowledge of the fallibility of the sensors is known, it can easily be incorporated into this step. For example, if one sensor is very reliable at indicating when an area is safe, the "safe" values in its occupancy grid should override the "unsafe" values in another, noisier, occupancy grid.
3. Find all convex hulls in the binary occupancy grid and store them as collections of boundary points. Each collection of boundary points is included as a single object in the final, object-based, representation.

Algorithm 2: Converting occupancy grids into a set of objects.

our robot: turning in place on our lab's floor is difficult due to a carpet, and all the starting and stopping produces a very jerky motion. Building paths out of circular segments remedies these problems. Figure 3 shows the family of circular paths that could be used to go through a set of obstacles.

4.3 Creating a single, object-based representation

A NaT-based navigation algorithm requires a representation of the geometric regions the robot is to avoid. So, as the first step in our navigation technique, we combine the relevant information from our occupancy grids and object-based representations to create a single object-based representation.

Each object in the robot's object-based representations has a simple "footprint" such as a circle that defines the region it occupies. From the occupancy grids, we can compute convex polygon footprints that contain all the unsafe regions in each occupancy grid. Convex polygons were chosen as the geometric shape representation because they provide a rich set of shapes, efficient convex hull algorithms exist, and the properties needed by the navigation algorithms (knowing whether a given line segment comes within a given distance of the object, and knowing what the heading to the left and right sides of the object are from a given point) are easily computed. The process we use to convert the occupancy grids into a set of objects is outlined in Algorithm 2.

`SetObstacleSpins( obstacles, start, end )`

1. Set the initial interval to be a segment from `start` to `end`.
2. Find the first obstacle that blocks the interval. If none is found, then the path over the interval is simply a straight line and a complete path has been found.
3. If there is an obstacle, "bend" the interval around the obstacle: choose a point to the side of the obstacle and replace the interval with the two intervals that connect the start to this point and that connect this point to the end. The interval will be bent in two ways: around either side of the obstacle.
4. For each of the two ways, perform steps 2 through 4 on each of the new intervals. This is done as a heuristic search based on an optimistic estimate of the final path length.
5. As soon as a path is completely defined (every interval has been looked at), each obstacle in `obstacles` is given a spin that is compatible with the chosen path.

Algorithm 3: Setting the spins for a set of obstacles.

4.4 Setting obstacle spins

While the robot is navigating, each object must have a spin assigned to it, specifying which way the robot should go around it. The combination of these spins specifies the path that the robot should take. The algorithm we use to set spins for the obstacles is a heuristic search through the space of sequences of straight line segments defined by legal combinations of obstacle spins (described in Algorithm 3).

4.5 Local Navigation Objectives

Once all obstacles have been assigned spins, the robot moves from one pair of local navigation objectives to the next until either the new sensor data indicates that the current path is no longer valid (prompting recomputation of the obstacle spins) or the robot reaches its goal. The process of choosing local navigation objectives is a simplification of that used by Slack with the substitution of circular segments for line segments. All obstacles that are either in front of the robot or in between the robot and the goal are sorted based on how far away they are from the robot. Then a circular path is chosen that is compatible with the largest subset of the obstacles and their spins. The local navigation objectives are the obstacles that restrict the set of compatible circular paths the most.

Actual navigation then consists of choosing a circular path that is consistent with the current local navigation objectives and following that path. The local navigation objectives are changed as the robot moves past them, or as new obstacles are sensed. In Figure 3, the two smaller circles are the local navigation objectives and any circular path between the two bold arcs would be legal. After the robot passes the lower circle, the larger circle will replace it as a local navigation objective.

The use of local navigation objectives produces several benefits for the navigation system:

- The algorithm that the robot uses to determine how to actually set the motors on the base is very simple and only requires the positions of the (two) local navigation objectives, how close it is safe to get to them, and the position of the goal. This allows the motor controller to run in a very tight loop, which means the robot drives very accurate paths.

- The low-level controllers do not have to deal with the complicated representations of the navigation system.

5 Experiments

The algorithms and data structures outlined in the previous sections have all been successfully implemented for our mobile robot, Chip. Before describing the actual experiments, we will provide some necessary information about the robot and the control and sensing systems it uses.

5.1 The robot

Chip is actually the combination of a network of computers and a physical robot. All image processing and high-level reasoning systems are implemented on external computers while computers on board are responsible for controlling the physical systems on the robot (arm, base, pan-tilt head and sonars.) Communication between on-board and off-board computers is done through a radio ethernet connection.

