Global Path Planning on Board the Mars Exploration Rovers

Joseph Carsten and Arturo Rankin
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
{joseph.carsten,arturo.rankin}@jpl.nasa.gov

Dave Ferguson and Anthony Stentz
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{dif, tony}@cmu.edu

Abstract— In January 2004, NASA's twin Mars Exploration Rovers (MERs), *Spirit* and *Opportunity*, began searching the surface of Mars for evidence of past water activity. In order to localize and approach scientifically interesting targets, the rovers employ an on-board navigation system. Given the latency in sending commands from Earth to the Martian rovers (and in receiving return data), a high level of navigational autonomy is desirable. Autonomous navigation with hazard avoidance (AutoNav) is currently performed using a local path planner called GESTALT (Grid-based Estimation of Surface Traversability Applied to Local Terrain). GESTALT uses stereo cameras to evaluate terrain safety and avoid obstacles. GESTALT works well to guide the rovers around narrow and isolated hazards, however, it is susceptible to failure when clusters of closely spaced, non-traversable rocks form extended obstacles. In May 2005, a new technology task was initiated at the Jet Propulsion Laboratory to address this limitation. A version of the Carnegie Mellon University Field D* global path planner has been integrated into MER flight software, enabling simultaneous local and global planning during AutoNav. A revised version of AutoNav was uploaded to the rovers during the summer of 2006. This paper describes how global planning was integrated into the MER flight software, and presents results of testing the improved AutoNav system using the MER Surface System TestBed rover.

Keywords— MER, robotics, Mars rover, flight software, autonomous navigation, path planning, Field D*

TABLE OF CONTENTS

1-4244-0525-4/07/$20.00©2007 IEEE
IEEEAC paper #1125

Figure 1. Artist's rendition of a Mars Exploration Rover. *Courtesy NASA/JPL-Caltech.*

1. INTRODUCTION

In January 2004, two robotic vehicles landed on Mars as part of NASA's Mars Exploration Rover (MER) mission (see Figure 1). Since that time, these two rovers, *Spirit* [1] and *Opportunity*, [2], have been searching the Martian surface for evidence of past water activity. Directing rover activities poses an interesting challenge for scientists and engineers. It can take as long as 26 minutes for a signal from Earth to reach Mars (and vice-versa). This makes teleoperation of the rovers infeasible. In addition, line-of-sight and power constraints further complicate the situation. In order to overcome these factors, each rover is sent a sequence of commands at the beginning of each Martian day (sol). This command sequence lays out all activities to be performed by the rover during the sol. The rover then executes the command sequence without any human intervention. In general, before the rover shuts down for the night, it will send data back to Earth. This data is then used to plan activities for the following sol. Due to the fact that commands are received only once per sol, rover autonomy is critical. The more autonomous the rover is, the more activities it can accomplish each sol. Here we will focus our attention on the navigation system, but this observation applies to all rover behaviors.

The purpose of the navigation system is to move the rover around the Martian surface in order to locate and approach scientifically interesting targets. To begin the process, engineers on Earth identify a goal location that they would like the rover to reach. Typically, images returned by the rover are used to select this goal. There are two main methods that can be used to reach this goal. The first and simplest is the blind drive. During a blind drive, the rover does not attempt to identify hazardous terrain and simply drives toward the goal location. The second option is autonomous navigation with hazard avoidance (AutoNav). In this case, the rover autonomously identifies hazards, such as large rocks, and steers around them on its way to the goal.

There are advantages and disadvantages to each approach. During a blind drive, the rover can cover a larger distance in a given time period since it does not have to process imagery of the surrounding terrain. However, this means that the engineers on Earth must verify that the terrain between the rover and the goal is free from hazards before commanding the drive. On the other hand, AutoNav is slower, but can keep the rover safe even in regions unseen by engineers on Earth. Often, the two methods are utilized in tandem. First a blind drive is commanded as far out as engineers can be sure of safety. Then AutoNav is used to make additional progress through unknown terrain. Thus, the increased autonomy provided by AutoNav allows much more forward progress to be made during a sol.

Although AutoNav is usually able to guide the rover to the goal, there are known circumstances where it is susceptible to failure, and the rover does not reach the goal. In July 2006, a new version of the MER flight software was successfully uploaded to the rovers. Due to the complexity and number of changes, a software patch was infeasible and a full flight software load was necessary [3]. In addition to bug fixes and other improvements, four new technologies were included. These new technologies were visual target tracking, on-board dust devil and cloud detection, autonomous placement of the instrument deployment device, and a global path planner designed to overcome some of the shortcomings of AutoNav [4]. This planner and its integration into the flight software are described below.

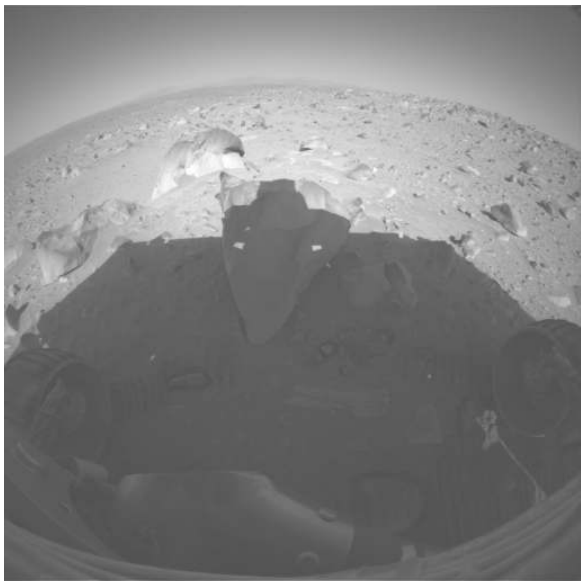


Figure 2. On sol 108, *Spirit* was unable to autonomously navigate to a goal location on the other side of this cluster of rocks. This image was captured by one of the front hazard avoidance cameras mounted on the body of the rover. *Courtesy NASA/JPL-Caltech.*

2. AUTONOMOUS NAVIGATION SYSTEM

Overview

The purpose of AutoNav is to enable the rover to safely traverse unknown terrain. AutoNav is based on the GESTALT (Grid-based Estimation of Surface Traversability Applied to Local Terrain) algorithm [5], [6]. AutoNav uses stereo image pairs captured by the rover's on-board camera system to gather geometric information about the surrounding terrain. These images are processed to create a model of the local terrain. Part of this model is a goodness map. This goodness map is grid based and represents an overhead view of a model of the terrain. Each grid cell in the map contains a goodness value. High goodness values indicate easily traversable terrain, and low goodness values indicate hazardous areas. The map is constructed in configuration space, meaning that hazards are expanded by the rover radius in all directions before their representations are included the goodness map. This allows the rover to be treated as a point in future computations.

