A Complete Navigation System for Goal Acquisition in Unknown Environments

Anthony Stentz and Martial Hebert

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Most autonomous outdoor navigation systems tested on actual robots have centered on local navigation tasks such as avoiding obstacles or following roads. Global navigation has been limited to simple wandering, path tracking, straight-line goal seeking behaviors, or executing a sequence of scripted local behaviors. These capabilities are insufficient for unstructured and unknown environments, where replanning may be needed to account for new information discovered in every sensor image. To address these problems, we have developed a complete system that integrates local and global navigation. The local system uses a scanning laser rangefinder to detect obstacles and recommend steering commands to ensure robot safety. These obstacles are passed to the global system which stores them in a map of the environment. With each addition to the map, the global system uses an incremental path planning algorithm to optimally replan the global path and recommend steering commands to reach the goal. An arbiter combines the steering recommendations to achieve the proper balance between safety and goal acquisition. This system was tested on a real robot and successfully drove it 1.4 kilometers to find a goal given no a priori map of the environment.

1.0 Introduction

Autonomous outdoor navigators have a number of uses, including planetary exploration, construction site work, mining, military reconnaissance, and hazardous waste remediation. These tasks require a mobile robot to move between points in its environment in order to transport cargo, deploy sensors, or rendezvous with other robots or equipment. The problem is complicated in environments that are unstructured and unknown. In such cases, the robot can be equipped with one or more sensors to measure the location of obstacles, locate itself within the environment, and check for hazards. By sweeping the terrain for obstacles, recording its progress through the environment, and building a map of sensed areas, the navigator can find the goal position if a path exists, even if it has no prior knowledge of the environment.

A number of systems have been developed to address this scenario in part. We limit our discussion to those systems tested on a real robot in outdoor terrain. Outdoor robots operating in expansive, unstructured environments pose new problems, since the optimal global routes can be complicated enough that simple replanning approaches are not feasible for real-time operation. The bulk of the work in outdoor navigation has centered on local navigation, that is, avoiding obstacles [1][2][3][5][7][14][17][20][27][28][31] or following roads [4][8][15][25]. The global navigation capabilities of these systems have been limited to tracking a pre-planned trajectory, wandering, maintaining a constant heading, or following a specific type of terrain feature. Other systems with global navigation components typically operate by planning a global route based on initial map information and then replanning as needed when the sensors discover an obstruction [9][10][21].

These approaches are insufficient if there is no initial map of the environment or if the environment does not exhibit coarse, global structure (e.g., a small network of routes or channels). It is possible to replan a new global trajectory from scratch for each new piece of sensor data, but in cluttered environments the sensors can detect new information almost continuously, thus precluding real-time operation. Furthermore, sensor noise and robot position error can lead to the detection of phantom obstacles and the incorrect location of obstacles, flooding the global navigator with more, and sometimes erroneous, data.

In Autonomous Robots, Volume 2, Number 2, August 1995

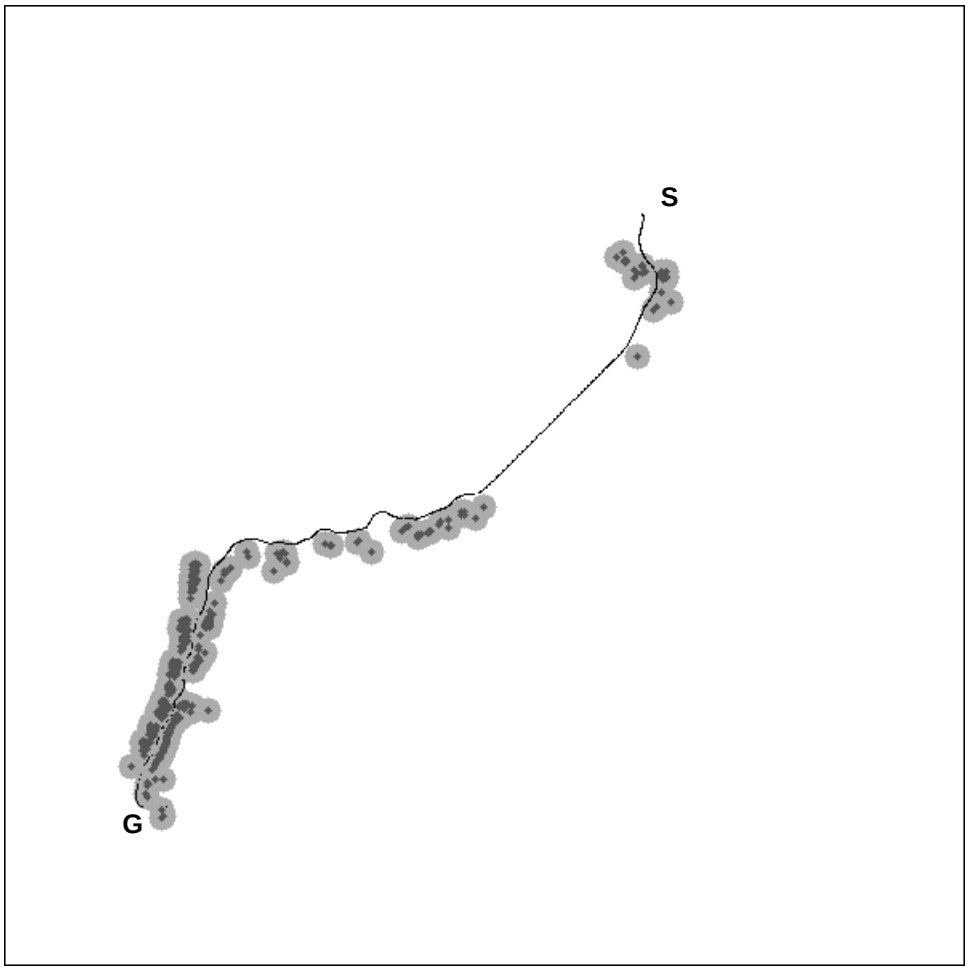
We have developed a complete navigation system that solves these problems. The system is capable of driving an outdoor mobile robot from an initial position to a goal position. It may have a prior map of the environment or no map at all. The robot is the Navigational Laboratory II (NAVLAB II) shown in Figure 1. The NAVLAB II is a high mobility multi-wheeled vehicle (HMMWV) modified for computer control of the steering function. The NAVLAB II is equipped with an Environmental Research Institute of Michigan (ERIM) scanning laser rangefinder for measuring the shape of the terrain in front of the robot. Three on-board Sun Microsystems SPARC II computers process sensor data, plan obstacle avoidance maneuvers, and calculate global paths to the goal.

In addition to perception sensors, the NAVLAB II is equipped with navigation sensors. For the experiments reported in this paper, the navigation sensors consisted of a Litton Inertial Navigation System (INS) and an odometry encoder. The readings from these sensors were integrated to provide a two-dimensional position estimate accurate to roughly 1% distance travelled and a heading estimate accurate to a few degrees per hour. The accuracy of the navigation sensors was the basis for selecting many of the parameters described later in the paper. In particular, it defines the expected amount of drift in the position of obstacles in the global map.

Figure 1: NAVLAB II Robot Testbed

Figure 2 shows the results of an actual run on the robot. The dimensions of the area are 500 x 500 meters. The robot began at the position labelled *S* and moved to the goal location at *G*. Initially, the robot assumed the world to be devoid of obstacles and moved toward the goal. Twice a second, the perception system reported the locations of obstacles detected by the sensor. Each time, the robot steered to miss the obstacles and replanned an optimal global trajectory to the goal. The robot was able to drive at approximately 2 meters per second. The robot's trajectory is shown in black, the detected obstacles in dark grey, and a safety zone around the obstacles in light grey.

Figure 2: Example of an Actual Run



This example is typical of the type of mission that we want to achieve and of the experiments reported in this paper. First, we desire to have optimal global path selection. The traverse shown in the figure is clearly not the shortest path from *S* to *G* once the entire obstacle map is known, but it is optimal in the sense that, at every point along the traverse, the robot is following an optimal path to the goal given all obstacle information known in aggregate at that point. Second, we desire to have real-time performance, even in large environments. Thus, we need to carry out experiments over long distances, typically greater than one kilometer.

For practical reasons, we make some simplifications in order to carry out our experiments. First, the goal point *G* is physically marked in the environment. We know that the goal has been reached successfully by checking that the robot has reached the marked position. The coordinates of *G* are the only input to the system at the beginning of the experiment. More realistic scenarios would call for marking the goal with a specific landmark recognizable by a vision system, but this would add complexity without contributing to the validation of our planning and perception algorithms. Second, the laser rangefinder we use for obstacle detection cannot measure the range to highly specular surfaces, such as water. This is a practical problem because we do not have independent sensing capabilities, such as polarization sensors, that can detect and measure puddles. In order to circumvent this problem, we indicated the presence of water to the system “by hand” by preventing the robot from driving through it. Although we would have preferred to demonstrate fully autonomous experiments, we believe that this problem is due to shortcomings of the current sensing modality, and it does not affect the fundamental performance of our planning and sensing algorithms.

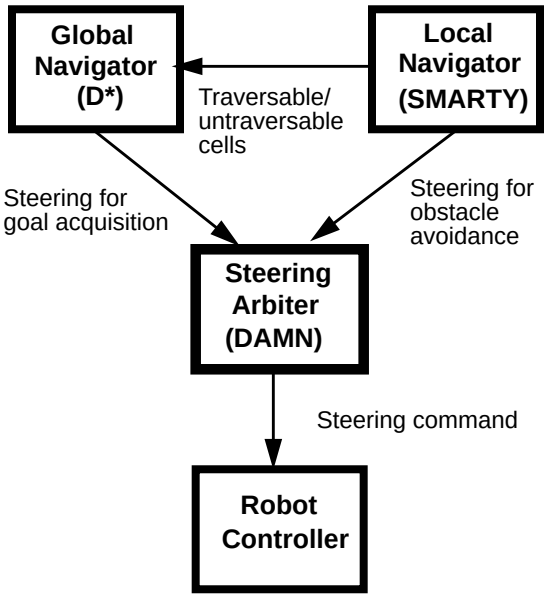
This paper describes the software system for goal acquisition in unknown environments. First, an overview is presented followed by descriptions of the global navigator, local obstacle avoider, and the steering integration system. Second, experimental results from actual robot runs are described. Finally, conclusions and future work are described.

2.0 The Navigation System

2.1 System Overview

Figure 3 shows a high-level description of the navigation system, which consists of the global navigator called D*, local navigator called SMARTY, and steering arbiter called the Distributed Architecture for Mobile Navigation (DAMN). The global navigator maintains a coarse-resolution map of the environment, consisting of a Cartesian lattice of grid cells. Each grid cell is labelled untraversable, high-cost, or traversable, depending on whether the cell is known to contain at least one obstacle, is near to an obstacle, or is free of obstacles, respectively. For purposes of our tests, all cells in the map were initialized to traversable. The global navigator initially plans a trajectory to the goal and sends steering recommendations to the steering arbiter to move the robot toward the goal. As it drives, the local navigator sweeps the terrain in front of the robot for obstacles. The ERIM laser rangefinder is used to measure the three-dimensional shape of the terrain, and the local navigator analyzes the terrain maps to find sloped patches and range discontinuities likely to correspond to obstacles. The local navigator sends steering recommendations to the arbiter to move the robot around these detected obstacles. Additionally, the local navigator sends detected untraversable and traversable cells to the global navigator for processing. The global navigator compares these cells against its map, and if a discrepancy exists (i.e., a traversable cell is now untraversable or vice versa), it plans a new and optimal trajectory to the goal. The key advantage of the global navigator is that it can efficiently plan optimal global paths and is able to generate a new path for every batch of cells in a fraction of a second. The global navigator updates its map and sends new steering recommendations to the steering arbiter.

