Scientific Programming 15 (2007) 95–105
IOS Press

Performance characterization of a rover navigation algorithm using large-scale simulation

Richard Madison*, Abhinandan Jain, Christopher Lim and Mark Maimone
Jet Propulsion Laboratory, California Institute of Technology, CA, USA

Abstract. Autonomous rover navigation is a critical technology for robotic exploration of Mars. Simulation allows more extensive testing of such technologies than would be possible with hardware test beds alone. A large number of simulations, running in parallel, can test an algorithm under many different operating conditions to quickly identify the operational envelope of the technology and identify failure modes that were not discovered in more limited testing. GESTALT is the autonomous navigation algorithm developed for NASA's Mars rovers. ROAMS is a rover simulator developed to support the Mars program. We have integrated GESTALT into ROAMS to test closed-loop, autonomous navigation in simulation. We have developed a prototype capability to run many copies of ROAMS in parallel on a supercomputer, varying input parameters to rapidly explore GESTALT's performance across a parameter space. Using these tools, we have demonstrated that large scale simulation can identify performance limits and unexpected behaviors in an algorithm. Such parallel simulation was able to test approximately 500 parameter combinations in the time required for a single test on a hardware test bed.

Keywords: Simulation, autonomous navigation, mars rover

1. Introduction

Autonomous rover navigation algorithms choose commands to direct a robotic vehicle toward a goal while avoiding obstacles. Efficient control of Mars rovers requires such algorithms, because the time delay in communicating with Mars excludes earth-bound operators from all but the slowest control loops. NASA's Mars Rovers, MER (current) and MSL (planned), use the GESTALT algorithm [4,9] to provide autonomous navigation. Using autonomous algorithms in uncertain environments is inherently risky, but the risk can be mitigated by testing the algorithms extensively on the entire range of conditions that the algorithm may face. This establishes the algorithm's operational envelope, allowing a rover operator to maximize efficiency on Mars by using autonomy without jeopardizing the rover or being overly cautious.

Such large-scale testing is best done in simulation. Hardware testing is severely limited by the availability and speed of hardware testbeds and the expense of constructing terrains. By contrast, multiple copies of a simulator can be available early in a program, and the parameters of virtual terrains are easily modified. Many copies of a simulation, running simultaneously on many terrains, can rapidly and systematically explore a navigation algorithm's performance over a large parameter space. In addition, simulation can test conditions not available to hardware test beds because of expense, risk to the hardware test bed, or simply physical impossibility.

This paper describes a prototype capability we developed to rapidly explore the operational envelope of rover technologies within a large parameter space, using as a specific example the GESTALT autonomous navigation algorithm operating on terrains with various slopes, obstacle height and distribution, and wheel slip-

*Corresponding author: Richard Madison, M/S 125-109, 4800 Oak Grove Drive, Pasadena, CA 91109-8099, USA. Tel.: +1 818 354 1897; Fax: +1 818 393 0879; E-mail: armadison@earthlink.net.

ISSN 1058-9244/07/$17.00

page. We integrated GESTALT with the ROAMS rover simulator to test the algorithm's performance over a broad range of terrain environments. We developed a progressive sequence of tests to verify the integration of GESTALT with ROAMS and the performance of component algorithms within GESTALT. We configured the ROAMS Monte Carlo (RMC) batch manager to perform these tests, running ROAMS with GESTALT many times, over a range of terrain parameters. We developed a tool to evaluate the performance on each parameter set. We also adapted RMC to operate on a supercomputer, running many tests in parallel to radically decrease the time required for a thorough walk through the parameter space. Finally, we conducted three tests to evaluate the utility of this ensemble of programs in identifying the performance envelopes of rover technologies, through the specific example of GESTALT on various terrains.

2. Testing navigation in simulation

To provide an initial testing capability, we adapted the ROAMS rover simulator to incorporate the GESTALT navigation algorithm. This allows closed loop tests of the algorithm, with a single simulation run replacing a single hardware test. The following sections describe the ROAMS simulator, the GESTALT algorithm, and integration of GESTALT into ROAMS

2.1. ROAMS

JPL has developed ROAMS (ROver Analysis, Modeling and Simulation) [1,3,6], a physics-based rover simulation environment to support trade studies, development, and closed-loop testing of rover technologies in simulation. ROAMS models the components of a robotic vehicle's mechanical subsystem, sensors, effectors, on-board control software, and environment. ROAMS operates in a series of simulation steps, calling sensor models to read rover state data, flight software to convert these data into rover commands, and effector models to evolve the rover state based on these commands and the interaction between the rover's physical components and its environment. ROAMS provides interfaces to close control loops at many levels, from motor control up through autonomous navigation. Control algorithms can be integrated as flight software components in the simulation, or they can be run standalone and use ROAMS as a server by sending effector commands to ROAMS and querying vehicle state information from ROAMS sensors. The ROAMS SimScape [2] terrain modeling toolkit allows ROAMS to incorporate arbitrary terrains, specifying terrain height, texture, and rover slippage parameters, which is useful for navigation testing. ROAMS generates geometrically realistic images of this terrain [10,11], conforming to the CAHVORE model for cameras with significant lens distortion, and made more realistic with user-specified terrain textures, sun placement and intensity, and optional terrain and rover shadows. This permits navigation algorithms, such as GESTALT, that take images as inputs.