Once the terrain has been evaluated, a set of candidate arcs (short paths from the current rover location) is considered. Nominally, the arc set consists of forward and backward arcs of varying curvature, as well as point turns to a variety of headings. Each arc is evaluated based on three criteria. These are avoiding hazards, minimizing steering time, and reaching the goal. For each arc, a vote based on each of these criteria is generated. The goodness map is used to generate the hazard avoidance vote. Arcs that travel through cells that are difficult or dangerous to traverse receive low votes. Steering bias votes are constructed based on the amount of time that is needed to turn the wheels from the current heading to the heading required to execute the candidate arc. Arcs requiring less steering time receive higher votes. Waypoint votes are constructed based upon the final criteria: reaching the goal. Arcs that move the rover closer to the goal location receive higher waypoint votes. The three votes are then weighted and merged to generate a final vote for each arc. Once votes have been generated, the best arc is selected for execution. The rover then drives a short, predetermined distance along the selected arc . This process is repeated (evaluate terrain, select arc, drive) until the goal is reached, a prescribed timeout period expires, or a fault is encountered.

Shortcomings

AutoNav is very good at keeping the rover safe and usually gets the rover to the goal location. However, in some in-

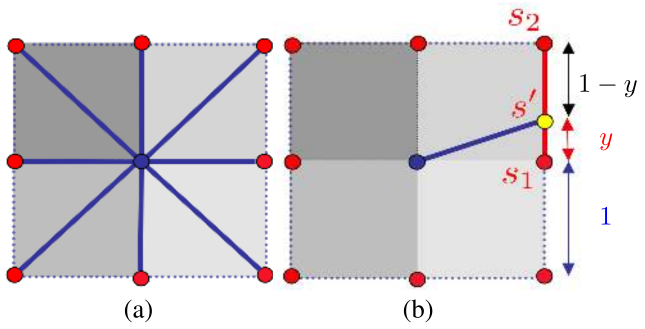


Figure 3. (a) The typical transitions (in blue) allowed from a node (shown at the center) in a uniform grid. Notice that only headings of 45 degree increments are available. (b) Using linear interpolation, the path cost of any point s' on an edge between two grid nodes s_1 and s_2 can be approximated. This can be used to plan paths through grids that are not restricted to just the 45 degree heading transitions.

stances AutoNav is not able to reach the goal. Figure 2 illustrates one such situation. In this case, *Spirit* spent approximately 105 minutes trying to get around a cluster of rocks, but was unable autonomously do so. Forty-seven drive steps were taken during the attempt. The simple method used to construct the waypoint votes leads to this problem. Arcs that decrease the Euclidean distance between the rover and the goal always receive higher votes. Therefore the rover will attempt to take a straight-line path to the goal. When the waypoint votes are merged with hazard avoidance votes, some deviation to get around small hazards can occur. However, the amount of deviation that can occur is fairly minimal. When the rover encounters a large hazard in its path, the waypoint votes and hazard avoidance votes conflict severely. The hazard avoidance votes will not allow the rover to drive through the unsafe area, and the waypoint votes will not allow enough deviation from the straight-line path for the rover to get around the hazard. The rover becomes stuck and is unable to reach the goal.

3. Global Path Planning

For improved performance, a better waypoint vote metric is needed; something that is more accurate than Euclidean distance. A better metric can be produced by planning paths to the goal that take into account all of the obstacles in the environment. Typically, the environment will be only partially-known to the rover, and thus complete information regarding the obstacles will not be available. However, incorporating obstacle information that *is* available into these global plans typically provides much better estimates than Euclidean distance, and these estimates only improve in accuracy as more information is acquired during the rover's traverse.

The AutoNav system has been extended to use the Field D* algorithm to generate these global paths. Field D* is a planning algorithm that uses interpolation to provide direct, low-cost paths through two-dimensional, grid-based representations of an environment [7]. Each grid cell is assigned a cost

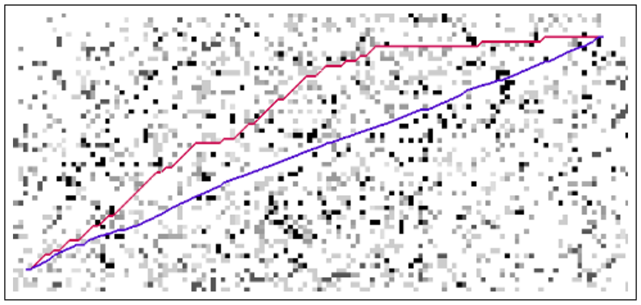


Figure 4. Paths produced by classic grid-based planners (red/top) and Field D* (blue/bottom) in a 150×60 uniform resolution grid. Darker cells represent higher-cost areas.

of traversal. Based upon these costs, the algorithm generates a path between two locations, with the aim of minimizing the cost of traversing that path.

Although two-dimensional grids present an easy and computationally efficient way to represent the environment, a major limitation of classic grid-based planning algorithms is the restricted nature of the paths produced. For example, classic grid-based planners usually restrict paths to transitioning between adjacent grid cell centers or corners, resulting in paths that are suboptimal in length and involve unnecessary turning. Figure 3(a) shows the typical transitions allowed from a particular grid cell.

The Field D* algorithm removes this restriction and allows paths to transition through any point on any neighboring grid cell edge, rather than just the neighboring grid cell corners or centers. To do this efficiently, it uses linear interpolation to approximate the path cost to any point along a grid cell edge, given the path costs to the endpoints. Equation 1 and Figure 3(b) illustrate how linear interpolation is used to provide an estimate of the path cost to an edge node s' given the path costs to end nodes s_1 and s_2. Here y is the distance between s_1 and s', measured as a fraction of the length of a grid cell side.

$$\begin{aligned} PathCost(s') \quad \approx \quad & y \cdot PathCost(s_2) + \\ & (1-y) \cdot PathCost(s_1) \end{aligned} \tag{1}$$

As a result, Field D* is able to provide much more direct, less-costly paths than standard grid-based planners without sacrificing real-time performance. It is also able to efficiently repair its solutions as new information is received, for example through onboard sensors. Figure 4 shows a path planned by Field D* along with the classic grid-based path.