Figure 3: Navigation System Diagram



The steering recommendations sent to the arbiter from the two navigators consist of evaluations for each of a fixed set of constant-curvature arcs (i.e., corresponding to a set of fixed steering angles). The global navigator gives a high rating to steering directions that drive toward the goal, and the local navigator gives a high rating to directions that

avoid obstacles. The arbiter combines these recommendations to produce a single steering direction which is sent to the robot controller.

The rest of this section details the global navigator (D*), the local navigator (SMARTY), the steering arbiter (DAMN), and the interfaces between them.

2.2 The D* Algorithm for Optimal Replanning

2.2.1 Overview

If a robot is equipped with a complete map of its environment, it is able to plan its entire route to the goal before it begins moving. A vast amount of literature has addressed the path-finding problem in known environments (see Latombe [18] for a good survey). In many cases, however, this scenario is unrealistic. Often the robot has only a partial map or no map at all. In these cases, the robot uses its sensors to discover the environment as it moves and modifies its plans accordingly. One approach to path planning in this scenario is to generate a "global" path using the known information and then circumvent obstacles on the main route detected by the sensors [10], generating a new global plan if the route is totally obstructed. Another approach is to move directly toward the goal, skirting the perimeter of any obstructions until the point on the obstacle nearest the goal is found, and then to proceed directly toward the goal again [19]. A third approach is to direct the robot to wander around the environment until it finds the goal, penalizing forays onto terrain previously traversed, so that new areas are explored [24]. A fourth approach is to use map information to estimate the cost to the goal for each location in the environment and efficiently update these costs with backtracking costs as the robot moves through the environment [16].

These approaches are complete (i.e., the robot will find the goal if a path exists), but they are suboptimal in the sense that they do not make optimal use of sensor information as it is acquired. We define a traverse to be optimal if, at every point *P* along the traverse to the goal, the robot is following a lowest-cost path from *P* to the goal, given all obstacle information known in aggregate at point *P.* It is possible to produce an optimal traverse by using A* [22] to compute an optimal path from the known map information, moving the robot along the path until either it reaches the goal or its sensors detect a discrepancy between the map and the environment, updating the map, and then replanning a new optimal path from the robot's current location to the goal. Although this brute-force, replanning approach is optimal, it can be grossly inefficient, particularly in expansive environments where the goal is far away and little map information exists.

The D* algorithm (or *Dynamic* A*) is functionally equivalent to the brute-force replanner (i.e., sound, complete, and optimal), but it is far more efficient. For large environments requiring a million map cells to represent, experimental results indicate that it is over 200 times faster than A* in replanning, thus enabling real-time operation. (See Stentz [29] for a detailed description of the algorithm and the experimental results.)

D* uses a Cartesian grid of eight-connected cells to represent the map. The connections, or arcs, are labelled with positive scalar values indicating the cost of moving between the two cells. Each cell (also called a "state") includes an estimate of the path cost to the goal, and a backpointer to one of its neighbors indicating the direction to the goal.

Like A*, D* maintains an OPEN list of states to expand. Initially, the goal state is placed on the OPEN list with an initial cost of zero. The state with the minimum path cost on the OPEN list is repeatedly expanded, propagating path cost calculations to its neighbors, until the optimal cost is computed to all cells in the map. The robot then begins to move, following the backpointers toward the goal. While driving, the robot scans the terrain with its sensor. If it detects an obstacle where one is not expected, then all optimal paths containing this arc are no longer valid. D* updates the arc cost with a prohibitively large value denoting an obstacle, places the adjoining state on the OPEN list, then repeatedly expands states on the OPEN list to propagate the path cost increase along the invalid paths. The OPEN list states that transmit the cost increase are called RAISE states. As the RAISE states fan out, they come in contact with neighbor states that are able to lower their path costs. These LOWER states are placed on the OPEN list. Through repeated state expansion, the LOWER states reduce path costs and redirect backpointers to compute new optimal paths to the invalidated states.

Conversely, if the robot's sensor detects the absence of an obstacle where one is expected, then a new optimal path may exist from the robot to the goal (i.e., through the "missing" obstacle). D* updates the arc cost with a small value denoting empty space, places the adjoining state on the OPEN list as a LOWER state, then repeatedly expands states to compute new optimal paths wherever possible. In either case, D* determines how far the cost propagation must proceed until a new optimal path is computed to the robot or it is decided that the old one is still optimal. Once this determination has been made, the robot is free to continue moving optimally to the goal, scanning the terrain for obstacles.

Figure 4 and Figure 5 illustrate this process in simulation. Figure 4 shows a 50 x 50 cell environment after the initial cost calculation to all cells in the space. The optimal path to any cell can be determined by tracing the backpointers to the goal. The light grey obstacle represents a known obstacle (i.e., one that is stored in the map), while the dark grey obstacle represents an unknown obstacle. Note that the backpointers pass through the dark grey obstacle since it is unknown. The robot, equipped with a 5-cell radial field-of-view sensor, starts at the center of the left wall and proceeds toward the goal. Initially, it deflects around the known obstacle, but heads toward the unknown obstacle. As its sensor detects the obstacle, the cost increases fan out from the obstacle via RAISE states until they reach the LOWER states around the bottom of the light grey obstacle. These LOWER states redirect the backpointers to guide the robot around the bottom and to the goal. Figure 5 illustrates the final map configuration, after the robot has reached the goal. Note that optimal paths have been computed to some but not all of the cells in the environment. This effect illustrates the efficiency of D*. It limits the cost propagations to the vicinity of the obstruction, while still ensuring that the robot's path is optimal.

Figure 6 shows a 450 x 450 cell simulated environment cluttered with grey obstacles. The black curve shows the optimal path to the goal, assuming all of the obstacles are known a priori. This traverse is known as omniscient optimal, since the robot has complete information about the environment before it begins moving. Figure 7 shows planning in the same environment where none of the obstacles are stored in the map a priori. This map is known as optimistic, since the robot assumes no obstacles exist unless they are detected by its 15-cell radial sensor. This traverse is nearly two times longer than omniscient optimal; however, it is still optimal given the initial map and the sensor information as it was acquired.

Figure 4: Backpointers after Initial Propagation

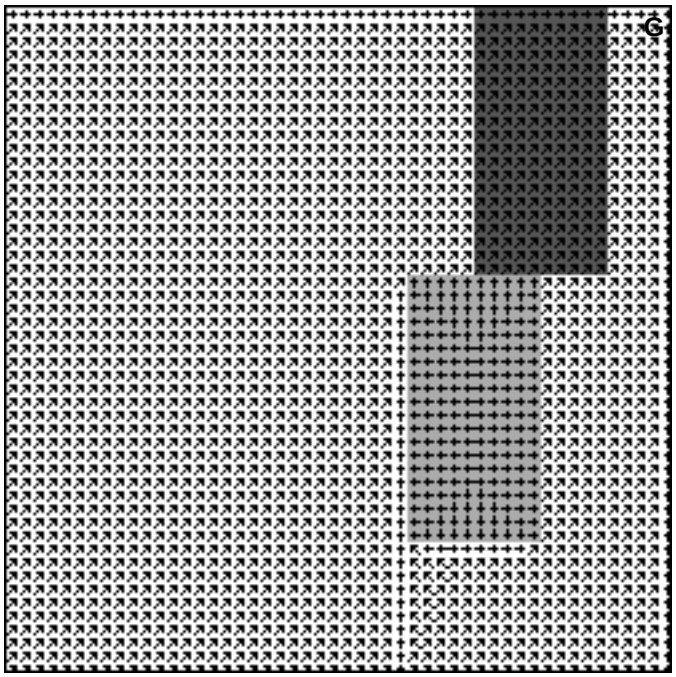


Figure 5: Final Backpointer Configuration

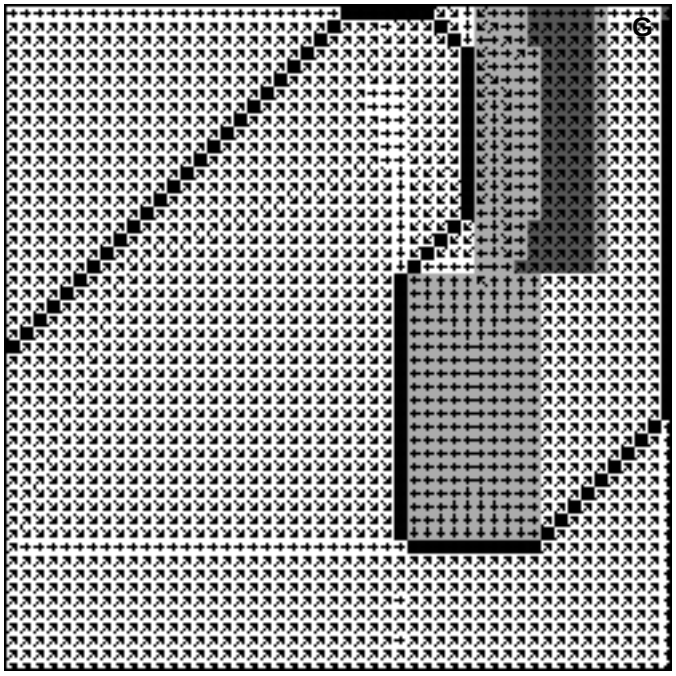


Figure 6: Planning with a Complete Map

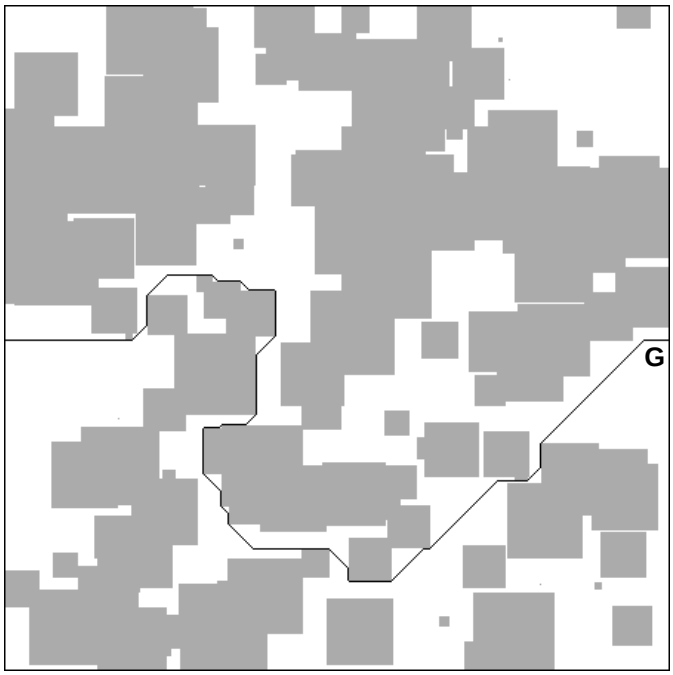
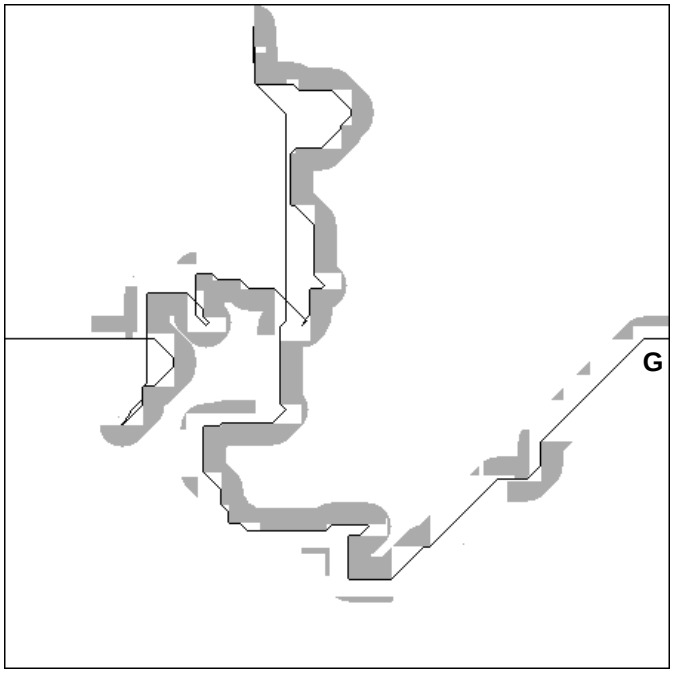


