Published in 2002 IEEE Aerospace Conference Proceedings, March 2002, Big Sky, Montana, USA

Stereo Vision and Rover Navigation Software for Planetary Exploration

Steven B. Goldberg
Indelible Systems
8921 Quartz Ave
Northridge, CA 91311
+1 (818) 998 - 6895
isteve@robotics.jpl.nasa.gov

Mark W. Maimone Larry Matthies
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
+1 (818) 354 - 0592
{mwm,lhm}@robotics.jpl.nasa.gov

http://robotics.jpl.nasa.gov/people/mwm/visnavsw/

Abstract—NASA's Mars Exploration Rover (MER) missions will land twin rovers on the surface of Mars in 2004. These rovers will have the ability to navigate safely through unknown and potentially hazardous terrain, using autonomous passive stereo vision to detect potential terrain hazards before driving into them. Unfortunately, the computational power of currently available radiation hardened processors limits the amount of distance (and therefore science) that can be safely achieved by any rover in a given time frame.

We present overviews of our current rover vision and navigation systems, to provide context for the types of computation that are required to navigate safely. We also present baseline timing results that represent a lower bound in achievable performance (useful for systems engineering studies of future missions), and describe ways to improve that performance using commercial grade (as opposed to radiation hardened) processors. In particular, we document speedups to our stereo vision system that were achieved using the vectorized operations provided by Pentium MMX technology. Timing data were derived from implementations on several platforms: a prototype Mars rover with flight-like electronics (the Athena Software Development Model (SDM) rover), a RAD6000 computing platform (as will be used in the 2003 MER missions), and research platforms with commercial Pentium III and Sparc processors.

Finally, we summarize the radiation effects analysis that suggests that commercial grade processors are likely to be adequate for Mars surface missions, and discuss the level of speedup that may accrue from using these instead of radiation hardened parts.

TABLE OF CONTENTS

0-7803-7231-X/01/$10.00/
IEEEAC paper # 324

1. INTRODUCTION

Planetary rovers now have the ability to navigate safely through unknown and potentially hazardous terrain, using autonomous passive stereo vision to detect potential terrain hazards before driving into them. A local map of the terrain can be maintained onboard, by resampling and effectively managing the range data generated by stereo vision. NASA's Mars Exploration Rover (MER) missions will drive safely on the Red Planet in early 2004 using this type of technology.

Stereo vision is an attractive technology for rover navigation because it is passive; sunlight provides all the energy needed for daylight operations. Hence only a small amount of power is required for the imaging electronics to obtain knowledge about the environment. And with enough cameras or a wide enough field of view, there need be no moving parts in the system. Having fewer motors reduces the number of components that could fail.

Our navigation system relies on a geometric analysis of the world near the rover, combining various range data snapshots generated by the stereo system into a local map. We developed a system for interpreting this data, called the Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) system, based on Carnegie Mellon's Morphin algorithm [9], [10].

Although the MER mission (launching in mid-2003) only requires the rover to travel at most 100 meters per day, future missions like the Smart Lander Rover being considered for 2009 will require rovers to travel even farther, hence at faster speeds. In this paper we describe our current algorithms for autonomous rover navigation, and provide baseline timings for implementations of these algorithms on a variety of platforms. Our implementations are primarily written in C and C++, but certain optimizations are hard coded in assembler to take advantage of vector operations. These timings provide a benchmark from which future rover driving capabilities can

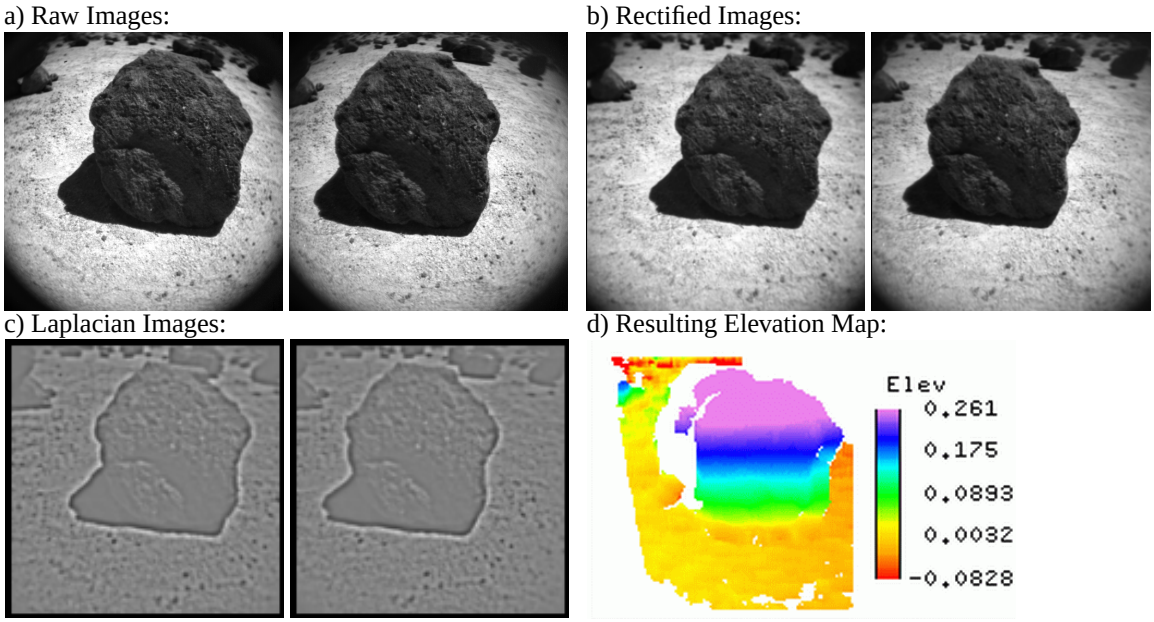

Figure 1. Illustration of the steps involved in Stereo Vision Processing.

be derived.