2.2. GESTALT

NASA's Mars Exploration Rovers use JPL's Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) software to drive autonomously through unknown terrain [4]. The software accepts a cloud of 3D points as input, typically (but not necessarily) generated by stereo vision processing [12]. Points are grouped into rover turn-in-place-sized discs (e.g., 1.3 meter radius to the farthest solar panel tip) centered over a grid of 20 cm^2 cells, and a plane is fit to each set of points. Parameters from each plane fit are used to assess the overall traversability of that patch of terrain, using three distinct filters. The Step filter looks for obstacle-sized deviations from the best-fit plane, the Tilt filter compares the surface normal tilt against a pre-set limit, and the Roughness filter compares the overall residual of the plane fit against a pre-set limit. The most conservative of these filter values is assigned as the resulting "Goodness" for that grid cell.

Having constructed a complete local map, GESTALT then considers many possible motions from the current rover position. Potential paths include forward and backward arcs of varying curvature and turn-in-place motions followed by straight-ahead drives. Paths as long as 3 meters are normally evaluated, although generally only 0.5 meters are traversed before re-imaging. The "Goodness" values of the cells that comprise each path are combined using a variety of filters (e.g., cells nearby are weighted more highly than those farther away) to generate path evaluations. These are merged with similar evaluations from other higher level behaviors (like preferring to drive toward the goal), and finally the safest arc that will move the rover toward its goal is selected.

Although GESTALT has been demonstrated on several rovers [5,7], tests run on these vehicles are necessarily limited in scope. Terrain is limited to those on

which the vehicle is known to be safe, and it becomes impractical to rerun tests through identical conditions while varying just a few algorithmic parameters. Simulation provides the capability of continuously validating the software during active development, saving time by enabling initial parameter tuning across a variety of terrains, and exploring terrain types that are unsafe or impractical for actual vehicle tests.

2.3. ROAMS/Gestalt closed loop integration

ROAMS consists of a collection of *models*, representing components of the rover's hardware, software, and environment. ROAMS advances a simulation clock, calling each model at each simulation step. The models use internal state information along with input signals to calculate outputs, which in turn become input signals to other models. One way to integrate a flight software algorithm, such as GESTALT, into ROAMS is to wrap it within a ROAMS model.

The GESTALT software consists of two large pieces: a function library that encodes most of the algorithm; and a MER flight software module that executes the highest level functions and interfaces with the spacecraft's other software modules. We integrated the library into the ROAMS code base, made a ROAMS model that calls GESTALT's high level functions and converts their inputs and outputs between GESTALT and ROAMS formats, and connected the model into ROAMS.

The ROAMS model of GESTALT inputs a goal position, the estimated rover pose and images from the rover's cameras. It calls library functions to calculate the best arc to move the rover closer to the goal while avoiding obstacles. It fills one of four outputs – parameters of the arc or three flags representing end conditions – and zeroes the other three outputs. If the rover's goal and current position are close enough together, the model raises the *at-goal* flag. If GESTALT calculates arcs 500 times and still the rover has not reached the goal, then it probably will never reach the goal, so the model raises the *time-out* flag. If GESTALT finds no suitable arcs to move the rover safely toward the goal, it raises the *give-up* flag. If none of these conditions are met, GESTALT outputs the length and turn radius of the best arc. The ROAMS model can read input images and convert them to range maps for GESTALT, or it can read range maps directly. When reading range maps directly, it can read a single range map from the rover's front-mounted cameras, or it can merge range maps from both front and rear cameras.

We connected the GESTALT model into ROAMS so that GESTALT runs in closed loop. We connected the inputs of the GESTALT model to other models that estimate rover pose and generate images or range maps from virtual, rover-mounted cameras. We connected the commanded-arc output of GESTALT to the locomotor model, which in turn connects through as series of models that combine the commands, terrain, and rover dynamics to update the rover pose. This pose then becomes the input to the estimator and imaging models that provide the inputs to the GESTALT model in the next simulation step.

To run ROAMS/GESTALT, we provide the goal-coordinates input to the GESTALT model and identify other parameters used by the non-GESTALT models, such as a terrain model and initial rover pose. We let ROAMS run the simulation clock, and we monitor the GESTALT model's three output flags. The ensemble of models in ROAMS causes the rover to iteratively sense, navigate, and drive until it raises one of the three output flags. When a flag is raised, the GESTALT model produces no more arcs, so the rover simply idles. We conclude that GESTALT successfully navigated the input goal and terrain parameters if and only if the *at-goal* flag is raised.

3. Performance test plan

To evaluate our ability to test rover technologies using ROAMS, we planned a sequence of increasingly difficult exercises. In each exercise, we would run the system and verify that any failures were due to input parameters exceeding the operational envelope of GESTALT rather than any problems within the simulation system. The tests begin with several components of GESTALT disabled to simplify localizing any problems that we do find. The early tests are designed primarily to verify the system. The later tests demonstrate testing GESTALT in increasingly realistic environments and with pieces of GESTALT reinstated, leading to a test of in-tact GESTALT and a complete simulation system.