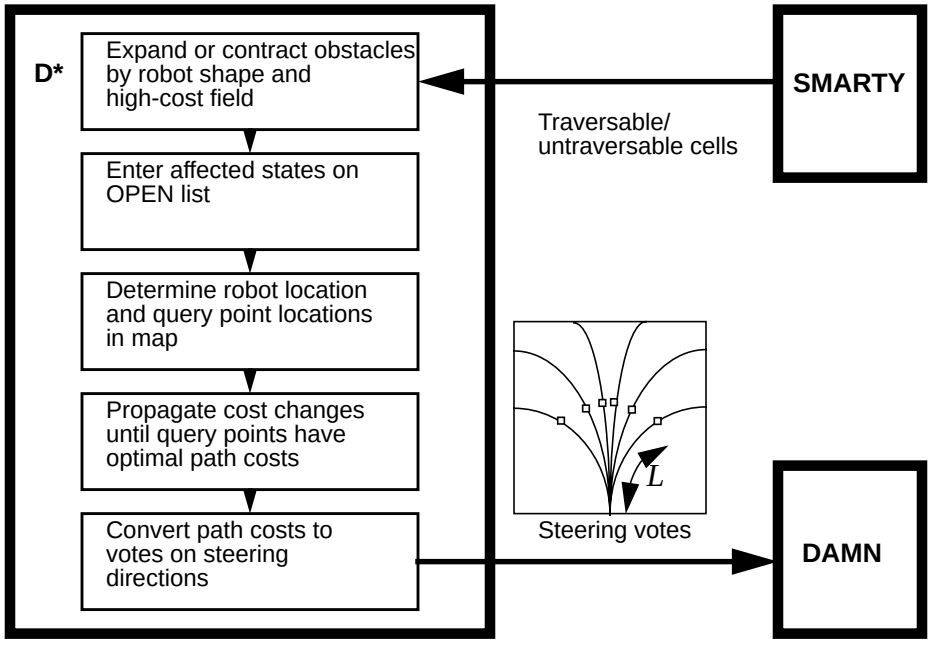
Figure 7: Planning with an Optimistic Map



2.2.2 Cell Expansion from SMARTY Data

A number of modifications were made to D* to adapt it to an actual robot system. The system diagram is shown in Figure 8. As the robot drives toward the goal, its laser rangefinder scans the terrain in front of the robot. The SMARTY local navigator processes this sensor data to find obstacles and sends the (*x*,*y*) locations of detected obstacles (untraversable cells) to D* at regular intervals, using the TCX [6] message passing system. Additionally, SMARTY sends (*x*,*y*) locations of cells known to be devoid of obstacles (traversable cells). Since the D* map is used to represent a global area, its grid cells are of coarser resolution than SMARTY's (i.e., 1 meter versus 0.4 meter). D* keeps track of the locations of obstacles within each of its grid cells, adding or deleting obstacles as needed based on the data from SMARTY. If an obstacle is added to a previously empty map cell or all of the obstacles are deleted from a previously obstructed cell, then this constitutes a significant event within D* since a traversable cell becomes an untraversable cell or an untraversable cell becomes a traversable cell, respectively.

Figure 8: D* Overview



Since D* does not explicitly model the shape of the robot (i.e., it is assumed to be a point), the new untraversable cells are expanded by half the width of the robot to approximate a configuration space (C-space) obstacle. All map cells within a 2-meter radius are classified as untraversable. Note that the longer dimension (i.e., length) of the robot is not used in the expansion. The length must be modelled to detect possible bumper collisions when driving forward or backward; however, D* is not intended to perform local obstacle avoidance. Instead, it is concerned with detecting channels between obstacles that are wide enough to admit the robot if it can get properly oriented. This heuristic breaks down in environments where the channels between obstacles are so constricted and convoluted that the robot's length or minimum turning radius prevents motion through them without collision. In such environments, planning with nonholonomic constraints in a full three-dimensional configuration space is required, but in our experiments we did not encounter the need for it.

The C-space expansion provides some buffering to keep the robot away from obstacles. We found that additional buffering in the form of a high-cost field leads to better performance. The idea is to penalize the robot cost-wise for passing too close to an obstacle, causing the robot to approach obstacles only when open space is unavailable. For example, an area cluttered with trees may be navigable, but a route around the trees would be less risky and therefore preferable. For each true untraversable cell (i.e., those containing obstacles from SMARTY, not those created from the C-space expansion), all traversable cells within a radius of 8 meters are classified as high-cost. The cost of traversing a high-cost cell is 5 times that of a traversable cell.

The selection of these parameters is a compromise between the degree to which we are willing to take risks by driving through narrow areas misclassified as traversable, and the degree to which we are willing to accept longer routes avoiding areas that are actually open. The degree of acceptable risk is related to the reliability of the sensor processing. That is, the buffer need not be large with high cost if the sensor classifies cells as traversable with high confidence. We have found empirically that setting the size and cost of the buffer to 8 and 5, respectively, is a good compromise. Therefore, if a channel between two obstacles requires the robot to drive through 10 high-cost cells, it would choose a longer, alternate route passing through up to 50 additional, traversable cells.

When an untraversable cell is changed to traversable, all of the corresponding untraversable and high-cost cells created during the C-space expansion are changed to traversable. Every time a cell changes classification (i.e., among traversable, untraversable, and high-cost), it is placed on the OPEN list for future consideration. The cost changes are propagated as needed to produce steering recommendations for the DAMN arbiter.

2.2.3 Steering Arc Evaluation for DAMN

Every 500 msec, D* sends steering recommendations to DAMN. D* first checks the robot position and then computes the endpoints of a static set of 51 steering arcs, linearly distributed in arc curvature from -0.125 meter^{-1} to +0.125 meter^{-1}. The points are computed to be at constant distance L from the current robot position along the arcs. L is currently fixed at 10 meters for a speed of about 2 m/sec. D* converts the points to absolute locations in its internal map. It then expands states on its OPEN list until it has computed an optimal path to every point in the list. The cost of the optimal path from each point to the goal is converted to a vote between -1 and +1, where +1 is a strong recommendation and -1 is a veto, and is sent to the arbiter. If a point happens to fall in an untraversable cell, a cell unreachable from the goal, or a cell not in D*'s map, it is assigned a vote of -1. Otherwise, if $c_{\min}$ and $c_{\max}$ are the minimum and maximum values of the costs in the current list of points, the vote v for an arc is derived from the cost c of the corresponding point in the following way:

$$v = \frac{c_{max} - c}{c_{max} - c_{min}} \quad if\ c_{max} \neq c_{min}$$

$$v = 0 \quad if\ c_{max} = c_{min}$$

This simple formula ensures that arcs going through obstacles are inhibited and that the preferred direction goes through the point of minimum cost. The vote for $c_{\max}$ is set to 0 instead of -1 because a high cost means that the arc is less desirable but should not be inhibited.

Because D* does generate enough information to inhibit arcs going through obstacles, one could be tempted to eliminate the obstacle avoidance behavior from SMARTY and to use D* as the sole source of driving commands. Although it is quite possible to configure the system to run in this mode, the result would be poor performance for at least three reasons. First, D*'s map is lower resolution than the map used internally by SMARTY (1 meter vs. 0.4 meter in the current implementation). As a result, D* cannot control fine motion of the robot. Second, SMARTY typically generates commands faster than D* can update its map and propagate costs, thus ensuring lower latency. Third, SMARTY evaluates the distances between the arcs and all the obstacle cells in the map whereas D* evaluates only the cost of a single point on the arc. In addition to these practical considerations, it is generally ill-advised to mix local reflexive behavior such as obstacle avoidance and global behaviors such as path planning in a single module.

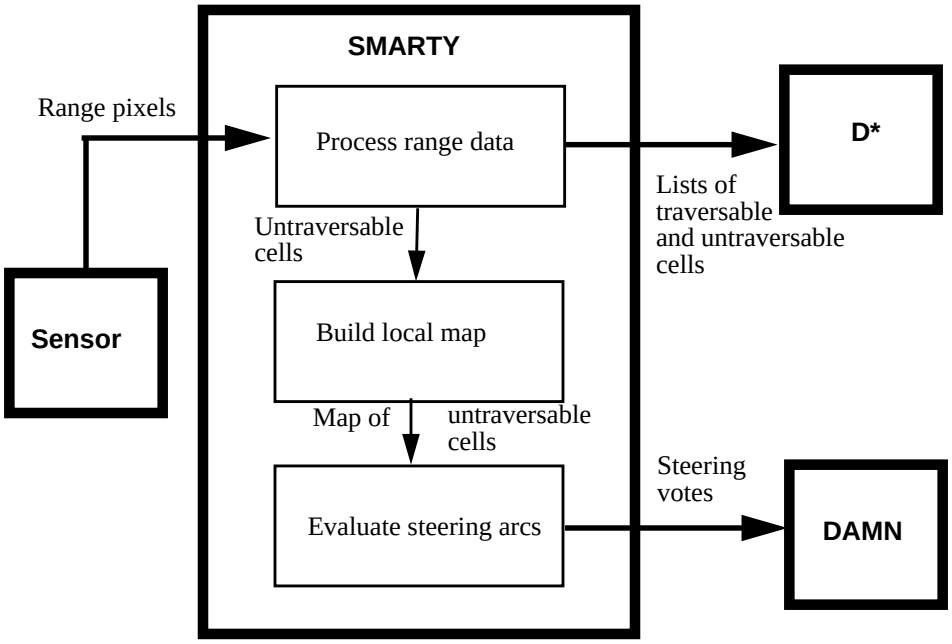
Finally, we note that the interface between D* and DAMN is actually implemented as a separate module connected through TCX. The module generates steering requests to D* every 500 msec and converts costs to votes and passes them to DAMN. We chose the distributed approach because it isolates D* cleanly from the details of the driving system and because the additional computation and scheduling time would slow down D*'s main expansion loop. Because the query points sent to D* are expressed with respect to the robot, they do not change once L is fixed. We use this property to reduce the message traffic by first sending an initialization message to D* which contains the list of all the points with respect to the robot. After initialization, the interface simply sends the current robot position to D* which converts the points to its coordinate system according to this position. In this approach, a request from the interface module consists of a short message of three numbers: x, y, and heading of the robot.

2.3 The SMARTY System for Local Navigation

2.3.1 Overview

SMARTY is a system for controlling a robot based on input from a range sensor. Figure 9 shows SMARTY's basic organization. The range data processing component takes a stream of range pixels as input and outputs untraversable locations with respect to the robot position at the time the data was acquired. The local map manager receives the lists of untraversable cells as soon as they are detected by the range processing portion and maintains their location with respect to current robot position. The local map manager sends the entire list of untraversable cells to the steering arc evaluation module at regular intervals. The arc evaluator computes safety scores for a fixed set of arcs based on the relative positions of untraversable cells with respect to the current robot position and sends them to the DAMN steering arbiter. These three parts are implemented as a single Unix process in which the range data is processed continuously as fast as it is acquired and the arc evaluator is activated at regular time intervals. Like D*, SMARTY communicates with external modules through the TCX messaging system. We briefly describe the three SMARTY components in the next paragraphs and conclude this section with a detailed description of the interface between D* and SMARTY.