2. Stereo Vision Algorithm

JPL has applied Stereo Vision software to rover motion control for many years. Although certain aspects of our approach have appeared before [3], [14], we will summarize the overall algorithm here before presenting new results that take advantage of some commercial vectorized processors that have only recently become available.

Our algorithm depends on certain physical properties of the stereo camera system. The pair of stereo cameras must be rigidly mounted to a camera bar. Using a pair of images of a known calibration target, a pair of geometric camera lens models is calculated using Gennery's CAHVORE formulation.[2] This formulation assumes the system will maintain its geometric calibration over some useful time period (e.g., days or weeks for research purposes, weeks or months for deployed vehicles). This is a reasonable assumption: examples of NASA-sponsored robot camera systems that have maintained their stereo calibration in spite of both high vibration deployments and/or long periods of use include Dante [8], Nomad [13], Rocky 7 [12], [5], and the Mars Pathfinder Lander [11].

Our stereo vision algorithm can be described as follows:

1. To decrease the computational burden and the effect of the rigidity constraint, often the raw sensor images are reduced in size, e.g., from 1024×1024 source pixels down to 256×256 pixels, by averaging pixel values (see Figure 1a). Each so-called *pyramid level* reduction results in an 8-fold decrease in computation: a factor of two from each spatial dimension, and an additional factor of two from a reduction in the number of integer disparities that need to be searched. There is a cost though: the depth resolution of the resulting range estimates doubles (i.e., becomes less precise) with each pyramid level reduction [4].
2. Each image pixel encodes the appearance of a location in the 3D world; in particular, the surface of that object nearest the camera along a certain ray. To find the pixel that represents the same object surface in the other image, it is sufficient to search only along the projection of that ray. Since the bulk of the processing time in stereo vision is spent doing this search, we simplify later processing by resampling each image so that searching these rays requires only integer operations. Pairs of images are thus *rectified*, ensuring that these rays (called *epipolar lines*) are aligned with the horizontal, as in Figure 1b.
3. We compute the Laplacian of each image to remove any pixel intensity bias, e.g., Figure 1c. Actually, our implementation computes an approximation, a Difference of Gaussians, which can be done more quickly.
4. The filtered images are then fed into a 1-D correlator that uses a 7×7 pixel window. The correlator considers a number of potential matches for each pixel in the left image of each stereo pair, assigning a score to each potential match. The range of pixels to be searched is called the *disparity range*, and is derived geometrically from the input range of depth values to be searched (e.g., from 30 cm to 3 meters in front of the cameras). The maximum-scored match is selected, and the camera model is applied to determine the corresponding range estimate. This process is repeated for every pixel in the left image. We take advantage of the inherent parallelism using a *sliding sum* implementation to compute the correlation scores efficiently.
5. Not every range estimate is accepted, however. A variety of checks is applied to prune out unreliable estimates. For example, the *peak filter:* the chosen score must be better than that at adjacent pixels. A flat correlation peak would mean