- **Flat plane.** The rover is positioned on a flat, level terrain with the navigation goal at parameter distance and heading. This provides a sanity check to verify that GESTALT correctly performs the simplest possible navigation. Any problems indicate failures in the integration.
- **Grid of rocks.** The rover is positioned on level terrain amid a regular grid of rocks. Parameters

include rock size and spacing, along with distance and direction to the goal. This test verifies that GESTALT recognizes and drives around obstacles. It also tests how high a rock must be for GESTALT to consider it an obstacle.

– **Slippery slope.** The rover drives on a slippery, inclined plane that contains a single rock of sufficient size (determined in the previous test) that GESTALT considers it an obstacle. The rover initially faces the goal. The rock is halfway between rover and goal, offset to the side by parameter distance to modulate the extent of the detour that the rover must take to avoid the obstacle. Other parameters include the slope and friction angle (slipperiness) of the plane, the downhill direction relative to the traverse direction, and the distance to the goal. One goal of this test is to verify that terrain slippage is properly invoked in the test. A second goal is to evaluate how well GESTALT navigates when slippage causes the rover's actual movements to differ from the arcs commanded by GESTALT.
– **Wheel odometry.** In the preceding tests, the pose-estimate provided to GESTALT was the ground truth rover pose. This test repeats the slippery slope or other previous test but incorporates a ROAMS model of a noisy estimator to simulate a real sensor. Noisy pose estimates should degrade GESTALT performance. The test would demonstrate proper integration of the new model in preparation for realistic testing of the navigation algorithm.
– **Grid of holes.** This test repeats the Grid of Rocks test but use negative obstacles, demonstrating GESTALT's ability to avoid holes, such as small craters.
– **Rock distribution model.** This test repeats the Grid of Rocks test, but rather than using an even grid of rocks, it uses the Golembek model [8] to produce a realistic distribution of rock sizes and positions. This allows GESTALT testing in more realistic terrains where results should map well to performance on Mars.
– **Wheel traps.** Certain rocks are small enough for the rover to traverse but large enough to wedge between wheels, rendering the rover immobile. GESTALT recognizes that these are not "step" obstacles, but their presence creates a "rough terrain" obstacle. The Wheel Traps test places these rocks along the path of the rover and verifies that GESTALT routes around them.
– **Stereo Imaging.** GESTALT identifies hazards in 3D point clouds. It can either input these clouds directly or generate them from input stereo image pairs. ROAMS cameras create both types of data. In the preceding tests, GESTALT used point clouds directly, eliminating one potential source of integration error. In the Stereo Imaging test, the system is modified to feed stereo image pairs to GESTALT. Repeating one or more of the previous tests verifies that GESTALT correctly reads and processes these stereo pairs.
– **Visible Rover.** In the previous tests, the rover was made invisible so that GESTALT would not misclassify as obstacles those pieces of the rover that are visible in the rover camera images. GESTALT prevents the misclassification by masking out rover components from the input images. The early tests input point clouds directly, incidentally bypassing the masking step. After the Stereo Imaging test reinstates the masking, the Visible Rover test makes the rover visible again and repeats the Flat Plane test, verifying that GESTALT directs the rover forward, not detecting rover components as obstacles.

4. Testing on a range of input parameters

The key to success in any of the tests outlined above is to run many times, varying the values of terrain parameters. We automated this task using the ROAMS Monte Carlo (RMC) package. We adapted RMC to repeatedly run ROAMS/GESTALT, varying input parameters across a parameter space, and recording end conditions and the path traversed by the rover in each run. We further adapted RMC to run on the JPL supercomputer, allowing about 400 test runs per hour. We developed a tool to display the paths and end conditions for many runs simultaneously, revealing patterns in which input parameters lead to which output conditions.

4.1. ROAMS Monte Carlo (RMC)

The ROAMS Monte Carlo batch job manager (RMC) runs a user-specified script of ROAMS commands many times, varying parameters over a large parameter space. Written entirely in the Tcl scripting language, RMC allows the user to specify the input parameters, output parameters, number of jobs and the commands to execute. The RMC executes the specified ROAMS commands for the specified number of times, with each