Figure 9: SMARTY Overview



2.3.2 Range Data Processing

The range image processing module takes a single image as input and outputs a list of regions which are untraversable (see Figure 10). The initial stage of image filtering resolves the ambiguity due to the maximum range of the scanner, and removes outliers due to effects such as mixed pixels and reflections from specular surfaces. (See Hebert [12] for a complete description of these effects.) After image filtering, the (x,y,z) location of every pixel in the range image is computed in a coordinate system relative to the current robot position. The coordinate system is defined so that the z axis is vertical with respect to the ground plane. The transformation takes into account the orientation of the robot read from the inertial navigation system. The points are then mapped into a discrete grid on the (x,y) plane. Each cell of the grid contains the list of the (x,y,z) coordinates of the points which fall within the bounds of the cell in x and y. The terrain classification as traversable or untraversable is first performed in every cell individually. The criteria used for the classification are:

- the height variation of the terrain within the cell,
- the orientation of the vector normal to the patch of terrain contained in the cell,

- and the presence of a discontinuity of elevation in the cell.

To avoid frequent erroneous classification, the first two criteria are evaluated only if the number of points in the cell is large enough. In practice, a minimum of 5 points per cell is used.

The processing is performed scanline by scanline instead of processing the entire range image and sending the appropriate map cells at completion, as described in Hebert [11]. In scanline- or pixel-based processing, each pixel is converted to Cartesian coordinates and the state of the corresponding cell in the map is updated. A cell is reported as untraversable as soon as a component of its state exceeds some threshold, e.g., the slope becomes too high, and the number of points in the cell becomes high enough to have confidence in the status of the cell. This approach has several advantages over the traditional image-based approaches. First, it guarantees that an obstacle is reported to the system as soon as it is detected in the range image instead of waiting for the entire image to be processed, thus reducing the latency. Second, the scanline approach permits a simple scheduling algorithm for synchronizing range image processing with other tasks: The system simply executes a given task, e.g., sending steering evaluations to the planner, after processing a scanline if enough time has elapsed since the last time the task was executed.

Figure 11 shows a typical example of scanline processing of range images. Figure 11(a) is a video image of the scene which consists of a corridor limited by impassable rocks and trees. Figure 11(b) shows a 64 x 256 range image of the same scene. Points near the sensor are dark, and those farther away are light. Superimposed on the range image are the locations of two scanlines. Figure 11(c) shows an overhead view of the pixels from each of the scanlines marked in Figure 11(b). To produce this display, the pixels from the range image are converted to Cartesian coordinates and projected on the ground plane. The impassable regions are displayed as black dots and are found mostly on the right side of the robot, corresponding to the “wall” visible on the right of Figure 11(a). The order of the scanlines in Figure 11(c) reflects the order in which the data is processed: first the bottom scanline (closest point) and then the top (farthest point).

Run-time parameters can be set to accommodate the maximum anticipated robot speed. For a maximum speed of 3 m/sec, only the upper two-thirds of the range image is processed at a rate of 200 msec/image. A list of impassable cells is sent to the local map manager every 100 msec. In this configuration, the minimum detection range for a 30 cm object is 10 meters, although the system can detect larger objects up to 40 meters from the sensor. The maximum detection range, together with sensor latency, 0.5 sec/image, are the main limitations of the system.

Figure 10: Range Image Processing

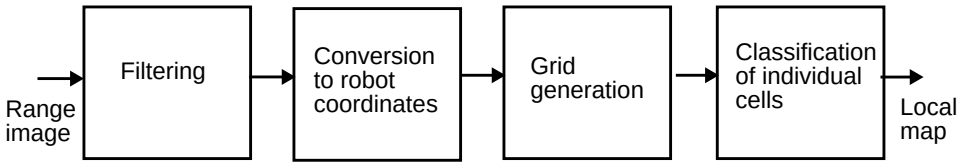
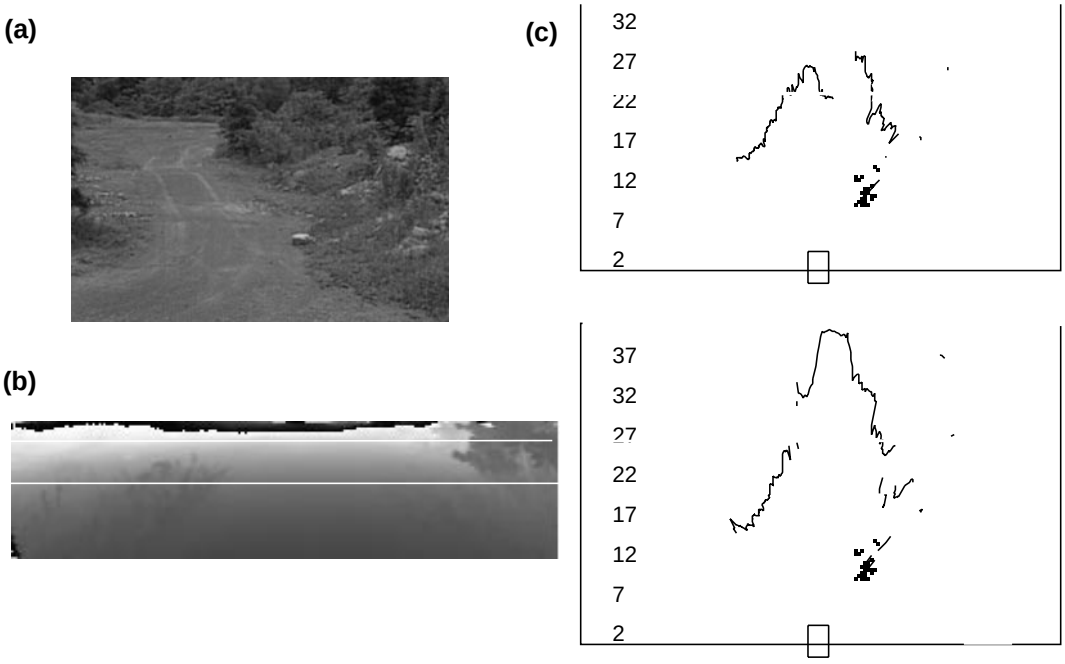


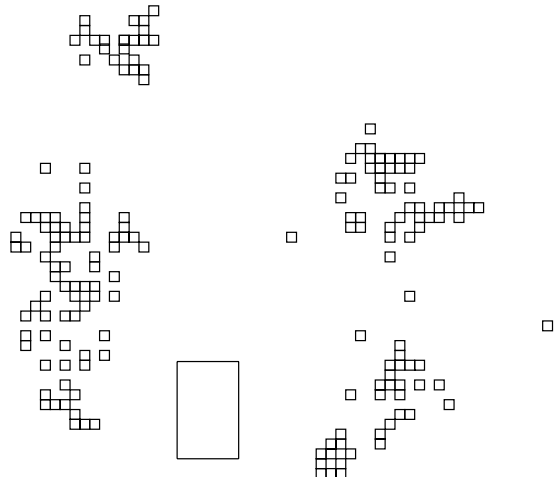
Figure 11: Scanline-Based Processing from a Range Image



2.3.3 Local Map Management

The local map is an array of cells with a simpler structure than the grid used in the range data processing component. Local map cells contain only a binary flag indicating whether the cell is traversable; if it is not, the cell also contains the coordinates of a 3-D point inside the obstacle. The positions of the untraversable cells in the local map are updated at regular intervals, currently 100 msec, according to robot motion. Figure 12 shows an overhead view of the local traversability map constructed from a sequence of images from the area shown in Figure 11(a). In this example, the cells are 40 cm x 40 cm. The untraversable cells are shown as small squares; the robot is indicated by the rectangle at the bottom of the display.

Figure 12: A Local Map

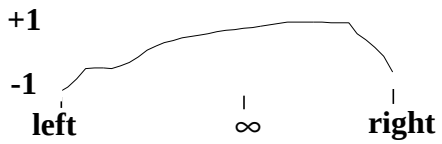


2.3.4 Steering Arc Evaluation for DAMN

The data in the local map is used for generating admissible steering commands. We give here only a brief description of the approach and refer the reader to Keirsey [13] and Langer [17] for a detailed description of the planning architecture. Each steering arc is evaluated by computing the distance between every untraversable cell in the local map and the arc. An arc receives a vote of -1 if it intersects an untraversable cell; if not, it receives a vote varying monotonically between -1 and +1 with the distance to the nearest untraversable cell. After the vote for each individual arc is computed, the entire array of votes is sent to an arbiter module [13] which generates an actual steering command that is sent to the robot.

Figure 13 shows an example of arc evaluation for the local map in Figure 12. The distribution of votes ranges from a minimum turning radius of -8 meters to a maximum of +8 meters. The curve shows the maximum votes are for moderate right turns of the robot and are close to -1 for left and right turns.

Figure 13: Distribution of Votes for the Map of Figure 12



2.3.5 Cell Transmission to D*

The system described so far is one of our conventional autonomous driving systems [17]. In order to use SMARTY in conjunction with the D* module, we added a direct link between D* and SMARTY (see Figure 3), because D* needs to update its internal map based on the information extracted from the range images. In theory, SMARTY should send lists of traversable and untraversable cells found in the current batch of range pixels to D* as soon as new data is available. In practice, however, this causes D* to receive data faster than it can process it, due to the overhead in sending and receiving messages.

In order to avoid this problem, the lists of traversable and untraversable cells are buffered instead of being sent to D* as soon as they are computed. In addition, the position of the robot at the time the data used for computing the current cells was acquired is also buffered. The robot position is sent to D* along with the lists of cells. The position information is necessary to enable D* to convert the robot-centered cell locations to cell locations in its internal global map. After a new line of range data is processed, SMARTY flushes the buffer if either of two conditions is met:

- Enough time has elapsed since the last message to D*. The time interval between messages is 500 msec. This value is set empirically for the hardware configuration currently used.
- The position of the robot at the time the most recent sensor data was acquired is different from the position of the robot at the time the data in the buffer was acquired.

The first condition ensures that messages are not generated at a frequency too high for D* to process them. The second condition is necessary because the messages sent to D* include a single robot position so that they cannot accommodate lists of cells acquired from two different robot positions. This message protocol provides a good compromise between the need to send up-to-date information to D* and the need to limit the number of messages to D*.

2.4 The DAMN System for Steering Arbitration

Our navigation system includes specialized modules such as obstacle detection (SMARTY) and path planning (D*). Each of the modules has its own view of the best action for the robot to take at every step. We use the DAMN architecture for combining recommendations from different modules and for issuing actual driving commands to the robot controller [17][23][26].

The DAMN architecture consists of an arbiter which receives steering recommendations from outside modules and combines them into a single driving command. The recommendations are in the form of votes between -1 and +1 for a pre-defined set of arcs. A vote of -1 means that the external module has determined that the arc should not be driven, e.g., because of an obstacle blocking the path, and +1 means that the path is highly recommended. First, the arbiter combines the votes from all the external modules into a single distribution of votes for the list of arcs. For each arc, DAMN multiplies the votes from each module by its module weight and then sums these weighted votes to produce a single, composite vote. Second, the arbiter chooses the arc with the highest composite vote and sends it to the robot controller.

In practice, each module is a separate Unix process which communicates with the arbiter through the TCX communication system. Because they may have very different cycle times, the modules operate asynchronously. The arbiter sends commands to the controller at regular intervals, currently 100 msec, and updates the list of combined votes whenever new votes are received from the other modules.