4. Integration

At the highest level, using Field D* to improve AutoNav involves two main tasks. The first is providing terrain information to Field D* in a form it can utilize. The second is using

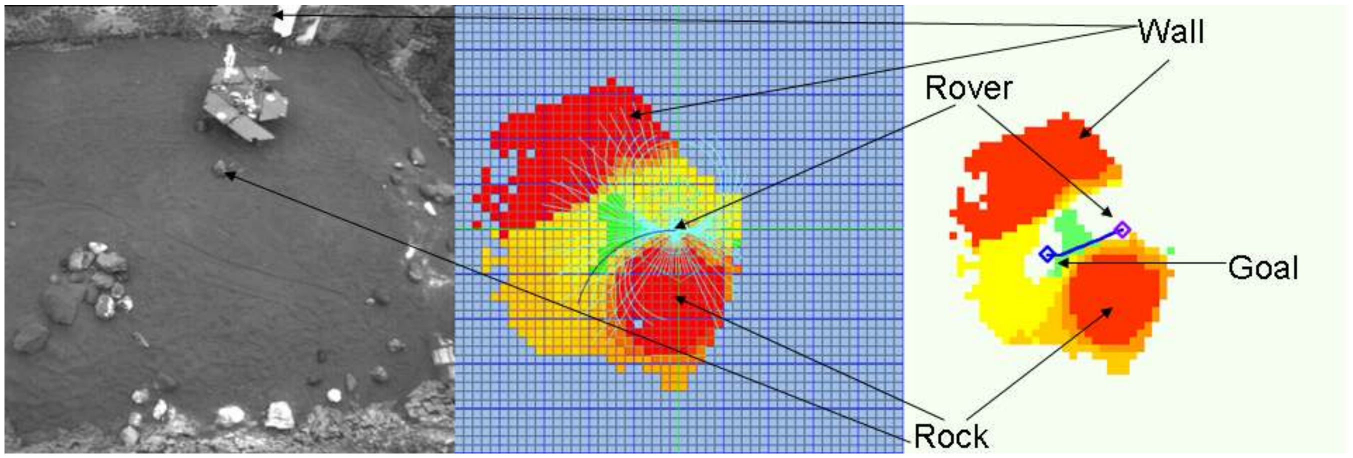


Figure 5. The left image is an overhead view of the rover. The middle image is the corresponding goodness map, and the Field D* cost map is shown in the right image. Blue cells have unknown traversability. All other cells are colored based on a gradient between green (high goodness/low cost) and red (low goodness/high cost). Note that the entire goodness map is presented, but only a small portion of the cost map is shown in here.

Field D* to generate steering recommendations in a form that AutoNav can understand.

Cost Map

Field D* uses a uniform grid as the basis of its world model. Each grid cell contains a value which represents the cost of traversing the width of the cell. Fortuitously, this is very similar to the goodness map representation of the world maintained by AutoNav. However, the goodness map is always centered on the rover location, and stores only information about the local terrain. Field D* plans on a global scale and must therefore store a much larger map. In addition, the Field D* map is fixed to the environment and does not move along with the rover. There are several other key differences between the two representations as well. Field D* operates on cost values, where more easily traversable terrain has a lower cost, but AutoNav stores a goodness map, where more easily traversable terrain has a higher goodness. In addition, grid cells in the goodness map can have "unknown" goodness. This indicates that there is not enough information about that cell location to determine its traversability. The Field D* cost map has no such value. All cells must be assigned a cost of traversal from the start.

Using the goodness map to update the cost map is fairly straightforward. First, because there is no notion of unknown cost, the entire cost map must be initialized to a given cost value. Initializing all cells to a low cost means the rover will be much more inclined to explore unseen regions. On the other hand, initializing to a high cost means that the rover will prefer to stay in regions it has already seen. Here, a midrange cost value was chosen. Next, at each step of the traverse, the position of the goodness map inside the larger cost map is determined. Then each goodness cell that is not unknown is merely translated into a cost value, and placed into the corresponding cost grid cell. For this to operate correctly, the goodness grid cells and cost grid cells must be the same size. In addition, grid cell boundaries in the goodness map must align with those in the cost map. These issues are addressed when the maps are created. Each goodness value is translated into a cost value as follows. Cells with very low goodness are set to a special cost value representing obstacle. Field D* will not plan paths through these cells. All other goodness values are inverted and then scaled to the range of cost values to produce corresponding costs. By virtue of its much larger map, Field D* tracks everything the rover has seen, even when it has been long forgotten by the local goodness map. Figure 5 shows a goodness map and the corresponding portion of the Field D* cost map.

Votes

Once the cost map has been populated, a method is needed to use Field D* to influence arc selection. The output of Field D* is the cost of traversing the optimal path from any query point to the goal location. However, the easiest way to provide steering recommendations to the rest of the system is through arc votes. Therefore, a way to convert costs of traversal to arc votes is necessary.

To begin this process, Field D* is used to compute the cost of traversal from the end of each candidate arc to the goal. Taken individually, these traversal costs mean very little. If the rover is 50 meters from the goal, the traversal cost for a given arc will be much higher than if the rover is 10 meters from the goal. This is merely due to the fact that there is much more ground to cover in the first case. Fundamentally, arc votes are just a way of ranking the candidate arcs from best to worst. When taken relative to each other, the traversal costs provide a similar ranking mechanism. The arc with the lowest cost of traversal to the goal is the best and the one with the highest cost is the worst. Numerical vote values are assigned using a weighted sum of v_{scale} and v_{close}, which are given in

Equations 2 and 3. v_{max} is the maximum possible vote, c_{max} and c_{min} are the maximum and minimum traversal costs for the current arc set evaluation, and c_i is the traversal cost for a given arc. The minimum vote value is zero.

$$v_{scale_i} = v_{max} * (c_{max} - c_i)/(c_{max} - c_{min}) \quad (2)$$
$$v_{close_i} = v_{max} * c_{min}/c_i \quad (3)$$

v_{scale} is a standard linear scaling of the cost values into vote values. v_{close} bases vote values upon how close the rover is to the goal. The closer the rover is to the goal, the greater the range of vote values that is generated. When the rover is far from the goal, c_{min}/c_{max} will be close to one and all votes will be close to v_{max}. On the other hand, when the rover is close to the goal, c_{min}/c_{max} will be close to zero, and the votes will be spread from zero to v_{max}. Alone, v_{close} is not particularly useful (especially when the rover is far from the goal), but when combined with v_{scale} it can be helpful. When combined with v_{scale}, v_{close} serves to reduce the range of vote values when the rover is far from the goal. This means the preference for one arc over another is less pronounced. When the rover is far from the goal, it is not critical exactly which arc is taken (as long as the rover is moving in generally the right direction). In this case it may be advantageous to let the other voting modules (steering bias and hazard avoidance) have more influence over the final arc selection. However, as the rover gets closer to the goal, the exact arc selected is more important and thus the entire range of vote values is utilized. Generally when combining the two vote values, v_{scale} receives a significantly higher weight than v_{close}.