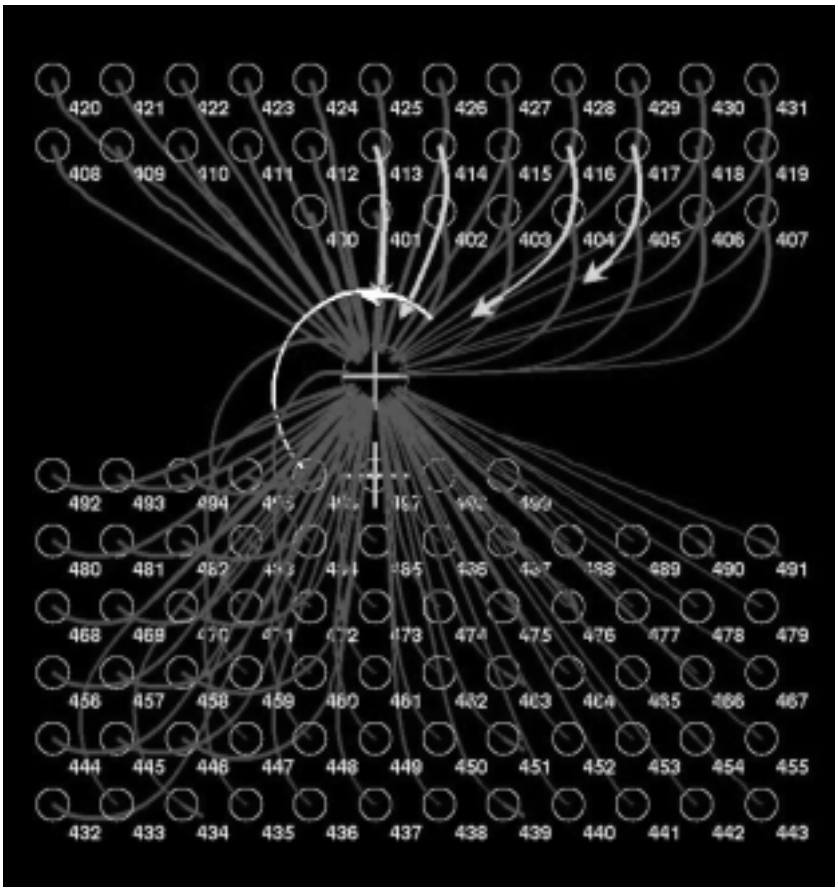


Fig. 1. The visualizer tool. A set of 100 tests beginning from a grid of starting points and several possible starting headings. The top and bottom clusters represent two different starting headings. The bottom left and bottom right seem to have starting headings 180 degrees apart, but actually the rover drove backwards there. The figure shows that most paths succeed (dark grey arrows), four times out due to overloaded processors (light grey arrows), and one time out because it cannot turn sharply enough reach the goal (white arrow).

run using a different combination of input parameters. Using Tcl to specify the code to generate input and output parameters allows the user to tailor the Tcl scripts without having to recompile the entire RMC software. RMC can draw input parameters randomly from specified ranges (Monte Carlo search), or it can test all combinations of specified sets of input parameter values (parametric search.) RMC allows the user to specify the following parameters:

– **Input variable list**. This is a list of primary "independent" variables that are varied to generate the individual batch-job parameters.
– **Variable generation from user-specified data sets**. This is a specification for the variation of an input variable. The following data sets are currently supported:
  ∗ *Uniform distribution*. The value is drawn from a continuous uniform random distribution.
  ∗ *Gaussian distribution*. The value is drawn from a continuous Gaussian random distribution.
  ∗ *Random list*. The value is picked randomly from an unordered list of predefined values.
  ∗ *Sequence list*. The value is picked in sequential order from a list of predefined values.
  ∗ *Constant*. The value is a constant.
– **Variable decorrelation**. A number of dependent variables in the simulation may need to be affected by the setting of the input (the independent) variable. In addition a number of side effects may be required upon setting of the variable. Decorrelation code allows the user to specify the one-to-many mapping as well as side effects desired for each setting of the individual variable.
– **Variable phasing**. A simulation is considered to have well defined phases at which the decorrelated variables need to be set. For example, cer-

tain parameters need to be set before any model is instanced, whereas others can only be set after a model has been instanced.

- **Result specification**. This specifies the quantities that must be output from the simulation. The recording of these quantities is specified by the corresponding code for extracting the variable of interest. After a job completes, output parameters (again defined through Tcl variables) are written to a separate results file for each job. Finally, after all the jobs are completed, the RMC gathers all the results into a single MySql database for post-processing through SQL commands.
- **Maximum number of jobs**. This specifies the number of sets of variable instances that will be generated and the maximum number of times the ROAMS script is executed. This number should be selected to both provide sufficient coverage of the individual probability distributions to ensure statistically meaningful results as well as to combinatorially exhaust the sequenced and list variables.
- **Commands to execute**. This specifies the ROAMS commands to execute. The commands are executed through Platform Computing Corporation's Load Sharing Facility (LSF) batch queuing software. RMC auto-generates an LSF script to submit the desired number of jobs to the supercomputer. The LSF software then schedules the jobs to assure fair and efficient use of the nodes on the supercomputer. Each job is given a time-out parameter (1 hour in our case) to ensure that each job will terminate properly if an error occurs. Each Monte Carlo job is executed as a completely independent process; jobs do not communicate with each other and a separate environment is maintained for each job.

4.2. Running many tests in parallel

RMC provides the capability to run many copies of ROAMS. To run these many tests in a reasonable amount of time, we used the JPL Supercomputer cluster. This is a cluster of 1024 Dell Xeon Pentium 4 processors running at 3.2 GHz. A single user can allocate jobs to as many as 200 processors, so RMC can run up to 200 copies of ROAMS at once. In early testing, the median traverse of 7 m required 7 minutes to accomplish on a single processor. An experiment with 1000 runs of ROAMS under varying parameters required $2^1/_2$ hours to complete, suggesting an average time of 30 minutes per run. The extra time stems from some processors becoming bogged down or hanging, perhaps running other programs for other users at the same time, with some runs timing out after an hour. Even with the extra time, a thousand simulation tests require about two and a half hours, comparable to two tests on a hardware test bed.