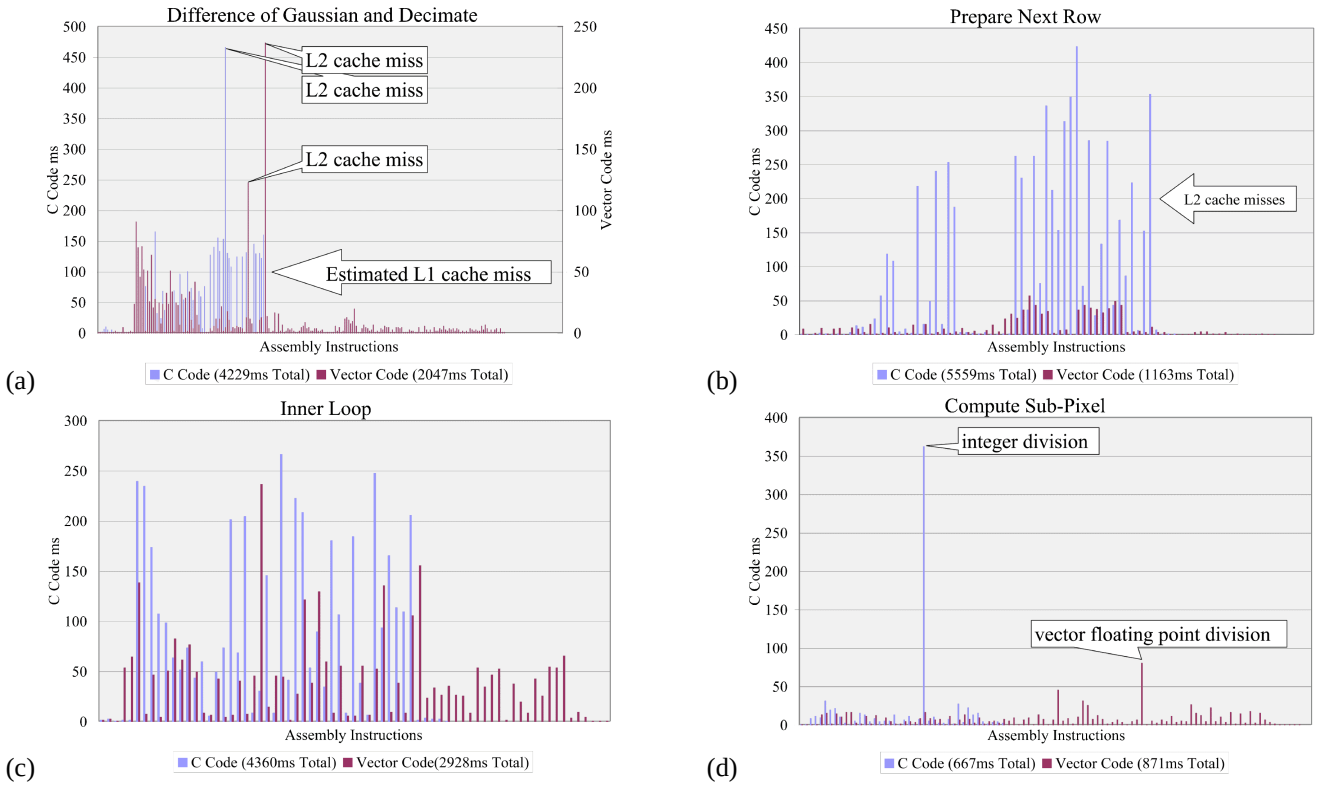

Figure 2. Stereo Vision Results: Unoptimized (light blue) vs. Vector-optimized (dark red) run times for 4 functions. The X axis represents individual assembly instructions executed for each function; the first instruction in a function is on the far left, the last instruction on the far right. The Y axis represents total time in milliseconds spent executing a particular instruction, integrated over 200 iterations.

that many nearby pixels have the same appearance, resulting in an unresolvable ambiguity. Also, the *Left/Right Line of Sight filter:* the correlator is run in the reverse direction, yielding independent range estimates for pixels in the right image as well. If an estimate from a pixel in the left image fails to match that from its correspondent in the right image, the estimate is discarded.

6. Having pruned some of these values, any remaining small isolated regions of range values are thrown out. These safety checks (the *peak filter*, *the Left/Right Line Of Sight filter*, and this *Blob filter*) result in a robust set of correspondences that can be used by the onboard autonomy system.

7. Finally, each disparity value can be mapped to a 3-D (X,Y,Z) location using the geometric camera model. This information can be displayed in many forms; an elevation map is shown in Figure 1d.

3. Stereo Vision Optimizations

The ready availability of commercial vectorized processors has allowed us to realize significant improvements in the performance of our stereo implementation. Faster interpretation of the world allows our rovers to drive safely at ever faster speeds, e.g. the Urbie robot [6] which can now drive safely at over 1 meter per second. In this section we document the speedups obtained by taking advantage of the Pentium MMX capabilities. All the graphs in Figure 2 reflect timings taken on a Pentium III 700 MHz CPU with 32 Kbyte L1 cache, 256 Kbyte L2 cache and 512 Mbyte RAM running Windows 2000. While currently used only on Earth-based rovers, such commercial (i.e., non-radiation hardened) processors might also be used on future space missions, as we discuss in Section 6.

We focus attention on four particular functions: local 2D pixel resampling in *Difference of Gaussian and Decimate*, buffer preparation in *Prepare New Row*, correlation score comparison in the *Inner Loop*, and integer-based quadratic peak finding in *Compute Sub-Pixel*. The first two require memory accesses that jump across image row boundaries, and the latter two perform many independent operations on 8-bit integer data. These properties make them useful candidates for vectorization.

Difference of Gaussian and Decimate — Used to filter and decimate the images before stereo calculations, this algorithm is implemented using sliding sums. Working with the original stereo images, 2 @ 240 Kbytes (512*480), this algorithm must access main memory and will incur L2 cache misses. To make the most of each L2 cache miss, the vectorized implementation operates over whole cache lines. To guarantee only whole cache lines are used, a small portion of the original algorithm is used to align the inputs. As shown in Figure 2a, the vectorized version does have a more localized occurrence of

L2 cache misses, and its efficient use of Pentium III prefetching makes better use of the L1 cache. Note, in Figure 2a, the L1 cache misses can not be directly measured so an estimation of L2 cache reads is used. This, coupled with a vectorized computation of sliding sum values, results in a 2× speed increase.

Prepare Next Row — This function maintains the sliding sum buffers. It accesses left and right input images and both sliding sum buffers. Optimizing this function produces the largest gains, 4.8×, from vectorization and Pentium III cache optimizations. This is accomplished by operating over full cache lines of aligned memory. Since access to image data is not aligned, this algorithm makes use of the Pentium III prefetch instruction to assist in loading the required cache lines into the L1 cache. The other memory, namely the sliding sum buffers, is guaranteed to be aligned. In Figure 2b, the tall light blue bars indicate how much time is wasted by the C algorithm waiting for the processor to fetch operands from memory.