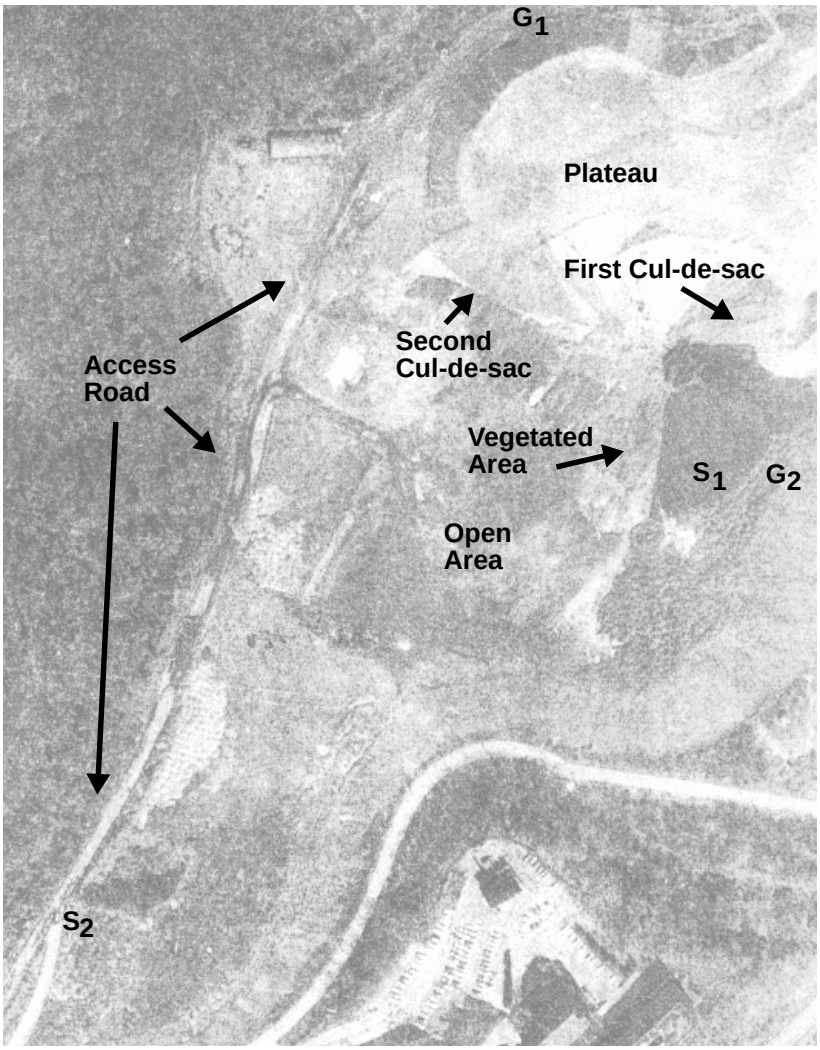
Because the DAMN arbiter does not need to know the semantics of the modules from which it combines votes, it is very general and has been used in a number of systems with different configurations of modules [30]. We concentrate here on the configuration of our navigation system as illustrated in Figure 3. The arbiter receives votes from two modules, D* and SMARTY. The global navigator, D*, sends votes based on its global map and the goal location. The local navigator, SMARTY, sends votes based on the data extracted from range images. The former generates driving recommendations based on global path constraints while the latter generates recommendations based on a detailed description of the local terrain. Module weights of 0.9 and 0.1 were used for SMARTY and D* respectively. This selection has the effect of favoring obstacle avoidance over goal acquisition, since it is more important to miss obstacles than to stay on course to the goal.

The current version of DAMN allows for forward motion, but it does not evaluate steering directions for reverse driving. Of course, this is not a problem for on-road navigation systems or for systems which use sufficient a priori knowledge of the environment. In our case, however, it is entirely possible that the only way for the robot to reach the goal is to drive in reverse out of a cul-de-sac. This capability was not yet added to DAMN at the time of this writing so that reverse driving had to be simulated by manually driving the robot out of occasional cul-de-sacs. We clearly indicate such occurrences in the examples given in the next section.

3.0 Experimental Results

Two of the trial runs that illustrate different aspects of the system are examined in this section. The system was run at a local test site called the Slag Heap, located about ten minutes from campus. The Slag Heap consists of a large open area of flat terrain bounded by a berm on one side and a large plateau on the other. The obstacles consist of sparse mounds of slag in the interior of the flat area and small trees, bushes, rocks, and debris around its edges. An access road skirts the perimeter of the area. An aerial view of the test site is shown in Figure 14. The dimensions of this area are approximately 800 x 1000 meters.

Figure 14: Aerial View of Slag Heap Test Site



3.1 Goal Acquisition with Backtracking

For both trials, an optimistic map was used (i.e., all cells are traversable). S_1 and G_1 mark the start and goal locations for the first trial. S_1 was located in the open area, and G_1 was located on the access road behind the large plateau. These points were chosen so that backtracking would be required to circumnavigate the plateau. Data from the trial at a number of points is shown in Figure 15 through Figure 20. Each of these figures consists of four parts: (a) the robot's trajectory superimposed on D*'s map; (b) the ERIM laser rangefinder image at a selected point along the trajectory; (c) SMARTY's local obstacle map at this same point; and (d) the votes from SMARTY, D*, and DAMN at this same point.

In Figure 15(a), the robot's trajectory is depicted by the black curve. The small rectangle near the end of the curve is the "selected point" from which the data for subfigures (b), (c), and (d) was taken. The C-space obstacles are shown in dark grey and the high-cost buffer cells in light grey. In Figure 15(b), grey scales encode the range values for the laser rangefinder, with dark grey values near the sensor and light grey values farther away. In Figure 15(c), the local map is shown for the area around the robot. The obstacle cells are shown as squares. The rectangular icon at the bottom of the display shows the position of the robot. A 20 meter local map was used to generate this display. In Figure 15(d), the steering votes for each module are shown, ranging from -1 to +1.

Figure 15: Driving into the Cul-de-sac

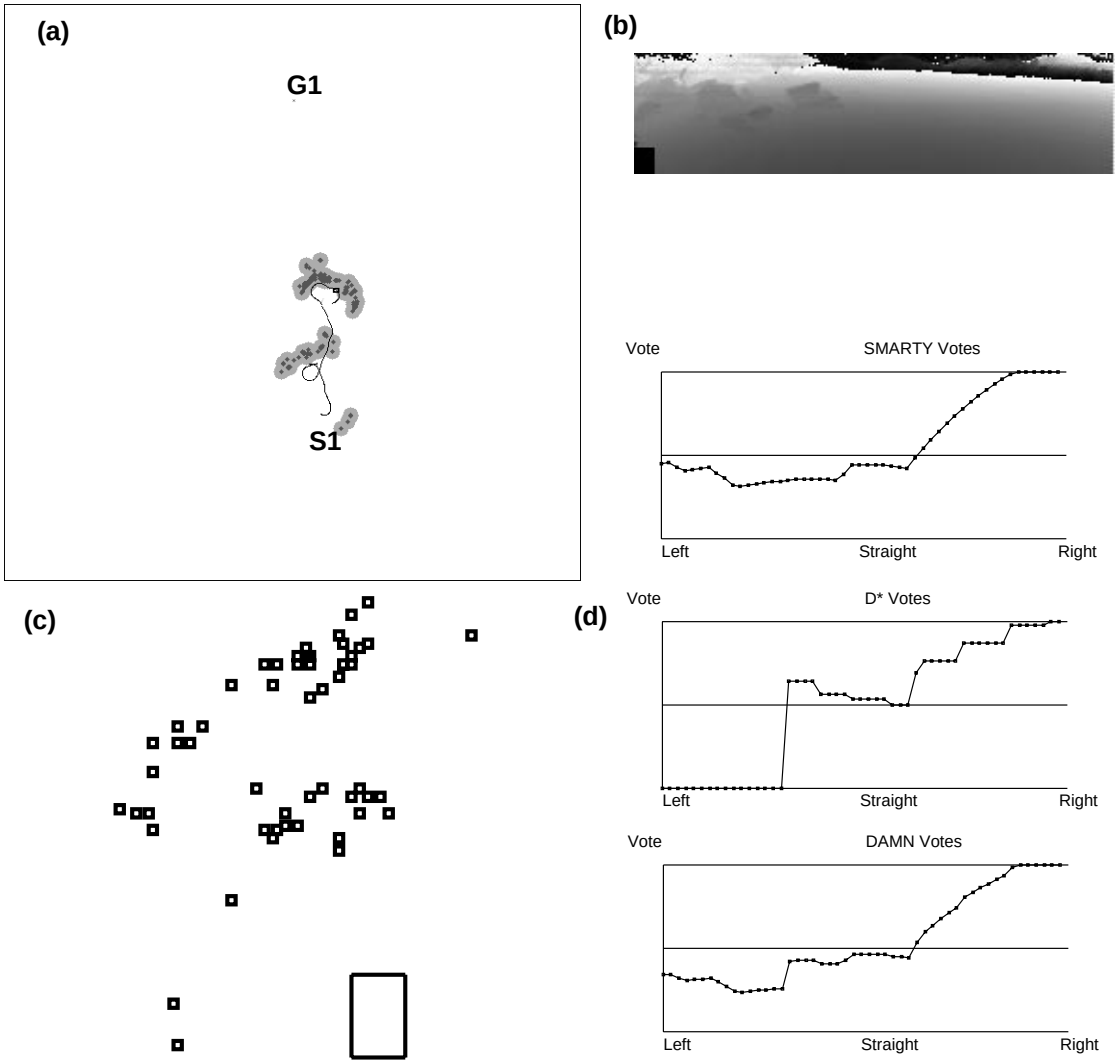


Figure 15 shows the first portion of the trial. The robot began pointing away from the goal, so it immediately turned around and headed directly toward it. The robot encountered a large obstruction, initially turned to the left, then looped around the obstacle to the right and drove into a cul-de-sac. At the selected point, SMARTY voted to turn right to avoid the boundary of the cul-de-sac, and D* voted in a similar fashion in order to loop back and explore the other side of the cul-de-sac.