4.3. Data visualizer tool

The combination of RMC and ROAMS/GESTALT produces one output file for each set of input parameters, recording the path traversed by the rover and the end condition of the traverse that resulted from those parameters. Analyzing these outputs reveals the region of the parameter space within which GESTALT operates successfully. To quickly detect these patterns, we created a graphical visualization tool.

An example output of the tool is shown in Fig. 1. The user specifies the run numbers of a set of RMC runs to compare. For each test, the visualizer shows the rover's starting point (yellow circle), the run number (yellow text to the bottom right of the circle), and a path leading to the navigation goal (large green cross). A yellow cross marks the coordinate origin, for reference.

Each path is color coded to show which of five end conditions terminated the test. An arrow shows the position and orientation of the rover at the end of the traverse. Paths are green when the rover reaches the goal. This is the most common result. Paths are orange when GESTALT is unable to find a suitable arc to continue the path to the goal. This does not appear in the simple case of Fig. 1. Paths are red when the rover does not reach the goal after driving 500 arcs. This appears only once in Fig. 1, where the rover, executing its sharpest possible turn, went into orbit around the goal, unable to get any closer. The arrowhead along the top of the arc shows that GESTALT had reversed direction and backtracked before giving up. Paths are cyan when the host processor kills the RMC process after running for an hour. This seems to occur when the processor runs much slower than normal, either because multiple users share a processor or because bus contention causes the processor to idle excessively. Paths are dark blue when the RMC process dies without explanation, neither being killed after an hour nor having ROAMS crash. The user can specify that the visualizer should show all paths or only paths of specific colors, allowing more rapid investigation of failure cases.

5. Verification

We evaluated our ability to test rover technologies using ROAMS, RMC, the Supercomputer, and the data visualizer by running the tests outline in Section 3. For each test, we configured RMC to test every combination of values for an appropriate set of input parameters. We also provided a ROAMS script that set the rover initial pose and goal location, enabled output logging, and then stepped the simulation clock until GESTALT set a flag to indicate an end condition. Due to time constraints, we ran only the first three tests: flat plane, rock grid, slippery slope.

5.1. Flat plane

In the first test, the rover drove to goals at various distances and various azimuths relative to the starting orientation. There entire test occurred on a plane without obstacles, so we expected GESTALT to succeed in all cases. The purpose was to identify any bugs in the test code or any errors in integration of GESTALT into ROAMS.

Figure 2 shows the ROAMS visualization of the rover on the plane and a typical visualizer tool screen shot. The test consisted of 1000 runs, with the rover beginning at each starting position on a 10×10 grid and at each of 10 initial headings separated by 0.6 radians. The figure shows the 100 runs at initial heading -2.4 radians from left facing. The visualizer was asked to show only successful runs, which include all but one run in this case. In fact, most runs were successful in the flat plane test. In a small number of tests, the rover could not turn sharply enough to reach the goal (an obvious problem, in retrospect), and a few others were halted by the host computer, perhaps after being slowed by file system contention.

While the final results described above were quite satisfactory, early testing identified two limitations of GESTALT that were not obvious – the needs for rear cameras and a timeout capability. Figure 3 shows the data visualizer results for early tests, showing the 100 starting points at each of two initial headings. Many tests timed out or died mysteriously without the rover traveling very far. Our initial response was to add the timeout end condition, which had not been part of the original design. Though not shown in the figure, this addition caused most of the failures to switch from killed-by-host end condition to time-out, suggesting that the real problem was that GESTALT was causing the rover to drive back and forth along the same arc.

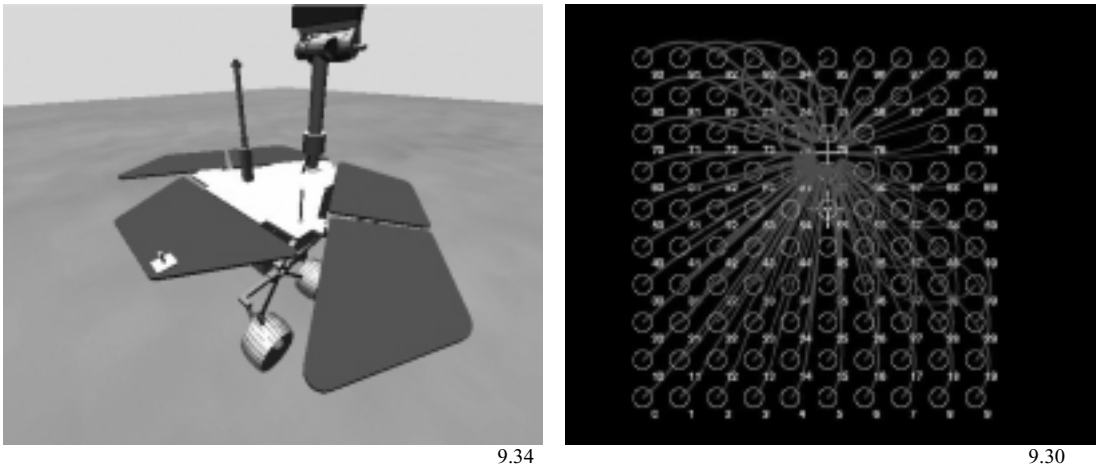
A closer look at the visualizer tool revealed that the failures all occurred when the rover was initially faced away from the goal. A little research showed that the initial integration of GESTALT into ROAMS used only the front hazcams. If the rover initially faces away from the target, the cameras show safe terrain ahead and no data for the terrain behind. GESTALT commands the rover to move into the safe terrain ahead. After moving forward for a while, GESTALT notices previously-observed, safe terrain behind the rover, toward the goal, and commands the rover to back up. This repeats until timeout. By modifying our interface to GESTALT to use both front and rear cameras, we eliminated the problem behavior, causing GESTALT to succeed in almost all cases. This provided the first example of running GESTALT on a large parameter space, visualizing the results, and identifying an algorithmic problem that, while obvious in retrospect, easily could have been overlooked if GESTALT were tested on only a few cases.