Inner Loop — The core of the stereo matching algorithm, this function finds the level of disparity with the best correlation score for left and right disparities, and saves the necessary information to generate sub-pixel information. The C algorithm used an unaligned data structure to store four shorts and one byte of data. This was not vectorizable so it was rewritten to store two shorts in one data structure and one correlation value in a table. With this simpler data structure, the "Inner Loop" could be coded using vector operations, specifically, vector comparisons. Using vector comparisons rather than if statements prevented pipeline thrashing and allowed four left and four right disparities to be calculated every pass. The use of a table of correlation scores was slower but necessary to reduce the size of the data structure. By reducing the size of the data structure and using vector comparisons, the vector algorithm performs over 1.5 times faster. This algorithm could be further optimized by removing the right disparity calculation and using a correlation table, but this would sacrifice the left-right-line-of-sight filter and sub-pixel disparity.

Compute Sub-Pixel — Run only after the best correlation score is found for each row, this function generates the final pixel and sub-pixel disparity image. The C version is faster here because it makes use of the spatial locality of the three best correlation scores for each disparity, while the vectorized version must do a table lookup to find the two of the three scores. This overhead is partially absorbed by the use of a vectorized division rather than individual integer divisions for each sub-pixel value. In Figure 2d, the two annotations point to the timings for these divisions. Note, the vectorized division is called 4 times fewer than the integer version, but is still 1.2× faster. Unfortunately, while this algorithm has most recently been optimized and may still provide a performance increase, at the moment it results in a 1.3× slow down.

These performance numbers, measured with Intel's VTune, have been measured without the benefit of compiler optimizations. With the compiler set to optimize for speed, the final results show the JPL stereo algorithm running 2.5 times faster. That translates to just over 20 frames per second on a Pentium III 750 with 256 Kbyte L2 cache, working on 256×240 images over 32 disparities and with a window size of 8×8.

4. GESTALT NAVIGATION SYSTEM

A primary input to any navigation system is a metrically-specified *waypoint.* Although one could tell the rover to drive randomly, typically it will be sent to a particular point in the world. Waypoints may be specified *statically* by simply giving an (X,Y,Z) value in a known world frame, or *dynamically* by providing a module that can track a feature in the world and always return its current position. In what follows we assume the waypoint is static, but the extension to dynamic waypoints is trivial.

At its core, GESTALT is a set of routines that decide the next best direction for a rover to move, given the state of the world already seen, new sensor data, and a desired waypoint goal. It first checks to see if the rover has already reached its goal, or at least a point within some tolerance band around it. If so, the navigation cycle has completed and the traverse will terminate successfully.

The rover will rarely start out already at its goal, however. When it has any distance left to travel, it will evaluate its terrain information to determine the safety of all possible nearby turns. Sensed data about the terrain can come from any number or type of sensors, so long as their results are prefiltered to provide individual point measurements of (X,Y,Z) data in the rover's (not the sensor's) coordinate frame. GESTALT then chooses from among the safe turns, one that will best help it reach the goal. The desired turn and a short distance (e.g., 35 cm) is then sent to the low-level wheel controller, and the rover is commanded to move blindly.

While the rover is driving its next step, it will not use its imaging sensors to look for obstacles. Other types of safeguarding will likely be enabled (e.g., tilt sensors, motor current limits, potentiometers that monitor kinematic limit configurations), but no additional high-level terrain-based planning or sensing need be performed.

At the end of each step, sensors are expected to provide a reasonably accurate estimate of the rover's new position. GESTALT does not require that the rover motion exactly match that which was commanded, but it *does* assume that wherever the rover ended up, its relative position and orientation can be reasonably inferred and provided as input. That is one limitation of the system, that it relies on other modules to deal with myriad position estimation problems (slipping in sand, getting stuck on a rock, freeing a jammed wheel, etc).