Once these votes have been constructed, they replace the waypoint votes constructed by GESTALT. They are then combined with steering bias and hazard avoidance votes in order to select the arc that will be followed. When constructing Field D* votes, it is possible that several arcs may have identical costs of traversal. Nothing special need occur to handle this situation. These arcs are merely assigned equal Field D* vote values. The GESTALT arc selection algorithm handles combining these votes with steering bias and hazard avoidance votes, as well as breaking any ties that might occur in the final combined vote values. Once the naive waypoint votes are replaced with those generated using Field D*, the autonomous navigation system becomes much more robust.

Limitations

It should be noted that AutoNav (both with and without Field D*) assumes that the rover position is known, and that candidate arcs can be executed nominally. There are cases in which these assumptions are violated. For instance, on sandy slopes the wheels may slip significantly, causing the estimated rover position to be erroneous. In addition, mechanical failure of wheel actuators can cause arcs to be executed abnormally. In these cases, AutoNav performance may be degraded. Although AutoNav makes no attempt to directly address these issues, other technologies can often be used to overcome them. For instance, visual odometery can be used in conjunction with AutoNav in order to maintain an accurate estimate of rover position, regardless of wheel slip [8].

5. Resource Limitations

The Mars rovers are constrained by very limited computational resources. The onboard computer uses a radiation hardened RAD6K processor running at 20 MHz, and has 128 Mbytes of DRAM [6]. To make matters worse, these already limited resources must be shared among the 97 tasks (including AutoNav) that make up the on-board flight software [9]. In light of these constraints, optimizations were made to the Field D* algorithm to improve efficiency. Specifically, the path cost minimization step of the algorithm is pre-computed, and the results are stored in a lookup table that is then accessed at runtime. This significantly decreases the computation time required for planning. See [7] for more details on how this is performed.

Another constraint is the limited bandwidth available to send data back to Earth. This data can be grouped into two broad categories: engineering data and science data. Science data contains information about Mars that is of interest to scientists. Engineering data is used to monitor the status of the rover, and contains information that is useful should an anomaly occur. Telemetry generated by Field D* falls into this category. Since the main purpose of the mission is to better understand Mars, it is desirable to limit the engineering data to a minimum in order to maximize the amount of science data that can be downlinked.

In the case of Field D*, CPU utilization, memory usage, and telemetry volume are all tied to the size of the Field D* cost map. Larger maps mean more resource usage. Therefore, it is advantageous to find ways to reduce the map size while still obtaining good path planning results.

Automatic Recentering

In order for Field D* to plan a path between some start location and a goal, both the start location and goal location must be located within the cost map. Therefore, the further the rover is from the selected goal, the larger the map must be. In order to allow long traverses that would be infeasible due to memory constraints, a scheme was developed to overcome this limitation. The constraint that the user selected goal must be within the bounds of the cost map is lifted, and any arbitrary goal location is allowed. If the goal happens to be outside of the cost map, an intermediate goal is selected that resides on the boundary of the cost map. The intermediate goal is placed at the point where the straight line between the current rover location and user selected goal intersects the boundary of the cost map. However, this does not completely solve the problem. Now the rover is being guided to a point on the edge of the map and not the real goal. In order for the rover to reach the real goal, map recentering is needed. Dur-

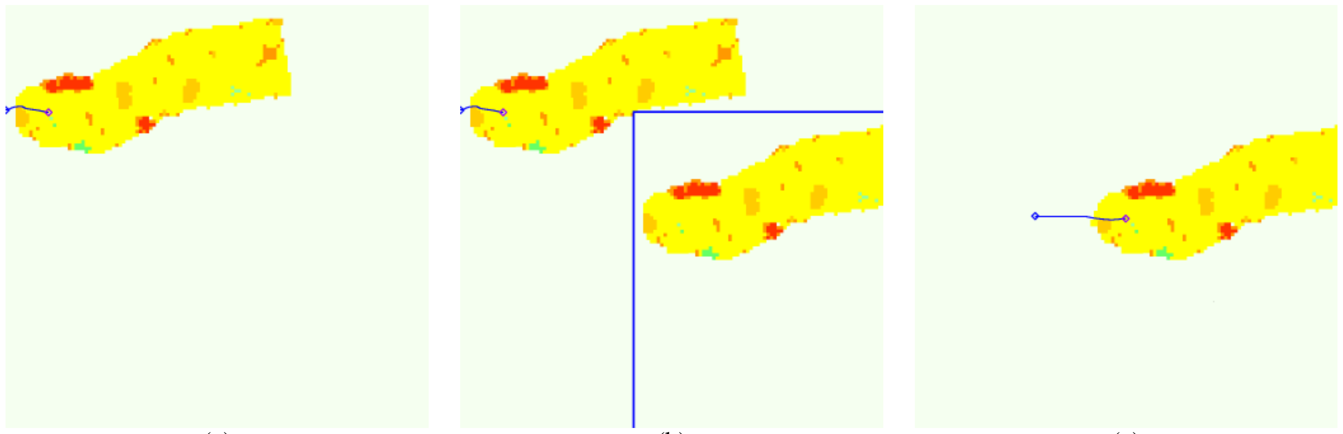

(a) (b) (c)

Figure 6. Field D* cost map and the recentering process. The rover is represented by a purple diamond and the goal is shown as a blue diamond. Recentering is needed for the map shown in (a). Cells common to the old and new map are copied to their new location in the lower right of (b). Shown in (c) is the final map after cells not common to both the old and new maps have been cleared, the cost map has been updated from the most recent goodness map, and the goal location has been recalculated.

ing the map update phase, if any portion of the local goodness map falls outside the cost map, the cost map is recentered on the current rover position.