5.2. Rock grid

In the second test, we drove the rover through a regular grid of rocks on a level plane. The objective was to verify that GESTALT circumvents simple obstacles and rolls over objects too small to be obstacles. Figure 4 shows the ROAMS visualization view of the rover on the terrain and a visualizer tool view of the results from a subset of the tests.

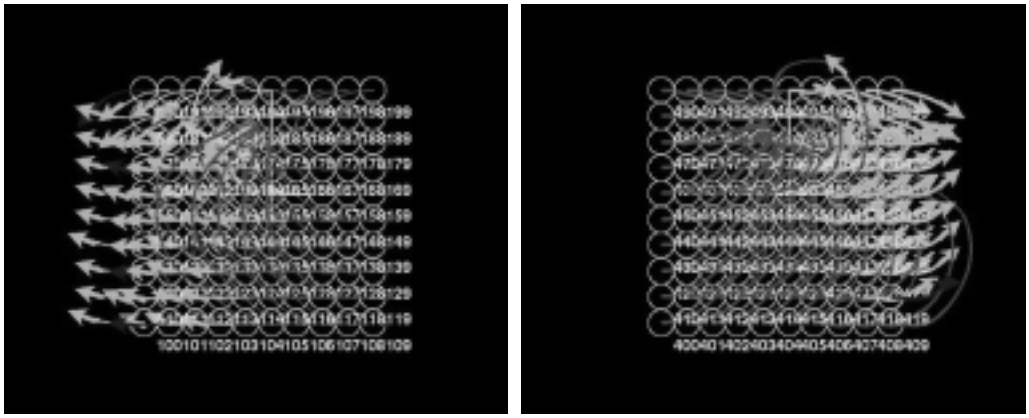
We ran 148 tests, varying the rock size, rock spacing, and goal positions. We used three rock sizes – 9 cm, 18 cm, and 27 cm – expecting that the rover would treat only the 27 cm rocks as obstacles. We tested 8 values of spacing between rocks, from 0.5 to 4.0 m apart at 0.5 m intervals. The rover was initially oriented parallel to the grid and situated midway between four rocks near the coordinate origin. The goal was set midway between two columns of rocks approximately 5 m ahead of the rover and offset to the left by one of 6 values, from 0 m to 5 m at 1 m spacing.

The visualizer tool screen shot in Fig. 4 shows the results for several of tests using 18 cm rocks. The rock grid terrain always has a rock at the coordinate origin, so starting positions are offset diagonally from the origin, centered between four rocks. The 8 possible rock spacings produce 8 possible starting positions. Rock spacings less than 2.0 m are impassible for the rover, as indicated by orange, “give up” paths. Slightly larger spacings allow the rover to pass but require paths composed primarily of horizontal and vertical segments.



9.34 9.30

Fig. 2. Rover on a flat plane. Visualizer showing all successful paths for the set of runs that begin facing about 60 degrees from north. In the end, almost all runs reached the goal.



9.37 9.38

Fig. 3. Two sets of rover tracks, each consisting of the the various starting positions with the rover pointing in a particular direction. The goal is near the top center. Starting positions with the rover pointing generally toward the goal cause GESTALT to succeed, while those with the rover facing generally away from the goal fail. The solution was to enable the rear cameras.

The largest spacings allow the rover to drive diagonally between the rocks to the goal.

In general, GESTALT behaved as expected. The rover reached the goal except when obstacle-sized rocks were close enough together that the entire terrain appeared to be an obstacle. A few runs were killed unexpectedly by the supercomputer host. We were surprised to see that 18 cm rocks were treated as obstacles, though GESTALT was set up to identify obstacles at 20 cm height. In a real testing scenario, this might cause a user to go back and review the GESTALT parameters to understand this unexpected behavior and perhaps to warn users if the behavior is correct but counterintuitive.

5.3. Slippery slope

In the third test, the rover drove on a slippery slope, circumventing a rock on the way to its goal. The rock is 18 cm tall, which GESTALT considers an obstacle according to the previous test. The goal for this third test was to evaluate the ability of GESTALT to avoid obstacles despite slippage. Figure 5 shows the rover on its terrain and a visualizer tool screen shot of some results.