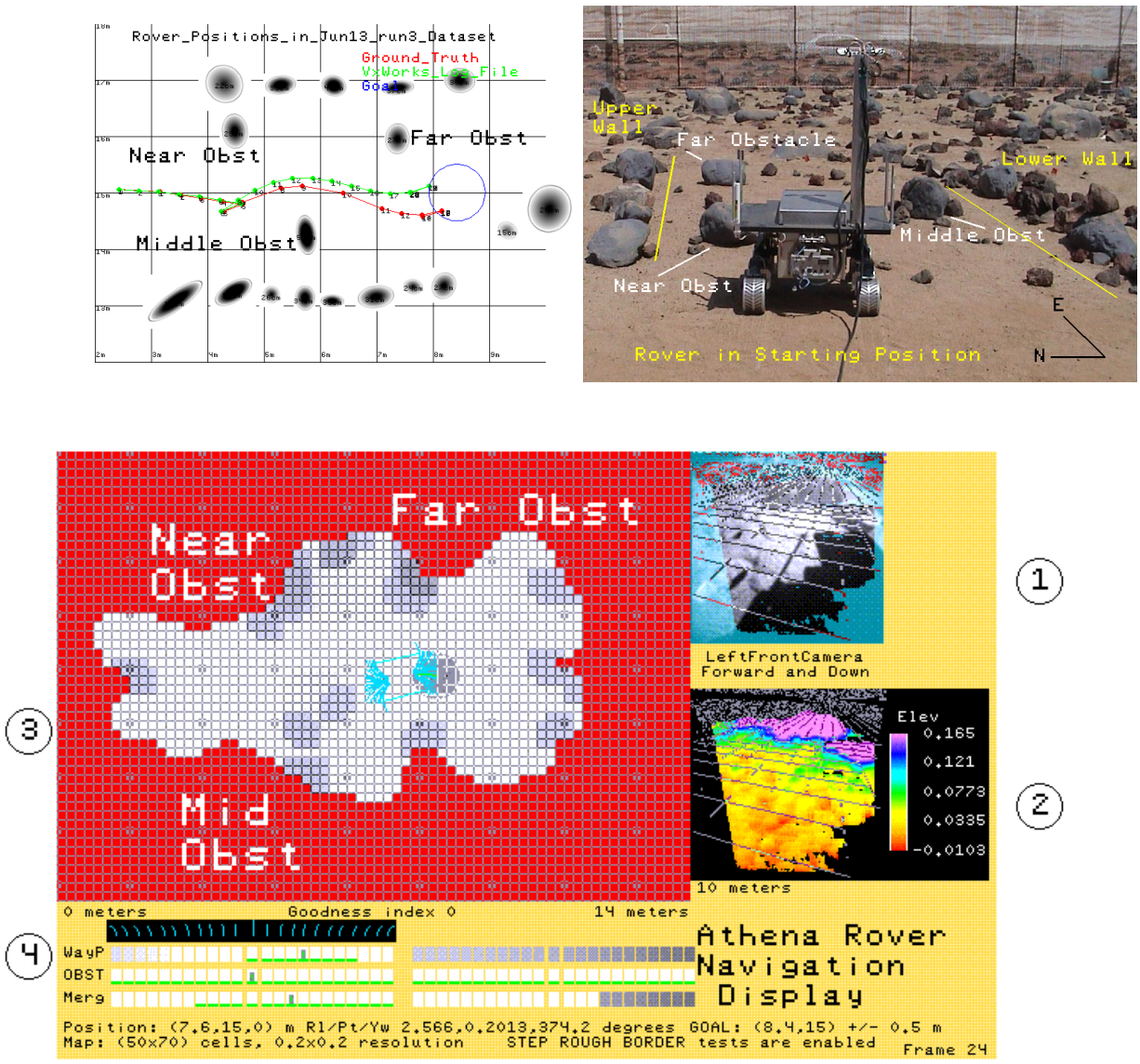

Figure 3. A successful 5 meter run through a narrow obstacle course. The upper left image shows actual obstacle locations and actual rover path, as measured using a surveyor's ranging theodolite. The upper right image is a picture of the test course and rover. The bottom image is rendered at the same scale as the upper left image, and shows the local map built by the rover during its traverse. This bottom image is one of GESTALT's diagnostic images, and includes (1) the rover view from the left forward-facing camera with grid superimposed, (2) an elevation image corresponding to (1), (3) the local occupancy grid with (dark) obstacles and possible steering arcs, and (4) a ranking of possible headings showing best heading.

Choosing a Safe Direction

Range images generated by Stereo Vision are usually not sufficient, in and of themselves, to determine a safe driving path. Field of view restrictions and error recovery behaviors might force a rover to turn into an unseen area. For this reason we keep a *local map* of the area around the rover, so that it can reason more effectively about its surroundings. This map is maintained not from the perspective of the rover cameras, but from an overhead "bird's eye" viewpoint. Figure 3 shows an example of a rover map next to a similarly-scaled (but independent) measurement of the environment.

GESTALT models the world as a grid of regularly spaced cells, with each cell typically the size of a rover's wheel. Each cell stores an 8 bit *goodness* and *certainty* value, or is tagged *unknown*. The resolution of the grid cells, the evaluation assigned to particular types of obstacles, the types of tests to be performed, are all parameters that may be changed prior to (some even during) a traverse; a nearly complete list can be found in Table 1.

The GESTALT local map currently uses a *configuration space* representation of the environment. That is, the contents of each cell in its map represents whether a rover-sized object

Table 1. Example GESTALT Parameters

Parameter	Description	Default
	Arcs	
max-curvature	*Max curvature (close to point turn)*	2.0 m^{-1}
num-forward-arcs	*Number of forward arc paths to evaluate*	23
num-backward-arcs	*Number of backward arcs to evaluate*	23
point-turn-amount	*How far to rotate during a point turn*	$\frac{\pi}{2} radians$
	Waypoints	
tight-curve-fraction	*Broad (0) v. Tight (1) waypt curves*	1.0
vote-index-variance	*Waypoint Gaussian variance*	225.0 $index^2$
	Grid	
x0, y0, xN, yN	*Grid bounds*	0 m, 0 m, 10 m, 10 m
xstep, ystep	*Grid resolution*	0.2 m × 0.2 m
xelts, yelts	*Computed from bounds and resolution*	50×50 cells
plan-min-dist	*Evaluate paths starting at this offset*	0.3 m
plan-max-dist	*Limit path evaluation to this length*	3.0 m
max-idles	*Max number of cycles before zeroing a cell*	2000 cycles
grid-dist-timeout	*Throw out map data this far from rover*	3.0 m
	Traversability Tests	
enabled-tests	*Which Traversability tests should be run*	step,rough,border
outliers-per-cell	*Number of outliers to reject during plane fitting*	0
eigen-ratio	*Min eigenvalue ratio to validate planar fit (1:∞)*	2.0
min-cell-coverage	*Min fraction of cell that must contain range data to enable surface normal (in each of X and Y)*	0.5
	Hazard (Goodness, Certainty) Pairs: each value ranges from 0 to 255	
unknown	*Value for unknown cell*	(11, 0)
waypoint-certainty	*Certainty for waypoint votes*	192
unknown-rovers	*If no rover model*	(0, 0)
border	*Value for goodness of border cell*	(250, 128)
min-allowed-goodness	*All path cells must be larger*	10
step-obst	*Step obstacle*	(0, 255)
pitch-obst	*Pitch obstacle*	(0, 255)
rough-obst	*Roughness obstacle*	(0, 255)
min-fwd-threshold	*Vote theshold for only imaging fwd*	180
min-vote-goodness	*Threshold for merging votes*	26
	Rover Parameters	
rover-length	*Rover Length*	104 cm
rover-width	*Rover width*	84 cm
obst-height	*Tallest traversable obstacle height*	20 cm
wheel-radius	*Wheel radius*	10 cm
max-pitch	*Maximum allowed pitch angle*	25 degrees