Recentering does not alter any memory allocations, but instead merely adjusts the world coordinates of the map center. In order to make the map consistent with the new coordinates, grid cells that are common to both the old map and new map are copied across the map to their new location. All other areas are cleared to the nominal cost value. Once this is done, a new goal is placed within the cost map. If the user selected goal is within the new map bounds, the goal is placed there. If not, another intermediate goal is placed using the procedure outlined earlier. This recentering and intermediate goal placement is repeated until the user selected goal is reached. Figure 6 illustrates the recentering process.

This approach has some limitations. There is a performance penalty whenever the map is recentered. Field D* is efficient because it does not have to replan from scratch when new costs are discovered. Instead, it is able to reuse the results of previous planning and repair the needed paths. However, because Field D* begins its search at the goal, whenever the goal is moved, all planning information is reset and the next path must be planned from scratch. In order to minimize this effect, the intermediate goal is not updated every time the rover moves. Instead, a new goal is placed only when the map is recentered. Map recentering is an infrequent event and therefore the overall performance impact is minor.

There is another limitation to this approach. It is possible that the intermediate goal could be placed inside an obstacle. When the goal is inside an obstacle, Field D* is unable to plan any paths and will fail. However, this problem is unlikely to be encountered in practice. Usually, the intermediate goal is ahead of the rover, in an area not yet visited. Because the rover has not seen the terrain around the intermediate goal, that location cannot be obstacle in the cost map until the rover is close enough to evaluate that terrain. The map is recentered slightly before the rover can see the edge of the map. Therefore, in general, the rover has never evaluated the terrain under any current intermediate goal. The exception is if the rover in the process of backtracking a significant distance in order to navigate around a very large hazard. In this case, the goal is behind the rover and the rover is driving away from it. Therefore, when the map is recentered, the rover may have already seen the region where the new intermediate goal is placed, and it is possible that there is an obstacle in this region. However, for this problem to occur, the rover must be attempting to navigate around a very large hazard, and must drive large distances. Due to power and time constraints, the distance that the rover can traverse in a single sol using AutoNav is limited. This limitation drastically reduces the chances of encountering the problem.

Coarse Resolution Cost Maps

Another way to manage limited memory resources is to change the resolution of the cost map grid cells. Instead of constraining the cost map grid cells be the same size as the goodness map grid cells, the cost map cells are allowed to be larger. This allows fewer grid cells to cover the same area, and thus for smaller cost maps in terms of grid cells, which is what dictates resource usage. Further, because the Field D* planner is able to compute paths that are not restricted to transitioning between grid cell centers or corners, it can be used to plan direct, low-cost paths even in very coarse resolution grids.

Allowing for larger cost map grid cells does present some complications. Updating the cost map from the goodness map is now more difficult. Before, there was a one-to-one correspondence between cost and goodness grid cells, meaning

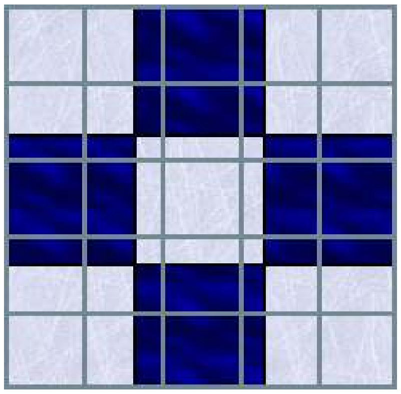


Figure 7. Goodness map overlain on a cost map. Goodness grid cells are outlined in grey. Cost cells are shown in white and blue. In addition to a complete goodness cell, each cost cell contains pieces of 3, 5, or even 8 other goodness cells.

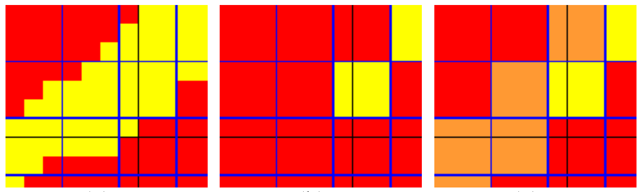


Figure 8. Here, each cost cell contains nine goodness cells (cost cells are outlined in blue). Red represents obstacle, yellow is traversable, and orange is the maximum traversable cost. Coloring is by goodness value in (a) and cost value in (b) and (c). The cost map in (b) is produced by using the minimum goodness value in each cost cell. The cost map in (c) is produced using a more lenient update rule. If the cost cell is less than half obstacle it is set to the maximum traversable cost instead of obstacle. Note that the corridor is blocked in (b), but not in (c).

that the goodness map could essentially be copied directly into the cost map. With larger cost cells, there are multiple goodness cells in each cost cell. In fact, there could even be fractional goodness cells in a given cost cell as shown in Figure 7. In order to simplify matters somewhat, the size of the cost cells are constrained to be an integer multiple of the goodness cells. This avoids splitting single goodness cells across multiple cost cells. Instead, each cost cell contains a fixed number of whole goodness cells. In this way, the complication of dealing with fractional cells can be avoided. However, a method is still needed to convert multiple goodness values into a single cost value.

One simple and safe method would be to use the minimum goodness value in a given cost cell to set the cost. In some cases this is not necessarily the best approach. Figure 8 illustrates what can happen when a narrow corridor is encountered. By using the minimum goodness value to update the cost value, narrow corridors in the goodness map can become completely blocked in the cost map. One way to mitigate this problem is to employ a more lenient standard when updating cost cells containing obstacles. Cost cells that are less than half obstacle are set to the maximum traversable cost. Cost cells that are half obstacle or more are set to obstacle. This greatly reduces the chances of closing off narrow corridors.