The test varied five parameters, providing 5040 combinations. The rover was positioned initially to face a navigation goal, 2 m, 4 m, or 6 m away. The obstacle was placed halfway between the start and goal positions, then offset to the left by 0.1 m, 0.3 m, 0.6 m,

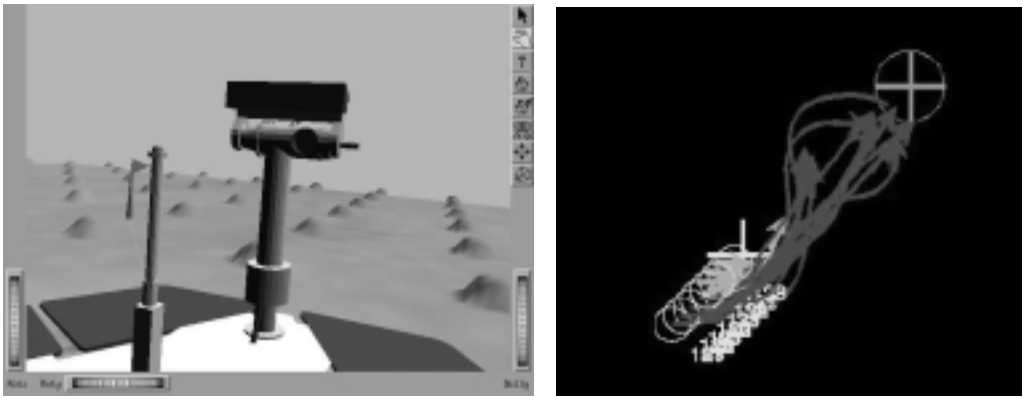


Fig. 4. The second test, wherein the rover traversed a grid of rocks. Close-together rocks are impassible, producing light grey trails. Farther spacings allow the rover to navigate, producing dark grey trails that reach the goal. Not all trails end at the marked goal, because the Visualizer Tool shows only one goal, but the goal has a different location for each rock spacing.

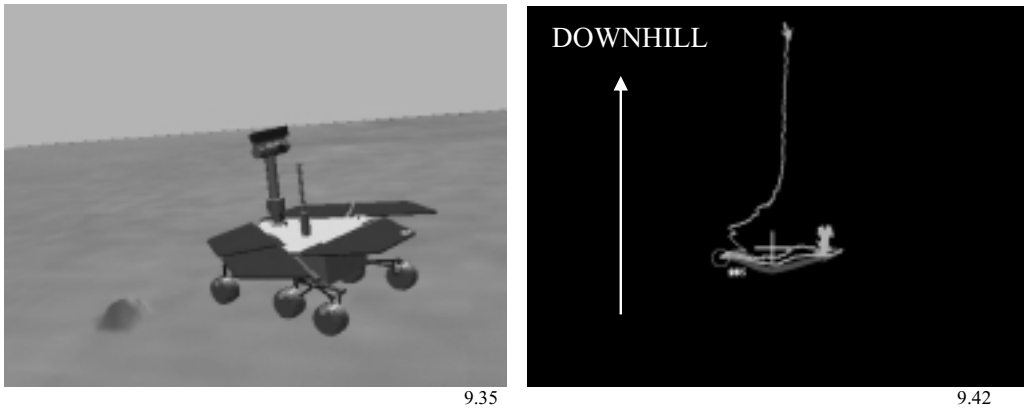


Fig. 5. The rover tries to drive past a rock on a slope. It succeeds when slippage is low and slides down hill when slippage is higher.

1.0 m, or 1.5 m. At the largest offset, the rover should drive directly to the goal. With decreasing offset and decreasing distance to the target, GESTALT must command increasingly drastic arcs to avoid the obstacle. In addition, the terrain was sloped and slippery, requiring GESTALT to overcome slippage-induced failure to execute the commanded arcs. The downhill direction relative to the goal direction was varied in 30° increments from 0° to 330°. The terrain slope was varied from 0° to 30° in increments of 5°. The friction angle took on values of 17°, 23°, 29°, and 34°. On a pure plane, the rover would have traction for terrain slopes less than the friction angle and slide uncontrollably for terrains with higher slopes. ROAMS adds some randomness to create a more gradual, more realistic transition between these two behaviors. We expect to see rover slippage increase gradually as the terrain slope increases and/or the friction angle decreases.

The visualizer view in Fig. 5 shows a typical result, in this case several runs where the only parameter that varies is terrain slope. The top of the image is downhill. The yellow cross marks the obstacle, which begins slightly downhill from the rover-to-goal path. The friction angle is 23°. At low slopes, up to 20°, the rover drives around the rock, showing successful, green tracks. The rover detours uphill from the rock, which is a shorter diversion because the rover is initially uphill from the rock. Some slippage is evident from the thickness of the green line, which is actually four not-quite-overlapping lines. At slopes of 20° and 25°, near the friction angles, the rover slips noticeably. As it approaches the target, it slips further down hill and therefore points further uphill to continue to point to the target. The rover has the least traction when pointing up hill, and once it is directly below the target, it cannot muster the traction to climb up to the target. It

eventually times out, producing a cyan track. At a 30° slope, friction is so lacking that the rover slides off the virtual world, producing an orange track as GESTALT recognizes that it has left the terrain and thus cannot identify a viable arc to command.