Figure 16: Discovering the Cul-de-sac

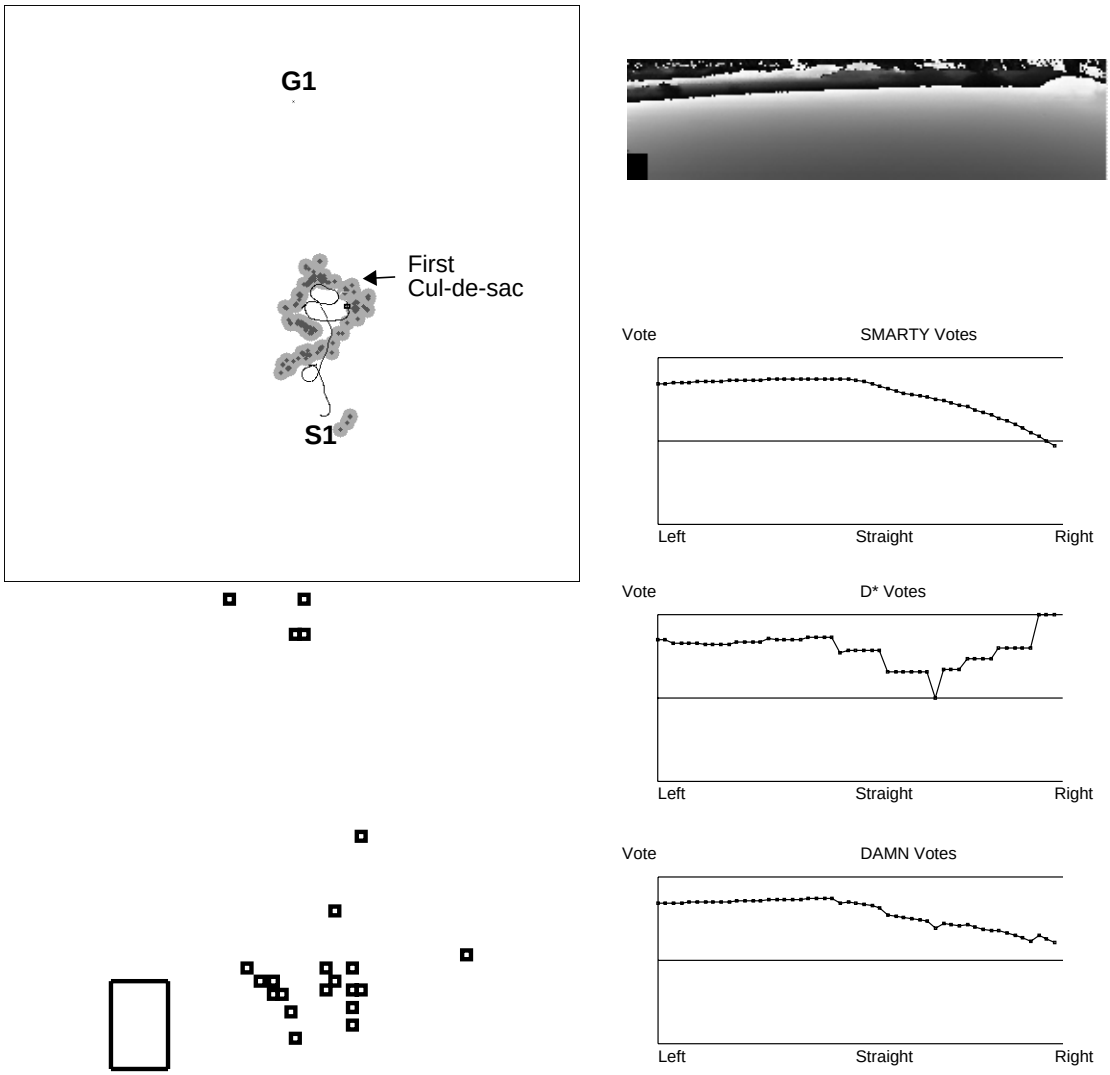
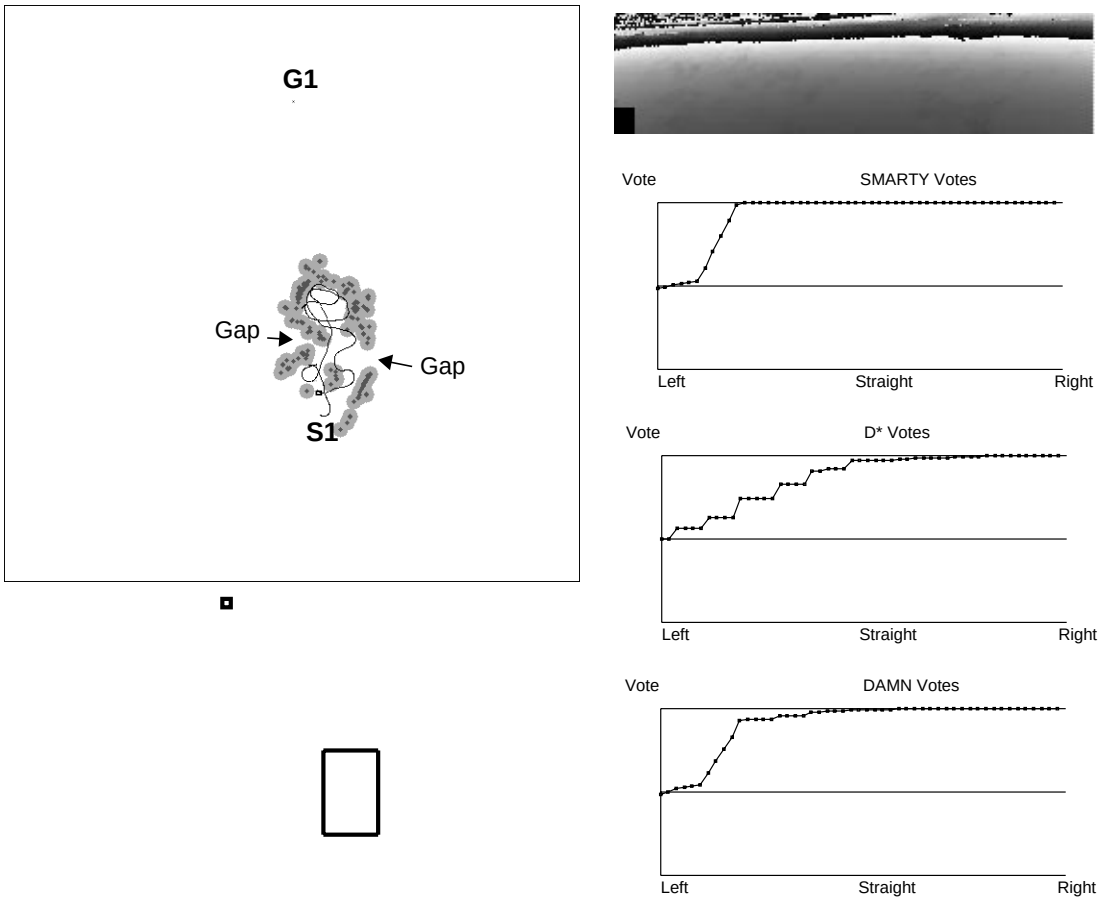


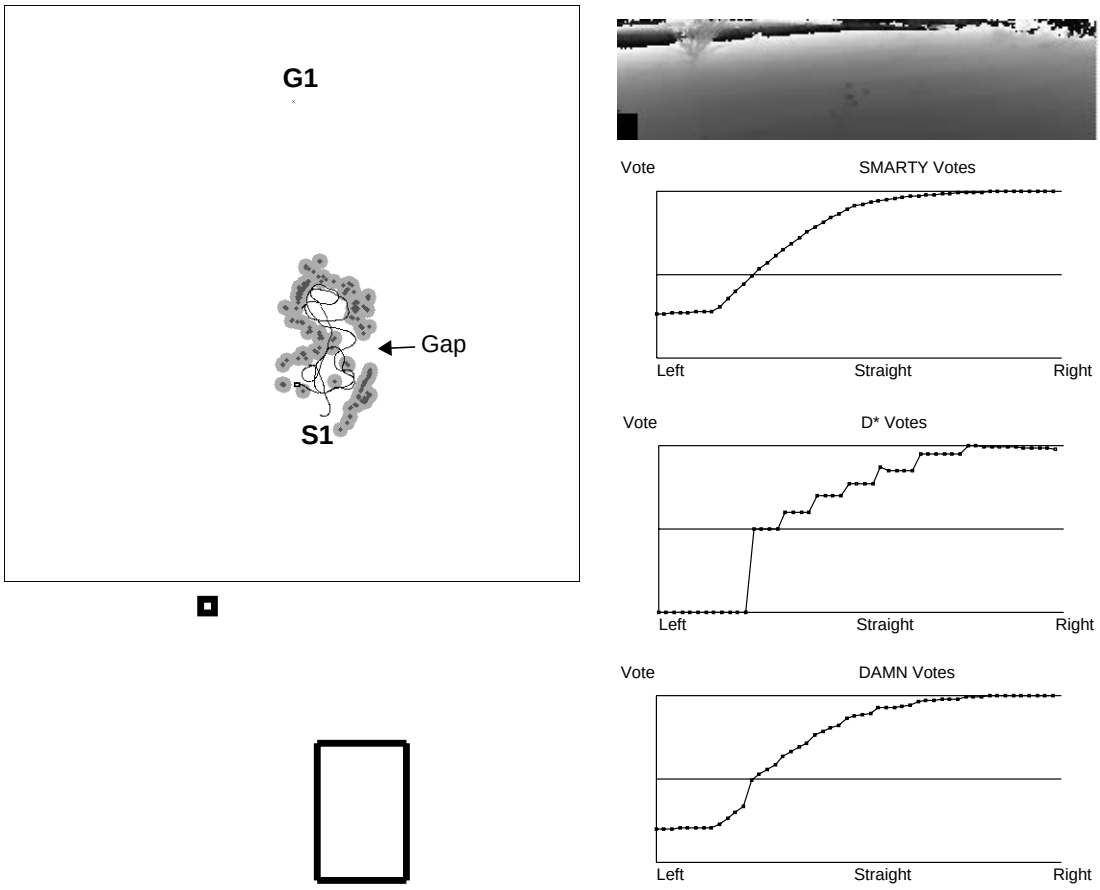
Figure 16 shows the robot driving side-to-side discovering the bounds of the cul-de-sac with its sensor. It appears that, at the selected point, D* preferred to venture into the high-cost buffer rather than backtrack out of the cul-de-sac; it was overruled by SMARTY's votes to avoid the cluttered area altogether. Since D* considers the cost to the goal only from the ends of the steering arcs, it relies on SMARTY to steer clear of obstacles coincident with, or near to, the arcs themselves.

Figure 17: Exiting the Cul-de-sac



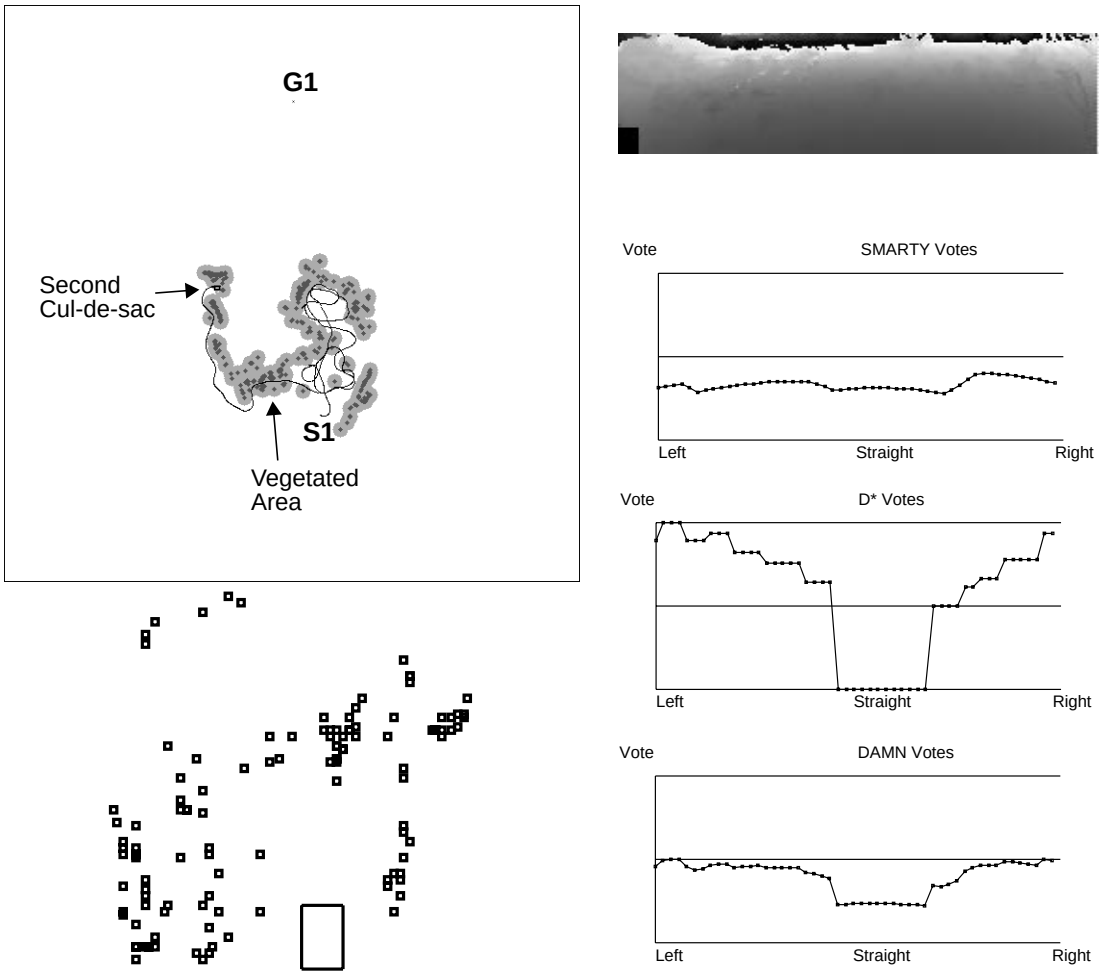
Once the robot discovered that the "route" was obstructed, it backtracked out of the cul-de-sac as shown in Figure 17. After exiting, the robot looped back in an attempt to drive through perceived gaps in the surrounding berm. Note that at the selected point, D* voted to turn right and head back toward a gap.