Larger cost map grid cells also require a new strategy for handling unknown goodness cells. There is no traversability information in these cells, and previously they could just be ignored. The situation is more complicated when there are multiple goodness cells in each cost cell. One option for handling unknown goodness cells is to not update cost cells containing *any* unknown goodness cells. This is a less than ideal solution. A cost cell could contain many goodness cells with known values, but if there is one unknown value, all this information will be ignored. This situation happens frequently at the edge of the field of view. In order to fully utilize the terrain assessment, the unknown goodness cells could merely be ignored, and the minimum goodness of the populated goodness cells used to update the cost cell. This approach presents a more subtle problem which is illustrated in Figure 9. During each step the rover takes, an area around the edge of the goodness map is set to unknown. This erases old data behind the rover in order to make room for new data in front of it. The problem arises when obstacle goodness cells behind the rover are set to unknown, but there are still some goodness cells in a given cost cell that have not been cleared and are not obstacle. The minimum goodness is therefore no longer obstacle, and if the minimum goodness is used to update the cost cell, the cost will be changed from obstacle to traversable. This causes Field D* to forget about obstacles, which is highly undesirable.

The solution to these problems is to make use of another value that is stored as part of the local terrain map. In addition to a goodness value, each goodness cell contains a certainty value as well. If the certainty is not zero, then the grid cell is in the current field of view and was just updated. It acts as a sort of new data flag. Therefore, if there is no certainty in a given cost cell, it is probably an old cell that is behind the rover. In this case, the first approach is utilized, which is to not update the cell if it has any unknown goodness. This avoids forgetting data in the cost map. On the other hand, if there is certainty in a given cost cell, it contains new data and is probably in an area that hasn't been seen before. For these cells the second approach is used, and the cost is updated using the minimum goodness value. In this way, new terrain assessments are added to the cost map as early as possible.

Map Filtering

In certain situations, the planning process for a given drive step can take more than an order of magnitude longer than usual. Recall that Field D* is used to plan a path from each arc endpoint to the goal. Also recall that Field D* will not plan paths through obstacle cells. If an arc endpoint happens

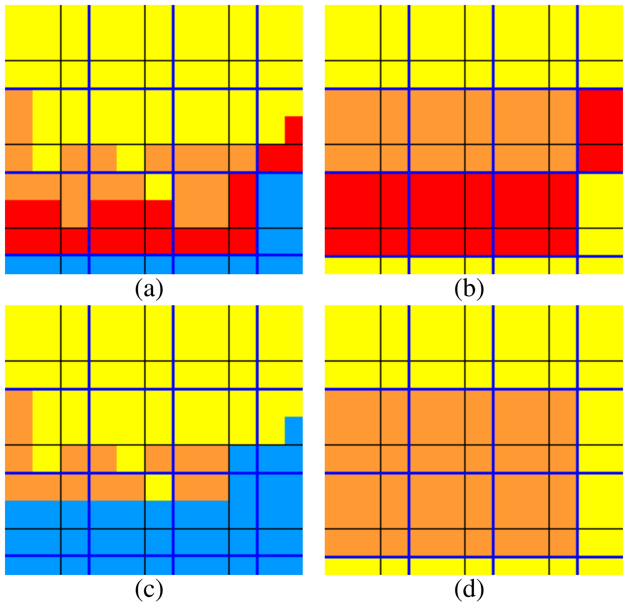

(a) (b) (c) (d)

Figure 9. Here, each cost cell contains nine goodness cells (cost cells are outlined in blue). Red represents obstacle, yellow and orange are traversable, and blue is unknown. The goodness and cost maps for one step are shown in (a) and (b) respectively. Similarly, (c) and (d) are for the next step. Between steps, the rover has moved up to the left. The cost maps are produced using the minimum goodness value that is not unknown. Note that as the rover moves, obstacles are forgotten from the goodness map, and using this update strategy these obstacle cells are set to traversable in the cost map.

to fall in an obstacle cell, the algorithm immediately returns, indicating that there is no path to the goal. However, planning time can swell if an arc endpoint falls in a non-obstacle region completely surrounded by obstacle cells, as shown in Figure 10(a). This is an artifact of how the planning process is carried out. The search begins at the goal location and expands outward. When the start state (the arc endpoint in our case) is reached, the search terminates and the path cost is returned. Unfortunately, when an arc endpoint falls into a small region completely surrounded by obstacles, it is in fact unreachable. In order for Field D* to make this determination, every reachable state in the cost map must be expanded. Usually relatively few states need to be expanded, and thus expanding the entire map leads to a significant increase in planning time. This explosion in usage of already limited CPU resources is very undesirable.

In order to solve this problem, the goodness map is filtered before it is used to update the Field D* cost map. A flood fill algorithm is used to identify all cells in the goodness map reachable from the current rover location. The rover location is first marked as reachable. Then each adjacent (eight-connected), non-obstacle cell is added to a list for later processing. Next, a cell is removed from the list, marked as reachable, and its new non-obstacle neighbors are added to

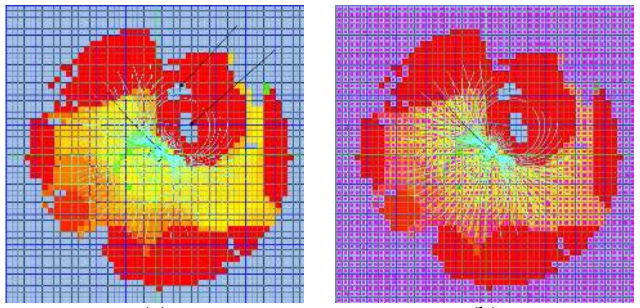

(a) (b)

Figure 10. Goodness map filtering. Obstacle cells are shown in red. The goodness map in (a) contains regions completely surrounded by obstacle cells. Planning time is greatly increased when arc endpoints fall in these regions. All cells reachable from the rover location are shaded pink in (b). Note that the regions surrounded by obstacle are not marked as reachable.

the list. This repeats until the list is empty, indicating that all cells reachable from the rover location have been identified, as shown in Figure 10(b). Finally, all non-reachable, non-obstacle cells are set to obstacle. By doing this, arc endpoints that would have been problematic now end in obstacle cells. In this case, no planning is necessary to determine that no path to the goal exists.

Even though map filtering is done during every map update, in the long run it still saves time. As an added benefit, it also allows for much more consistent and predictable planning times. There are a couple of reasons why map filtering at every step is much faster than letting Field D* occasionally expand the entire cost map. First, map filtering is done on the goodness map, which is much smaller than the Field D* cost map. In addition, the simple flood fill check for reachability takes much less time than the full Field D* planning process. In fact, the time necessary to filter the goodness map is negligible when compared even to the nominal planning time necessary for each drive step.