Beyond indicating that the simulation functioned properly, this test revealed a behavior of GESTALT that might not have otherwise been noticed. At high slopes, near the friction angle, the rover can drive uphill at shallow angles but not at sharp angles. To reach a cross-slope goal, the rover must navigate toward a point slightly uphill from the actual goal, gaining altitude while at its initial cross-slope heading. Later, as it turns to face the new goal and begins to slide, it can slide down to the original goal.

6. Conclusions

Testing GESTALT or other algorithms in simulation on a supercomputer allows a much more thorough investigation of the limits of our navigation software than is possible using hardware testbeds alone. To demonstrate the viability of such software testing, we integrated the GESTALT algorithm into the ROAMS simulator, allowing closed-loop testing in simulation. We adapted the Roams Monte Carlo batch manager to run the simulation many times using varying input parameters, exploring the parameter space. We ran this on the JPL Supercomputer cluster, testing GESTALT's performance at about 400 points in the parameter space per hour. We developed a visualization tool to help rapidly identify trends in the results. We ran three experiments to verify that the ensemble of programs functioned together. These tests revealed some surprising behaviors of GESTALT that might have been overlooked if limited hardware-test bed testing had prevented exploration of the regions of the parameter space where the behaviors occurred. This capability should be beneficial to MSL, which is slated to use GESTALT and unlikely to have schedule for extensive hardware test bed testing of GESTALT.

The impact of this initial capability to do large-scale simulation extends beyond the example application to GESTALT. The task developed the capability to evaluate the performance of a rover technology over a range of environmental (or other) parameters by embedding the technology in the ROAMS simulator and running many simulations in parallel on a supercomputer, each evaluating one point in the parameter space. It demonstrated exploring a space of terrain parameters such as wheel slippage and slope that may not even be available to hardware testbeds. It demonstrated the ability to run hundreds of tests in the time required to run a single test on a hardware test bed.

The framework is now available for testing GESTALT against additional parameters such as pose estimation noise. It is also useful for testing the impact of environmental or other parameters on other rover technologies. It also can be used to test rover algorithms in general and, more broadly, to test new rover hardware designs.

Two future extensions to enhance the current capabilities are presented here. The first is to conduct the remaining experiments in the test plan. The second is to identify when the rover drives into a region that GESTALT considers unsafe. One way to do this is to add a tool that applies GESTALT's traversability evaluator to the entire terrain map. A user would use this tool before running ROAMS to create a map of areas that GESTALT considers dangerous. Then as ROAMS generates and reports the actual path of the rover, it could monitor the safety map and report, along with the simulation end criterion, the traversability of the worst point on the path.

Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and funded through the internal Research and Technology Development program. The research was sponsored by the JPL Research and Technology Demonstration program. We would like to thank Gerard Benenyan and Leonard Reder for writing the data visualizer tool and running the experiments.

References

[1] A. Jain, J. Balaram, J. Cameron, J. Guineau, C. Lim, M. Pomerantz and G. Sohl, Recent Developments in the ROAMS Planetary Rover Simulation Environment, IEEE 2004 Aerospace Conference, Big Sky, Montana, March 6–13, 2004.

[2] A Jain, J. Cameron, C. Lim and J. Guineau, SimScape Terrain Modeling Toolkit, Second International Conference on Space Mission Challenges for Information Technology (SMC-IT 2006), Pasadena, CA, July 2006.

[3] A. Jain, J. Guineau, C. Lim, W. Lincoln, M. Pomerantz, G. Sohl and R. Steele, Roams: Planetary Surface Rover Simulation Environment, i-SAIRAS 2003, Nara, Japan, May 19–23, 2003.

[4] J.J. Biesiadecki and M.W. Maimone, The Mars Exploration Rover Surface Mobility Flight Software: Driving Ambition, IEEE 2006 Aerospace Conference, Big Sky, Montana, March 2006.
[5] J.J. Biesiadecki, M.W. Maimone and J. Morrison, The Athena SDM Rover: A Testbed for Mars Rover Mobility, i-SAIRAS 2001, Montreal, Canada, June 2001.
[6] J. Yen and A. Jain, ROAMS: Rover Analysis Modeling and Simulation Software, i-SAIRAS 1999, Noordwijk, Netherlands, June 1999.
[7] J. Montgomery, Mars Science Laboratory GESTALT Technology Test Report, JPL technical report D-34115, August 2006.
[8] M. Golembek and D. Rapp, Size-frequency distributions of rocks on Mars and Earth analog sites: Implications for future landed missions, *JGRP* **102:E2** (Feb 1997), 4117–4129.
[9] M.W. Maimone, J.J. Biesiadecki, E. Tunstel, Y. Cheng and C. Leger, Surface navigation and mobility intelligence on the Mars Exploration Rovers, Intelligence for Space Robotics, TSI Press, Albuquerque, 2006, 45–69.
[10] R. Madison, M. Pomerantz and A. Jain, Camera Response Modeling and Verification in ROAMS, i-SAIRAS 2005, Munich, Germany, September 2005.
[11] R. Madison, Verification of ROAMS CAHVORE Imaging, JPL technical report D-35616, June 2006.
[12] S.B. Goldberg, M.W. Maimone and L.H. Matthies, Stereo vision and rover navigation software for planetary exploration, IEEE 2002 Aerospace Conference, Big Sky, Montana, March 2002.