centered at that cell would encounter obstacles *anywhere underneath the rover chassis*, possibly in the surrounding cells. The intuitive effect of this representation is that the appearance of an obstacle in the local map tends to grow beyond the obstacle's physical boundaries by half the vehicle width in all directions.

The flow of events that occurs during a single navigation step is as follows:

1. GESTALT is invoked with a current rover position and attitude estimate and new range data. This range data could come from stereo vision, lidar, laser scanners, or any type of range sensor; it is assumed to be a set of discrete (X,Y,Z) locations on surfaces visible from the rover's current position. These (X,Y,Z) locations are assumed to be expressed in the coordinate frame of the onboard map.

2. First and second order moment statistics are collected about all the range points in a given cell. No matter how many range points contribute to a cell, only 10 floats comprising these statistics are stored: number of points, $\sum X, \sum Y, \sum Z, \sum X^2, \sum Y^2, \sum Z^2, \sum X \cdot Y$, $\sum X \cdot Z, \sum Y \cdot Z$.

3. The *traversability* of each cell is found by merging the moment statistics from a rover-sized patch of surrounding cells and finding the best-fit plane. As long as there is enough data,

and the data passes some preliminary tests (e.g., at least half the cells have more than 1 point), the plane parameters are used to compute several hazard measures:

Step Hazard: Find the maximum elevation difference between any pair of cells in this patch e. If less than 1/3 the rover clearance height (h), there is no step hazard; else there is a hazard with goodness $127 \cdot (1 - \min(1, e/h))$.

Roughness Hazard: Compute the residual from the planar fit r. If less than *roughness fraction*$\cdot \frac{1}{3}h$, there is no roughness hazard; else there is a hazard with goodness $255 \cdot (1 - \min(1, \textit{roughness fraction} \cdot \frac{r}{h}))$.

Pitch Hazard: Compute the slope s from the planar fit (only if enough data are present). There is a pitch hazard with goodness $255 \cdot (1 - \min(1, \frac{s}{\text{max pitch angle}}))$.

Border Hazard: If this cell borders an unknown cell, there is a hazard with goodness *border goodness*.

The minimum goodness according to these hazards is assigned to the cell with a certainty of 255.

4. **Hazard Arc votes:** Arc paths are considered out to *min (dist to goal, default length)*. A weighted sum, biased to consider nearby cells more strongly, is used to find the value of each arc path. The values assigned to each arc path, forward and backward, are displayed in the diagnostic image as the middle row next to Figure 3, part (4).

5. **Waypoint Arc votes:** Independently of the hazard avoidance system, the same set of arcs is assigned goodness values according to how well they move the rover toward its goal point. The arc that best points the rover toward the goal gets the highest value, and that value forms the peak of a gaussian curve that is applied to the other arcs. The variance of this curve is a system parameter, *vote-index-variance*. Both forward and backward votes are assigned values; arcs running in the direction toward the goal get a peak of 255, those in the opposite direction get a peak of 128. Waypoint arc path votes are displayed in the top row of Figure 3, part (4).

6. **Merging Hazard and Waypoint arc votes:** Hazard and waypoint votes are merged pairwise. If either vote is below a threshold then the minimum is chosen. Otherwise a weighted goodness sum is computed, using the certainties as weights. The merged votes, from which the actual commanded arc will be chosen, are displayed in the bottom row of Figure 3, part (4).

7. Finally, the arc with maximum goodness is selected as the next rover step. In case of multiple peaks, the middle arc of the longest adjacent set of votes is chosen. Multiple candidates are indicated in Figure 3, part (4) with a green underline highlight, and the actual arc chosen is tagged with a green box.

As mentioned previously, the process of arc selection repeats until the desired waypoint is reached or some other condition terminates the run. Of course things tend to work better if the rover actually follows the trajectory commanded by GESTALT, but since the path is reevaluated at every step this is not a strict requirement.