6. RESULTS

The MER Surface System TestBed (SSTB) was used to extensively test flight software modifications. The SSTB is a high-fidelity engineering model of the Mars Exploration Rovers. It is essentially identical in form and electromechanical function to *Spirit* and *Opportunity*, with a few minor exceptions. The SSTB has no solar panels, and some of its electronics are housed in an adjacent clean room. A physical tether provides a link between the rover and these electronics. The tether also provides power to the rover. The SSTB is housed in an indoor sandbox approximately 9 meters wide and 22 meters long [10]. A ramp tilted at 25 degrees occupies one end. The SSTB is shown in Figure 11.

GESTALT alone performs well in simple situations, including navigation in areas free from hazards and navigating around small discrete obstacles. However, the real strength

Figure 11. MER Surface System TestBed rover.

of Field D* is navigation in much more complex situations. Unfortunately, the relatively small size of the sandbox makes constructing complex obstacle arrangements difficult. Testing was limited to this environment for several reasons. The SSTB was the only available system with enough fidelity to perform flight software testing requiring imaging and driving, with the driving decisions based upon imaging results. It was infeasible to move the rover to a larger outdoor environment due to the tremendous effort that would be necessary to move all the support equipment needed to run the rover (remember that most of the rover electronics are actually housed in a clean room adjacent to the sandbox). However, even in the limited sandbox environment, constructing situations for which GESTALT alone fails to reach the goal is not difficult. For instance, navigating around a cul-de-sac obstacle arrangement is nearly impossible for GESTALT alone. Figure 12 illustrates a situation with not one, but two cul-de-sacs. Due to the limited size of the sandbox, the goal is placed outside the sandbox and is not actually reachable. Figure 12(a) shows the initial position of the rover. The rover begins by driving straight into the first cul-de-sac. The rover reaches the bottom of the cul-de-sac in Figure 12(b). Up to this point, behavior with and without Field D* was roughly equivalent. However, with GESTALT alone the rover became stuck here. Field D* on the other hand, plans a path around the first cul-de-sac and into the second. The rover is then guided into the second cul-de-sac as shown in Figure 12(c). Once the determination is made that there is no route through the second cul-de-sac, the rover drives back toward the only unexplored region of the sandbox as shown in Figure 12(d). Eventually Field D* fails, indicating that no paths to the goal exist.

Over the course of testing, Field D* was used to guide the SSTB toward roughly 100 different goal locations. Initially, a variety of simple tests were completed. In an obstacle free setting, the rover was placed at a variety of different initial headings relative to the straight line to the goal. Navigation through a field of traversable rocks was tested. Navigation around a single rock in various positions relative to the path between the rover and the goal, and navigation between two rocks separated by a variety of distances were also tested. More complex obstacle arrangements in which GESTALT alone would almost certainly fail to guide the rover to the goal were tested as well. Situations were constructed necessitating navigation into and out of single or multiple cul-de-sacs. In addition, lines of rocks were used to produce an arrangement similar to the one shown in Figure 2. Overall, the performance was extremely good. In the vast majority of cases the rover was able to reach the goal when Field D* was used, and performance in the simple test cases was at least as good as with GESTALT alone. Surprisingly, one of the biggest problems faced during testing was goal placement. If the goal is placed in an obstacle cell, Field D* is unable to plan any paths. When the rover gets close enough to the goal to determine it is in an obstacle cell, Field D* will fail. Although the rover does not reach the goal, this should not necessarily be considered an AutoNav failure. In these situations the goal location is not safe, and the rover should not drive onto it. Due to the very limited space in the sandbox, squeezing the goal location into a safe area (after all obstacles have been expanded by the rover radius) was sometimes a challenging proposition.

With Field D*, the rover is able to explore the environment much more fully when attempting to locate a path to the goal. This allows the rover to almost always arrive at reachable goals. The downside, of course, is the increased resource utilization required. For Field D*, the additional CPU time and memory usage are fairly minimal. Much of the testing was done using 50 m x 50 m cost maps. The cost cells were 40 cm x 40 cm, which is twice as big as the goodness cells. Almost no difference was noticed in rover behavior when moving from 20 cm to 40 cm cost cells. With these settings, Field D* utilizes less than 1 Mbyte of memory. In addition, each drive step takes only about 3 percent longer when Field D* is enabled. Even with these very modest requirements, Field D* is able to significantly improve on-board autonomous navigation capability.

7. Conclusions

Autonomous hazard avoidance using GESTALT keeps the rovers safe and works well in the presence of simple discrete obstacles. However, it is susceptible to failure when more complex hazard arrangements are encountered. In order to address this shortcoming, the hazard avoidance system was augmented with a global path planner. Field D* was integrated into the MER flight software and uploaded in the summer of 2006 as part of a significant software upgrade. Field D* assisted hazard avoidance was extensively tested using the SSTB before the upload. Obstacle avoidance was at least as good as with GESTALT alone, and in many cases much better. Field D* allows the rover to much more robustly navigate around hazards. With Field D*, the rover is less prone to getting stuck and reaches the goal even when faced with complex hazards.

8. Acknowledgements

The research described in this paper was performed, in part, at the Jet Propulsion Laboratory, California Institute of Tech-