The robot moves via a base that is controlled by setting specific translational and rotational velocities. The robot is equipped with eight sonars, with placement biased towards the front of the robot. New sonar readings are acquired twice a second. Visual images are collected via stereo cameras mounted on a pan-tilt head and transmitted to the image-processing computers. New free-space readings are generated approximately twice a second.

5.2 Implementation details

The navigation system oscillates between two states: the first initializes and starts a set of processes that do the navigation, and the second deals with the problem situation where the robot gets too close to an obstacle and must back or turn away from it (due to sensor delay or objects moving to block the robot's path.)

The set of processes that run in the first state (basic navigation of the robot) are as follows:

`ReportSonarValues`: This process repeatedly polls the sonars and updates the sonar occupancy grid with the sonar readings.

`ReportFreeSpaceValues`: This process repeatedly calls the free-space vision routine and updates the free-space occupancy grid with the results.

`TrackSmallObjects`: This process repeatedly calls the small object finder and updates the object-based representation with the results.

`SetNavObjectives`: This process builds the composite, object-based representation, plans a path through it, and sends the local navigation objectives (based on the robot's position) to the robot.

`SetSpeed`: This process, based on the amount of empty space in front of the robot, sends a maximum safe traveling speed for the robot. In the case that the robot is too close to an obstacle, this behavior will tell the robot to stop and enter the second state.

In the state where the robot is too close to some obstacle, six paths are projected (forward or backwards, going straight or turning left or right) and whichever path will take the robot to safety quickest is selected. Then, the robot is instructed to follow that path until the robot is a safe distance from the obstacle it was too close to. Once it is free, the robot will enter the first state and continue navigation.

5.3 Experiment 1: Combining information

The first experiment demonstrates how information from three different sensing sources is combined. Chip drove through an environment consisting of two pop cans, a crushed pop can (detectable by the free-space visual routine but not the can finder), a cereal box, a pillar, and a chair. The environment was set up so that Chip could safely drive straight forward for two meters.

Chip rolled forward at 5 cm/sec for a distance of two meters while running `ReportSonarValues`, `ReportFreeSpaceValues`, and `TrackSmallObjects`. Figure 4

Figure 4: Detail of the Experiment 1. (a) Layout of the environment. (b) The final sonar occupancy grid. (c) The final free-space occupancy grid. (d) The final object-based representation overlayed on the two occupancy grids.

Figure 5: Detail of the Experiment 2. (a) Layout of the environment. (b) Early composite representation. The line indicates the path that the navigation system has chosen. (c) The final composite representation. Note that there are no occupancy grid obstacles outlined because the robot is close to the goal.

shows how the three sources of information combine to make the object-based representation. The medium gray areas towards the edges of the occupancy grids are where Chip has not done any sensing; darker areas are where Chip thinks obstacles are; lighter areas are where Chip thinks there are not any obstacles. Comparing the sonar and free-space occupancy grids reveals how in some cases the two sensing techniques are in agreement and in others how one sensing technique compensates for the weakness of another. The polygons (outlined in gray) outline the obstacles found in the occupancy grids.

5.4 Experiment 2: Navigating in a crowded environment

The second experiment demonstrates all of the components of the navigation system working together to guide the robot through an environment containing two metal beams, a chair and a pillar (as shown in Figure 5.) The robot had to drive in an S-curve to avoid the obstacles between it and its goal. The chair was positioned such that the free-space routine would fail to accurately find its footprint (the chair's base consists of two parallel rails and the chair was oriented such that there was free-space visible under the chair.)

Chip moved at 5 cm/sec, covering a distance of about 5 meters while running `ReportSonarValues` and `ReportFreeSpaceValues`. The second picture shows how the sonar and free-space occupancy grids put the chair in two different places (the free-space routine incorrectly located the chair too far to the upper right.) The sonar compensated for the free-space's fallibility in dealing with overhangs. The third picture shows the composite representation at the moment that Chip reached the goal. All the objects in the environment are represented in the occupancy grids.

The sequence of events by which the robot went from the start to the finish was as follows:

- Chip located the chair and the first beam and decided to pass them to the left (the clockwise bound was the leftmost point of the beam.)
- As Chip approached the beam and proceeded around it, it saw the second beam and decided to pass it to the right (the counter-clockwise bound was the rightmost point of the second beam.) At this moment, Chip had two local navigation objectives.
- Once Chip passed the first beam, it was able to see the full extent of the second beam and its counter-clockwise bound was updated accordingly. Also, the clockwise bound was removed because the first beam (behind Chip) was no longer an obstacle to Chip.
- Once Chip passed the second beam, the counter-clockwise bound was removed, and Chip was able to drive directly to the goal.