5. Baseline System Timings

In Section 3 we described certain optimizations to the Stereo Vision code that take advantage of the vectorized MMX operations available on the Pentium architecture. To provide a benchmark for future studies, in this section we present the (unoptimized) timings of the Stereo Vision and GESTALT Navigation software on four different CPUs. The R3000 CPU is part of a complete rover vehicle, the Athena SDM; the others are simply desktop machines on which the same code was built, and tested on real images logged from an earlier run on the Athena SDM. The desktop results demonstrate the kind of speedup attainable using commercial CPUs, and each of these CPUs has been used to control rovers in the past (or will be used in the near future).

R3000 (Athena SDM) — [1] A prototype of a Mars Rover design that predates MER, the Athena Software Development Model (SDM) rover has an R3000 12 MHz CPU with 2 Kbyte data cache, 4 Kbyte instruction cache, and 32 Mbytes DRAM memory. Its top speed mechanically is about 5 cm/s, but computational constraints limit it to 1 cm/s on average while driving with obstacle avoidance enabled.

RAD6000 — Some simulation timing numbers were generated on a desktop cage not connected to a physical vehicle. This unit has a RAD6000 RISC processor, similar to the type of processor planned for the 2003 Mars Exploration Rover mission, running at 20 MHz with 8 Kbytes L1 cache and 128 Mbytes RAM. Individual timings generated on this system are always in multiples of "ticks", one of which takes at most $\frac{1}{60}$ seconds.

Sun Ultra 10 Workstation — Additional simulation timings come from a desktop workstation not connected to a physical vehicle. This workstation has a Sun Sparc-IIi CPU running at 300 MHz with a 2 Mbyte external (L2) cache and 128 Mbytes RAM.

Linux Workstation — Additional simulation timings come from a desktop workstation not connected to a physical vehicle. This workstation has a Pentium III CPU running at 500 MHz with 32 Kbytes L1 cache, 512 Kbytes L2 cache and 256 Mbytes RAM.

Table 2 presents timing results averaged over eight test runs, which comprise more than 100 individual steps in total. The Athena SDM rover has both forward and rearward facing cameras, but the rearward facing cameras were not used at every step; only if the best forward goodness value was less than the system threshold *min-fwd-threshold*. Use of the rearward-facing cameras accounts for the high variance in Image Grab time (each image grab takes about 5 seconds), and the difference in number of Samples for GESTALT processing v. Actual Driving Time.

While Table 2 reflects the time required for a *complete* navigation cycle, including image acquisition and rover driving,

Table 2. Implementation Timings on the Athena SDM R3000 12 MHz CPU running VxWorks, using a 10 meter × 10 meter Grid with 20 cm × 20 cm Cells. Timings come from 8 separate runs comprising 105 distinct moves.

Step in the Navigation Algorithm	Execution Time (seconds)	Samples
Initializations that occur at system startup		
Initializing Images	0.006 ± 0.00827	8
Initializing JPLStereo	0.016 ± 0.00631	8
Loading camera models	0.009 ± 0.00756	8
Initializing BW Votes	0.013 ± 0.0075	8
Initializing FW Votes	0.017 ± 0.0089	8
Initializing NavRover	0.095 ± 0.0075	8
Image Acquisition		
Time for Camera selection	0.97 ± 0.0162	154
Time for Exposure setting	0.013 ± 0.00727	154
Time for Image grab	7.7 ± 2.7	154
Beginning to process image pair	0.16 ± 0.0115	154
About to load images from memory	0.014 ± 0.00681	154
Images loaded, now Calling MatchStereo	0.117 ± 0.00472	154
Stereo Image Processing: 512×512 down to 128×128		
Pyramid level box filter	0.277 ± 0.00812	154
Time for DoG and pyrlevel 0 downsample	0.83 ± 0.0149	154
Time for rectification	0.013 ± 0.00718	154
Time for pre-processing	0.013 ± 0.00675	154
Time for correl, minim, subpix, lrlos	3.34 ± 0.0202	154
Time for blob filtering	0.161 ± 0.00959	154
Navigation Map Processing		
Time to add disparity map to grid	2.6 ± 0.28	154
Time to generate planes at useful cell	5.2 ± 0.384	154
Time for traversability analysis	2.34 ± 0.0934	154
Time for arc creation	0.014 ± 0.00641	154
Time for path votes	3 ± 1.31	154
Actual Driving Time		
Done driving commanded arc	15 ± 2.92	105
TOTAL TIME (average)	42 seconds $\pm$ 7.80705	

in Table 3 we focus attention on just the vision and navigation components listed in the third and fourth blocks of Table 2. Table 3 thus presents timing results using only a single image pair, but run ten times on different processors to measure timing variances at varying grid resolutions. These timings are more suggestive of what could be achievable on commercial-grade processors.

Note that all of the timings presented in this section have used *unoptimized* code. We fully expect to achieve faster performance in the near future, after using this unoptimized software to validate our general approach to navigation.

The navigation system is still being fine-tuned, and documenting the rover's ability to actually reach a goal is beyond the scope of this paper. But Figure 4 is illustrative of the kind of navigation performance we hope to achieve routinely. During the course of that run through Viking Lander II-type terrain in JPL's Marsyard,[1] the Athena SDM rover successfully avoided large obstacles (plotted from ground truth measurements as ellipsoids in the figure), climbed over many smaller rocks (some of which are also plotted in the figure), and finally terminated its run within the area originally designated, over 17 meters from its starting position. Some other runs were not as successful, the rover having gotten confused by errors in the map caused by inaccurate position estimation.