Figure 18: Looping Back Toward the “Gap”



After looping back, the robot closed the first gap with data from the laser rangefinder, and SMARTY deflected the robot away from the second due to other obstacles in the vicinity (see Figure 18).

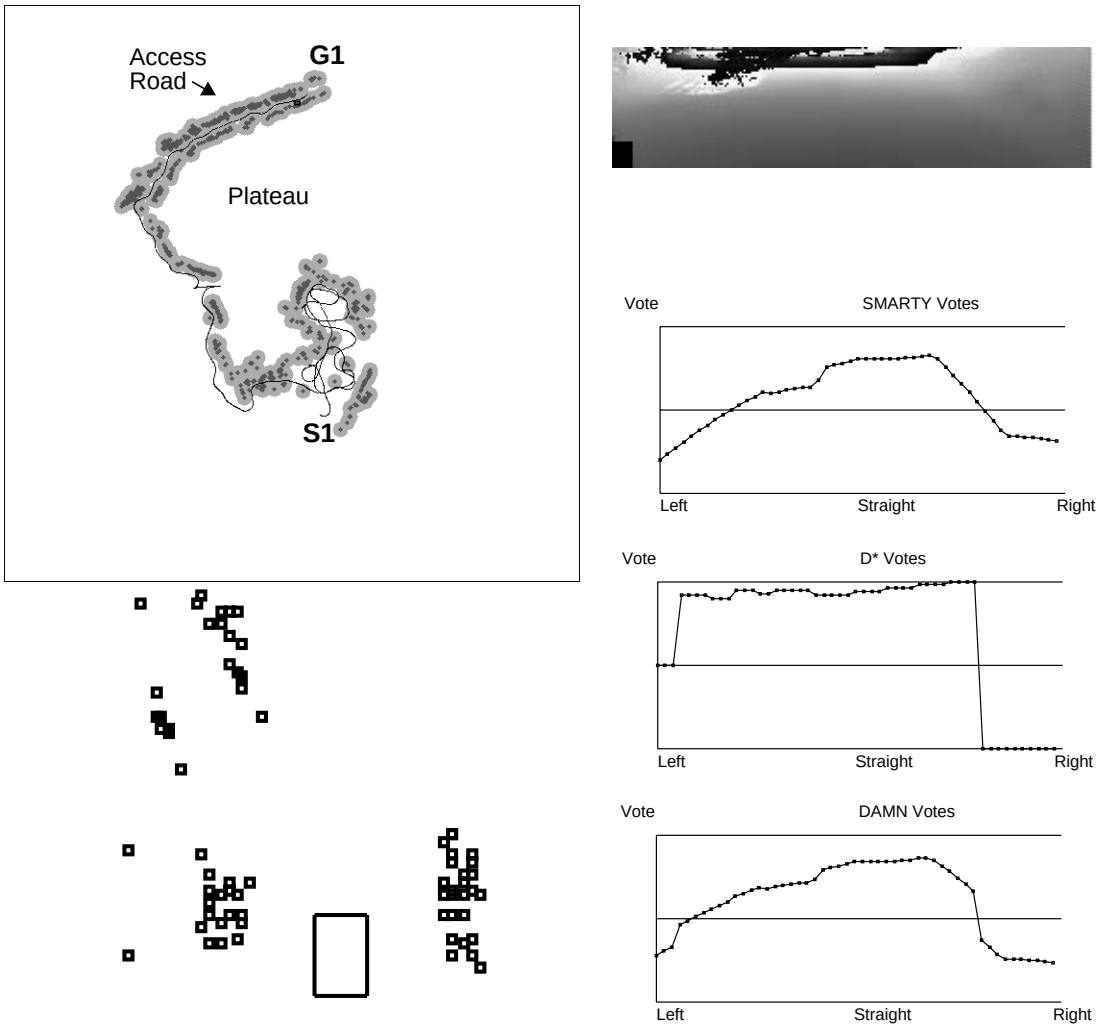
Figure 19: Driving Through and Around the Vegetated Area



In Figure 19, the robot moved into an area densely populated with small trees before driving out to the left. In these types of areas, the robot was driven predominantly by SMARTY, since obstacle avoidance takes precedence over the goal seeking behavior. After emerging from the area, D* guided the robot around the perimeter of the vegetation and into another small cul-de-sac. As was the case with the first cul-de-sac, the limited field of view of the ERIM sensor precluded the possibility of detecting the cul-de-sac before entry and avoiding it altogether.

This time the cul-de-sac was too small for the robot to escape without driving in reverse. Since the NAVLAB II is currently unable to do this automatically, the robot was manually driven in reverse until it exited the cul-de-sac. Note that, at the selected point, SMARTY detected obstacles in all directions and consequently voted against all steering arcs. D* favored backing out, but since such a move was not possible autonomously, it voted for the next best options: either a hard left or hard right turn.

Figure 20: Finding the Access Road that Leads to the Goal



After backing out, the robot was placed under autonomous control once again (see Figure 20). It drove around the perimeter of the large plateau, found an entry point to the road behind the plateau, and then drove along the access road until it reached the goal.

The total path length for the trial was 1410 meters. At six points during the trial, we manually intervened to steer the robot. Half of these interventions were to drive the robot in reverse, and the other half were steering deflections to avoid water or mud that was too difficult to detect with a laser rangefinder. D* sent steering recommendations to the DAMN arbiter every 500 msec. A total of 2952 sets of steering recommendations were sent. Since each set consists of 51 steering arcs, a total of 150,552 global paths to the goal were computed during the trial. SMARTY sent 6119 messages to D* containing a total of 1,851,381 terrain cell classifications. The radius of C-space expansion was 2 meters, and the radius of each high-cost buffer was 8 meters. A high-cost cell was 5 times more expensive to traverse than a traversable cell.

The number of cell classifications was large since each terrain cell is likely to be seen more than once, and each occurrence is transmitted to D*. It is also important to note that the classification for many terrain cells changed repeatedly from one sensor reading to the next. This effect was due in part to sensor noise and in part to the fact that the classification of a given cell improves in accuracy as the robot draws nearer and the sensor gets a better view.

Note that the high-cost buffer was essential to complete the boundaries of the cul-de-sacs, plateau, and roads. Without it, the robot would need to loop around many more times for more sensor data in the first cul-de-sac before D* became convinced the route was obstructed. We observed this behavior in actual experiments. In some cases, the robot had to loop up to ten times inside the cul-de-sacs in order to map the entire boundary of the region. Although this behavior is entirely justified from a planning perspective, since the robot needs to explore an entire area before determining it must backtrack out of it, it is clearly unacceptable from a practical standpoint. Better sensing is the proper solution.

3.2 Using Map Data from Prior Trials

In Figure 14, S_2 and G_2 mark the start and goal locations, respectively, for the second trial. The start was chosen to be at the entrance to the Slag Heap on the access road, and the goal was at the opposite end of the flat, open area. The objective of this trial was not to illustrate a difficult path to the goal; instead, it was to illustrate the effects of multiple runs over the same area. In Figure 21, the robot drove down the access road and across the open area to the goal. At the goal point, the robot was taken out of automatic control and driven manually to another portion of the access road (see Figure 22). During the manually-driven segment, the software continued to run; thus, the map was updated with obstacles detected by the sensor.

Figure 21: Driving the Initial Path to the Goal

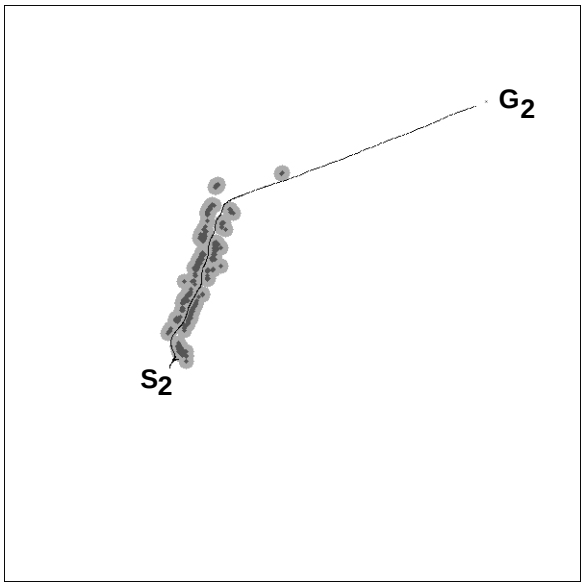


Figure 22: Manually Driving to a New Start Point

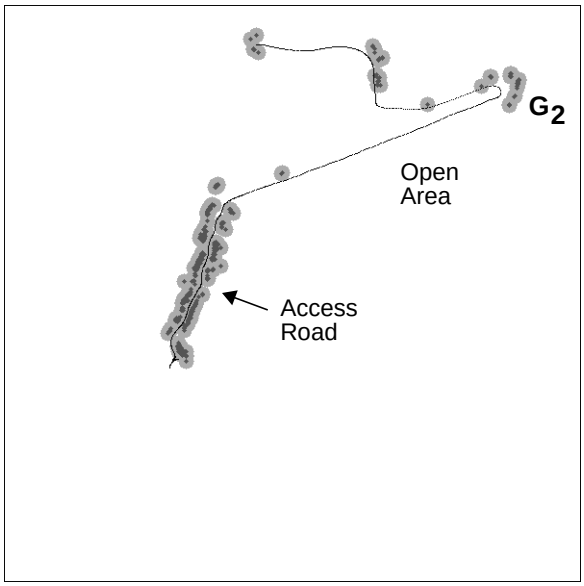
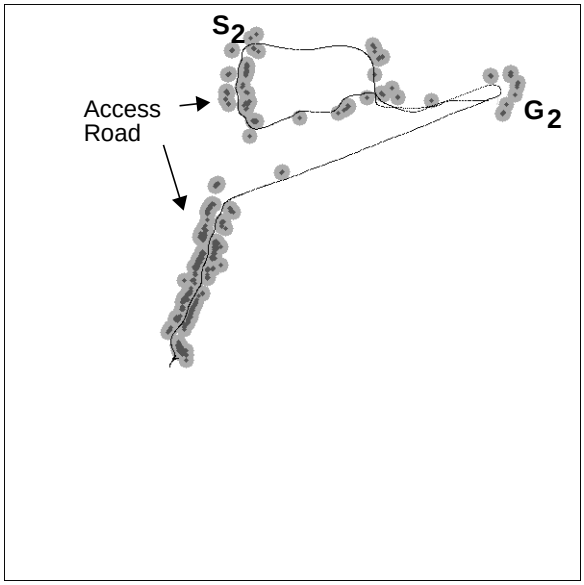


Figure 23: Finding the Goal for a Second Time



The robot was placed under automatic control again, and it drove down a segment of the access road until it found an entry point into the open area. It then proceeded to drive across the open area to the goal, avoiding a number of obstacles along the way (see Figure 23).

The robot was driven manually once again to a new start point (see Figure 24) and placed back under automatic control. It then drove to the goal for a third time (see Figure 25). Note that, in its third path to the goal, the robot used map data that was constructed during the previous traverses. As shown in the figure, the robot positioned itself to pass between two obstacles before its sensor was close enough to spot them.