5.5 Experiment 3: A simulated doorway

The third experiment demonstrates how task-specific navigation constraints can be integrated into the navigation system. The robot was placed in an environment with 2 cans which were treated as the sides of a door frame. Spins were enforced on the two cans so that the robot was forced to drive between them, even though the shortest path to the goal would be to drive to the left side of both of them.

Chip moved at 10 cm/sec, covering a distance of about 5 meters while running `TrackSmallObjects`.

Figure 6: Detail of the Experiment 3. Obstacle spins were set such that the robot had to drive between the two cans.

Figure 6 shows the smooth path that CHIP traveled in navigating between the two cans.

6 Conclusion

This paper describes a technique for fusing the information gathered from a number of differing sensors into a single representation that can be used in a wide range of navigation tasks. Our technique avoids a problem faced by most multiple behavior navigation systems, where different behaviors may provide conflicting information about what is the right action to take, thereby necessitating some sort of arbitration. Since all navigation is based on a single data structure, there is no conflicting information about where to go – all the different sources are cooperating instead of competing.

Incorporating *a priori* knowledge into the representation can be done in one of two ways. Known objects can be inserted into one of the object-based representations, with the advantage that they can then be reasoned about as objects of specific types, but with the disadvantage that in situations where the knowledge is incorrect, it may be very difficult to detect and correct. A more conservative way to incorporate *a priori* knowledge is to add the "footprints" of known objects into the occupancy grids. This strongly biases the system initially, but allows for later information to over-rule the old.

Finally, outside the realm of navigation, occupancy grids can be used to locate regions of interest in which to use more expensive sensing operations. Because the occupancy grids represent spaces that are *occupied*, those regions marked occupied that are disjunct from the objects in the object-based representations are the most likely to contain new objects that the robot may be interested in.

6.1 Future work

In the experiments shown in this paper, there is no active sensing; the cameras are aimed straight ahead and are looking down in front of the robot. However, the local navigation objectives give the robot natural areas to focus much of its attention. Having the robot direct its attention to the local navigation objectives and occasionally focus on other relevant areas (looking towards the goal) would be likely to allow the robot to safely travel faster and also should allow the robot to pass closer to obstacles because their locations would be more accurate.

Also, this navigation system is intended to be used for point-to-point navigation in a relatively small (around 5 meters square) area. As a result, in a larger, more dense environment, performance will suffer due to the large number of obstacles the path planner will have to deal with. In fact, due to the use of convex polygons for representing obstacles in the occupancy grids, there are situations where the path planner may just fail (for instance, a long, narrow L-shaped hallway.)

As a result, we are currently working on incorporating the navigation system into a large-scale navigation system that can reason about larger objects such as hallways, rooms and doorways. The small-scale navigation system will function in an area centered around the robot. The large-scale navigation system will give information about locations of walls (and give them spins) to the small-scale navigation system, effectively functioning as an additional object-based sensor.

References

Arkin, R. C. 1990. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In Maes, P., ed., *Designing Autonomous Agents*. MIT Press.

Borenstein, J., and Koren, Y. 1991. The vector field histogram — fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation* 7(3):278–288.

Elfes, A. 1989. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer* 46–57.

Firby, R. J.; Kahn, R. E.; Prokopowicz, P. N.; and Swain, M. J. 1995. An architecture for vision and action. In *Fourteenth International Joint Conference on Artificial Intelligence*, 72–79.

Firby, R. J.; Christianson, D.; and McDougal, T. 1992. Fast local mapping to support navigation and object localization. In *Sensor Fusion V*. Boston, MA: SPIE.

Horswill, I. 1993. Polly: A vision-based artificial agent. In *Eleventh National Conference on Artificial Intelligence*. Washington, DC: AAAI.

Huttenlocher, D. P., and Rucklidge, W. J. 1992. A multi-resolution technique for comparing images using the haussdorf distance. Technical Report CUCS

TR #92-1321, Cornell University Department of Computer Science.

Payton, D. W. 1990. Internalized plans: A representation for action resources. In Maes, P., ed., *Designing Autonomous Agents*. MIT Press.

Slack, M. G. 1990. Situationally driven local navigation for mobile robots. Technical Report JPL Publication 90-17, Jet Propulsion Laboratory.