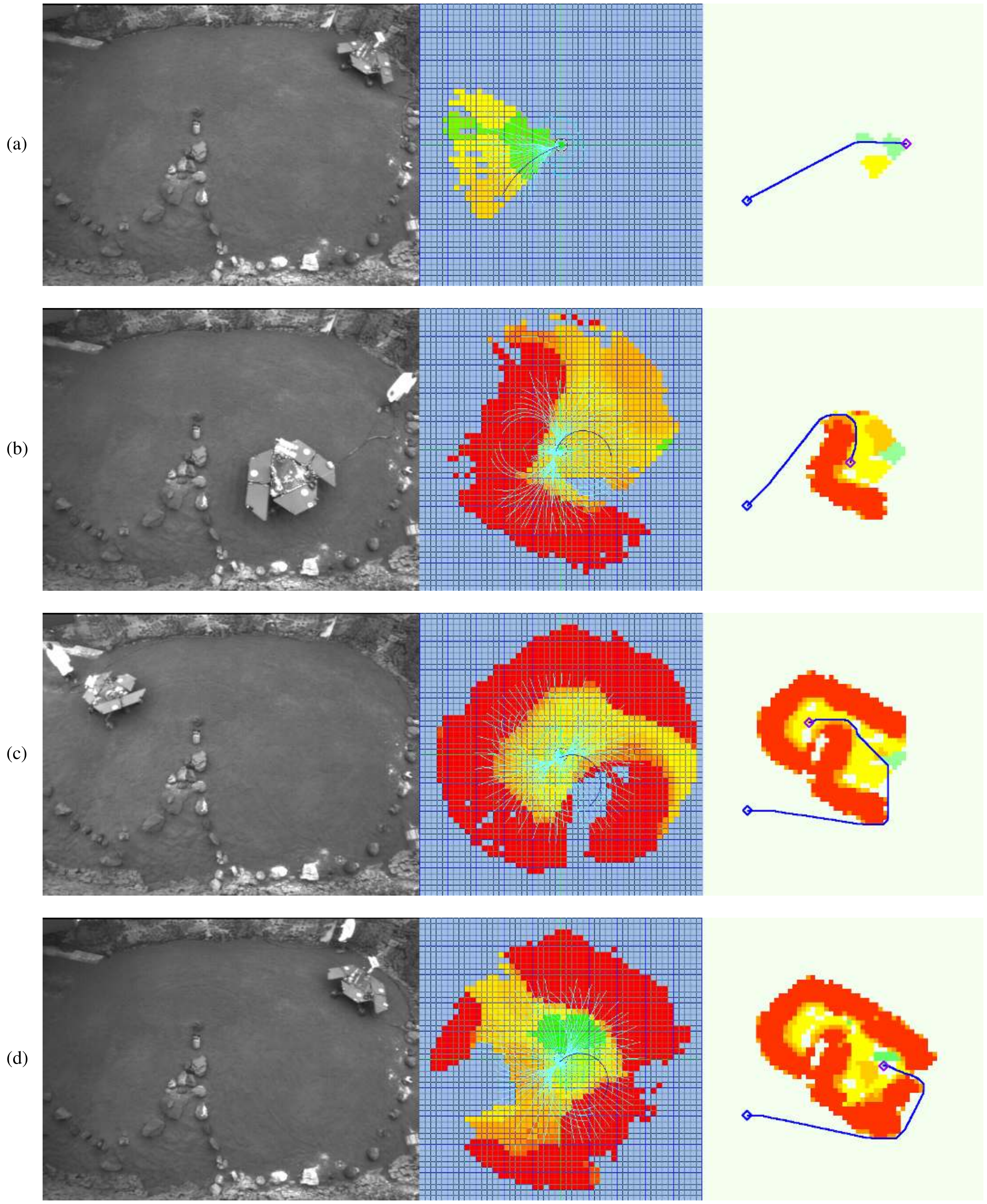


Figure 12. Field D* assisted hazard avoidance using the SSTB. The left image is an overhead view of the sandbox. The middle image is the local goodness map, and the image on the right is the Field D* cost map. Note that the entire goodness map is shown, but only a portion of the cost map is included. Blue cells have unknown traversability. All other cells are colored based on a gradient between green (high goodness/low cost) and red (low goodness/high cost). The blue line on the cost map is the path planned between the rover and the goal. The size of each goodness cell is 20 cm x 20 cm. Each cost cell is 40 cm x 40 cm.

nology, under contract with the National Aeronautics and Space Administration. The authors would like to acknowledge Carnegie Mellon University for providing a research license to use the Field D* software. The authors would also like to thank the Mars Technology Program for supporting this research. Finally, many thanks to Mark Maimone for his sage advice during the integration and testing process. Without his assistance this process would have been infinitely more difficult.

References

[1] C. Leger, et al, "Mars Exploration Rover Surface Operations: Driving Spirit at Gusev Crater". 2005 IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, Oct. 2005, pp. 1815-1822.

[2] J. Biesiadecki, et al, "Mars Exploration Rover Surface Operations: Driving Opportunity at Meridiani Planum". 2005 IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, Oct. 2005, pp. 1823-1830.

[3] M. Greco, J. Snyder, "Operational Modification of the Mars Exploration Rovers' Flight Software". 2005 IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, Oct. 2005, pp. 8-13.

[4] P. Schenker, "Advances in Rover Technology for Space Exploration". 2006 IEEE Aerospace Conference Proceedings, Big Sky, MT, Mar. 2006.

[5] S. Goldberg, M. Maimone, L. Matthies, "Stereo Vision and Rover Navigation Software for Planetary Exploration". 2002 IEEE Aerospace Conference Proceedings, Big Sky, MT, Mar. 2002.

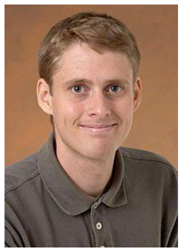
[6] J. Biesiadecki, M. Maimone, "The Mars Exploration Rover Surface Mobility Flight Software: Driving Ambition". 2006 IEEE Aerospace Conference Proceedings, Big Sky, MT, Mar. 2006.

[7] D. Ferguson, A. Stentz, "Using Interpolation to Improve Path Planning: The Field D* Algorithm". Journal of Field Robotics, Vol. 23, No. 2, Feb. 2006, pp. 79-101.

[8] Y. Cheng, M. Maimone, L. Matthies, "Visual Odometry on the Mars Exploration Rovers". 2005 IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, Oct. 2005, pp. 903-910.

[9] G. Reeves, J. Snyder, "An Overview of the Mars Exploration Rovers' Flight Software". 2005 IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, Oct. 2005, pp. 1-7.

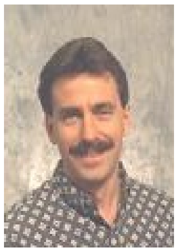
[10] T. Litwin, "General 3D Acquisition and Tracking of Dot Targets on a Mars Rover Prototype". 2005 IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, Oct. 2005, pp. 443-449.



Joseph Carsten *is a member of the technical staff at the Jet Propulsion Laboratory. He earned his M.S. degree in robotics from Carnegie Mellon University in 2005.*

Arturo Rankin *is a senior member of the technical staff at the Jet Propulsion Laboratory. He earned a Ph.D. in mechanical engineering from the University of Florida in 1997.*

Dave Ferguson *is a research scientist at Intel Research Pittsburgh. He earned a Ph.D. in robotics from Carnegie Mellon University in 2006.*



Anthony Stentz *is a research professor in the Robotics Institute at Carnegie Mellon University. He received his Ph.D. in computer science from Carnegie Mellon University in 1989.*