Figure 24: Driving Manually to a Third Start Point

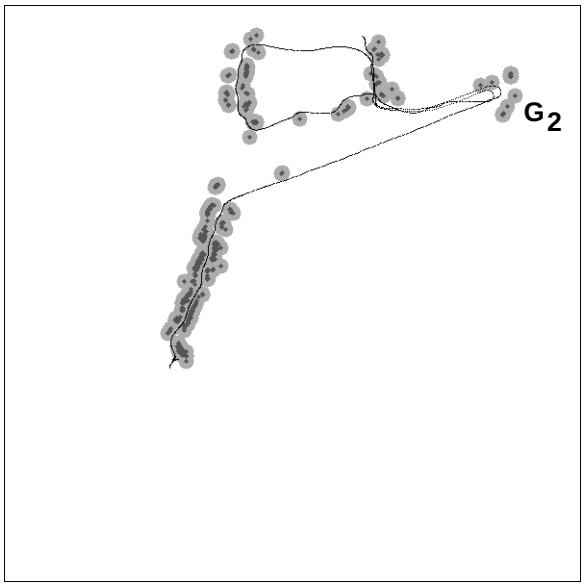
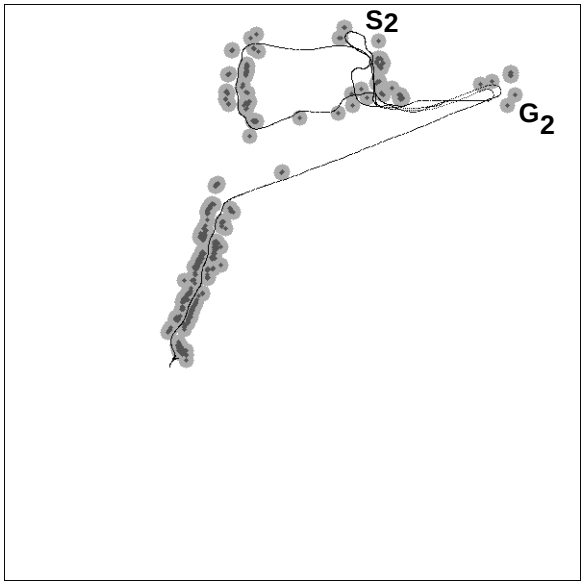


Figure 25: Driving to the Goal Using Prior Map Data



The length of the robot's path for this trial was 1664 meters, including both automatically and manually driven segments. D* sent a total of 3168 sets of steering recommendations to DAMN; thus, a total of 161,568 global paths were computed. SMARTY sent 6581 messages to D* with a total of 1,601,161 cell classifications.

4.0 Conclusions

4.1 Summary

This paper describes a complete navigation system for goal acquisition in unknown environments. The system uses all available prior map data to plan a route to the goal, and then begins to follow that route using its laser rangefinder

to examine the terrain in front of the robot for obstacles. If a discrepancy is discovered between the sensor data and the map, the map is updated and a new, optimal path is planned to the goal. Because of the efficiency of D*, this new path can be generated in a fraction of a second. Both the global navigator (D*) and the local navigator (SMARTY) send steering recommendations to the steering arbiter (DAMN). Because obstacle avoidance takes precedence over goal acquisition, the votes from SMARTY are weighted more heavily than those from D*. Thus, in areas dense with obstacles, the robot is driven primarily by SMARTY, while in open areas, it is primarily driven by D*. It was found that the high-cost buffers around obstacles were essential to fill in gaps between obstacles and preclude repeated sensing of the same area. It was also discovered that the two-dimensional approximation of the robot's three-dimensional configuration space was sufficient for the experiments conducted.

To our knowledge, this system is the first to demonstrate efficient goal acquisition and obstacle avoidance on a real robot operating in an unstructured, outdoor environment.

4.2 Future Work

In the near-term, a number of improvements will be made to minimize unnecessary processing and increase overall system speed. An enormous number of terrain cell classifications are transmitted from SMARTY to D*. Some of these classifications are erroneous due to noise or less-than-ideal viewing conditions. Noise filtering and verification of the classifications across sensor images would increase confidence in the data and reduce communications traffic. Currently, the cell classifications received by D* are processed sequentially to create the C-space expansions and high-cost buffers. This approach is highly inefficient given typical clustering of obstacles, and the additional computational burden resulted in D* sending less-than-optimal steering recommendations in cluttered areas in order to meet timing deadlines. Processing the obstacles in batch mode using the grassfire transform for expansion should greatly reduce this overhead. Furthermore, we will develop a more systematic approach to the scheduling of the interactions between SMARTY and the other modules, D* and arbiter. Currently, the frequency at which SMARTY sends information to the other modules is determined empirically for a typical robot speed. We will develop an algorithm that relates all the system parameters, such as sensor field of view or robot speed, to the communication frequency. This last improvement will involve first moving the modules to a real-time operating system in order to guarantee repeatable performance.

In the far-term, we will extend D* so that it dynamically allocates new map storage as needed rather than requiring the map to be pre-allocated to a fixed size. Furthermore, we will add the capability in D* to reason about robot maneuvers (coupled forward-backward motion) in order to handle very cluttered areas that require such complicated motion. We will include a mechanism for speed control in SMARTY in addition to the existing mechanism for steering control. Speed control involves reasoning about the distribution of map cells with respect to the robot and issuing recommendations for speed settings such that the robot slows down in cluttered environments. The speed recommendations may be encoded very much like the steering recommendations: a set of votes for speed values between 0 and a pre-set maximum speed. Additionally, we will improve the performance of SMARTY in terms of speed and maximum range in order to support higher robot speed. This will be achieved mostly by using better sensors such as single-line laser scanners or passive stereo.

Acknowledgements

We would like to thank Jeremy Armstrong, R. Craig Coulter, Jim Frazier, Joed Haddad, Alex MacDonald, Jim Moody, George Mueller, and Bill Ross for keeping the robot operational, Dirk Langer for ERIM and TCX help, and Julio Rosenblatt for DAMN.

This research was sponsored by ARPA, under contracts "Perception for Outdoor Navigation" (contract number DACA76-89-C-0014, monitored by the US Army TEC) and "Unmanned Ground Vehicle System" (contract number DAAE07-90-C-R059, monitored by TACOM). Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of ARPA or the United States Government.

References

[1] Bhatt, R., Venetsky, L., Gaw, D., Lowing, D., Meystel, A., “A Real-Time Pilot for an Autonomous Robot,” Proceedings of IEEE Conference on Intelligent Control, 1987.

[2] Chatila, R., Devy, M. Herrb, M., “Perception System and Functions for Autonomous Navigation in a Natural Environment,” Proceedings of CIRFFSS, 1994.

[3] Daily, M., Harris, J., Keirsey, D., Olin, K., Payton, D., Reiser, K., Rosenblatt, J., Tseng, D., Wong, V., “Autonomous Cross Country Navigation with the ALV,” Proceedings of the IEEE International Conference on Robotics and Automation, 1988.

[4] Dickmanns, E.D., Zapp, A. “A Curvature-Based Scheme for Improving Road Vehicle Guidance by Computer Vision,” Proceedings of the SPIE Conference on Mobile Robots, 1986.

[5] Dunlay, R.T., Morgenthaler, D.G., “Obstacle Avoidance on Roadways Using Range Data,” Proceedings of SPIE Conference on Mobile Robots, 1986.

[6] Fedor, C., “TCX, Task Communications User’s Manual. Internal Report,” The Robotics Institute, Carnegie Mellon, 1993.

[7] Feng, D., Singh, S., Krogh, B., “Implementation of Dynamic Obstacle Avoidance on the CMU Navlab,” Proceedings of IEEE Conference on Systems Engineering, 1990.

[8] Franke, U., Fritz, H., Mehring, S., “Long Distance Driving with the Daimler-Benz Autonomous Vehicle VITA,” PROMETHEUS Workshop, Grenoble, December, 1991.

[9] Gat, E., Slack, M. G., Miller, D. P., Firby, R. J., “Path Planning and Execution Monitoring for a Planetary Rover,” Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[10] Goto, Y., Stentz, A., “Mobile Robot Navigation: The CMU System,” IEEE Expert, Vol. 2, No. 4, Winter, 1987.

[11] Hebert, M., “Pixel-Based Range Processing for Autonomous Driving,” Proceedings of the IEEE International Conference on Robotics and Automation, 1994.

[12] Hebert, M.,Krotkov, E., “3D Measurements from Imaging Laser Radars,” Image and Vision Computing, Vol. 10, No. 3, April, 1992.

[13] Keirsey, D.M., Payton, D.W. and Rosenblatt, J.K., “Autonomous Navigation in Cross-Country Terrain,” in Image Understanding Workshop, Cambridge, MA, April, 1988.

[14] Kelly, A. J., “RANGER - An Intelligent Predictive Controller for Unmanned Ground Vehicles,” The Robotics Institute, Carnegie Mellon, 1994.

[15] Kluge, K., “YARF: An Open-Ended Framework for Robot Road Following,” Ph.D. Dissertation, CMU-CS-93-104, School of Computer Science, Carnegie Mellon University, 1993.

[16] Korf, R. E., “Real-Time Heuristic Search: First Results,” Proceedings of the Sixth National Conference on Artificial Intelligence, July, 1987.

[17] Langer, D., Rosenblatt, J.K., Hebert, M., “An Integrated System for Autonomous Off-Road Navigation,” Proceedings of the IEEE International Conference on Robotics and Automation, 1994.

[18] Latombe, J.-C., “Robot Motion Planning,” Kluwer Academic Publishers, 1991.

[19] Lumelsky, V. J., Stepanov, A. A., “Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment,” IEEE Transactions on Automatic Control, Vol. AC-31, No. 11, November, 1986.

[20] Matthies, L., “Stereo Vision for Planetary Rovers: Stochastic Modeling to Near Real-Time Implementation,” International Journal of Computer Vision, Vol. 8, No. 1, 1992.

[21] McTamaney, L.S., “Mobile Robots: Real-Time Intelligent Control,” IEEE Expert, Vol. 2, No. 4, Winter, 1987.

[22] Nilsson, N. J., “Principles of Artificial Intelligence,” Tioga Publishing Company, 1980.

[23] Payton, D.W., Rosenblatt, J.K., Keirsey, D.M., “Plan Guided Reaction,” IEEE Transactions on Systems Man and Cybernetics, Vol. 20, No. 6, pp. 1370-1382, 1990.

[24] Pirzadeh, A., Snyder, W., “A Unified Solution to Coverage and Search in Explored and Unexplored Terrains Using Indirect Control,” Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[25] Pomerleau, D.A., “Efficient Training of Artificial Neural Networks for Autonomous Navigation,” Neural Computation, Vol. 3, No. 1, 1991.

[26] Rosenblatt, J.K., Payton, D.W., “A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control,” in Proceedings of the IEEE/INNS International Joint Conference on Neural Networks, Washington DC, Vol. 2, pp. 317-324, June, 1989.

[27] Singh, S., Feng, D., Keller, P., Shaffer, G., Shi, W.F., Shin, D.H., West, J., Wu, B.X., “A System for Fast Navigation of Autonomous Vehicles,” Robotics Institute Technical Report CMU-RI-TR-91-20, Carnegie Mellon University, 1991.

[28] Stentz, A., Brumitt, B.L., Coulter, R.C., Kelly, A., “An Autonomous System for Cross-Country Navigation,” Proceedings of the SPIE Conference on Mobile Robots, 1992.

[29] Stentz, A., “Optimal and Efficient Path Planning for Partially-Known Environments,” Proceedings of the IEEE International Conference on Robotics and Automation, 1994.

[30] Thorpe, C., Amidi, O., Gowdy, J., Hebert, M., Pomerleau, D., “Integrating Position Measurement and Image Understanding for Autonomous Vehicle Navigation,” Proceedings of the Workshop on High Precision Navigation, Springer-Verlag Publisher, 1991.

[31] Wilcox, B., Matthies, L., Gennery, D., Cooper, B., Nguyen, T., Litwin, T., Mishkin, A., Stone, H., “Robotic Vehicles for Planetary Exploration,” Proceedings of the IEEE International Conference on Robotics and Automation,1992.