6. FUTURE MISSIONS

Current designs for a Mars Smart Lander/Rover mission, under study for a possible launch in 2009, call for a 180 day surface mission during which a rover traverses a total of 6 km, in two legs of 3 km each. Detailed scientific investigation would be conducted at three sites, including the landing site and the ends of each 3 km traverse leg. To achieve these goals within tentative time allocations for each aspect of surface operations, the rover would need the ability to traverse up to 675 meters/day, or an average of 6.25 cm/sec over a three hour driving day. During such a traverse, it is necessary to avoid obstacles, desirable to maintain position knowledge

[1] http://robotics.jpl.nasa.gov/infrastructure/marsyard/

Table 3. Simulation Timings, Running GESTALT 10 times on a Single Real Image Pair using a 9 meter x 9 meter Grid at Differing Resolutions

Grid Cell Resolution:	**Pentium III 500 MHz Linux System**		
	50×50 cm^2	**20×20 cm^2**	**10×10 cm^2**
	Time (seconds)	**Time (seconds)**	**Time (seconds)**
Rover and grid initialization	0.00218 ± 5.05e-05	0.00611 ± 0.000208	0.021 ± 0.000276
DoG and pyrlevel 2 downsample	0.285 ± 0.009	0.283 ± 0.0126	0.27 ± 0.00196
Rectification	0.000104 ± 5.87e-06	0.000105 ± 8.7e-06	0.000104 ± 4.8e-06
Pre-processing	3.98e-05 ± 7.48e-07	4.14e-05 ± 1.69e-06	4.02e-05 ± 4e-07
Correl, minim, subpix, lrlos	0.0729 ± 0.00544	0.0813 ± 0.0303	0.0742 ± 0.00886
Blob filtering	0.00311 ± 0.00013	0.0031 ± 9.86e-05	0.00307 ± 4.73e-05
Fit planes	0.0294 ± 0.00652	0.14 ± 0.0101	1 ± 0.00564
Traversability analysis	0.00171 ± 4.01e-05	0.0192 ± 0.000544	0.144 ± 0.0114
Path votes	0.022 ± 0.00118	0.0454 ± 0.000814	0.0838 ± 0.00147
Generating float range XYZ map	0.00436 ± 0.000154	0.00659 ± 0.00665	0.00649 ± 0.00666
TOTAL TIME (average)	**0.421 ± 0.0124**	**0.584 ± 0.035**	**1.61 ± 0.0171**
	Sparc 300 MHz Solaris System		
Rover and grid initialization	0.022 ± 0.00819	0.0258 ± 0.00864	0.0789 ± 0.0163
DoG and pyrlevel 2 downsample	1.06 ± 0.112	1.11 ± 0.185	1.06 ± 0.152
Rectification	0.000241 ± 3.16e-05	0.00301 ± 0.00831	0.00025 ± 7.07e-06
Pre-processing	0.000104 ± 2.25e-06	0.000113 ± 2.8e-05	0.000108 ± 2.1e-06
Correl, minim, subpix, lrlos	0.218 ± 0.0452	0.246 ± 0.049	0.271 ± 0.0815
Blob filtering	0.0177 ± 0.0134	0.0165 ± 0.01	0.0154 ± 0.0105
Fit planes	0.0761 ± 0.0161	0.404 ± 0.0745	2.72 ± 0.35
Traversability analysis	0.00351 ± 0.000341	0.0579 ± 0.0155	0.45 ± 0.106
Path votes	0.0637 ± 0.0155	0.126 ± 0.0236	0.214 ± 0.0511
Generating float range XYZ map	0.0156 ± 0.0116	0.0131 ± 0.00891	0.0154 ± 0.0123
TOTAL TIME (average)	**1.48 ± 0.124**	**2.0 ± 0.208**	**4.82 ± 0.408**
	RAD6000 20 MHz VxWorks System ($\frac{1}{60}$ second resolution)		
Rover and grid initialization	2.42 ± 0.147	2.37 ± 0.0489	2.02 ± 0.145
DoG and pyrlevel 2 downsample	4.65 ± 0.01	4.63 ± 0.0111	4.76 ± 0.194
Rectification	0 ± 0	0 ± 0	0 ± 0
Pre-processing	0 ± 0	0.00167 ± 0.005	0 ± 0
Correl, minim, subpix, lrlos	1.69 ± 0.00834	1.69 ± 0.00764	1.71 ± 0.00764
Blob filtering	0.0967 ± 0.00667	0.095 ± 0.00764	0.0967 ± 0.00667
Fit planes	1.4 ± 0.0289	4.21 ± 0.015	25.1 ± 0.0125
Traversability analysis	0.0567 ± 0.00816	0.685 ± 0.005	5.49 ± 0.013
Path votes	1.15 ± 0	1.69 ± 0.00764	3.06 ± 0.00764
Generating float range XYZ map	N/A	N/A	N/A
TOTAL TIME (average)	**11.5 ± 0.151**	**15.4 ± 0.0544**	**42.2 ± 0.0328**

to 3% of distance traveled, and desirable to conduct "traverse science" observations with onboard instruments.

The stereo vision and GESTALT systems described earlier in this paper provide a baseline obstacle avoidance system for possible use in the Smart Lander/Rover mission. The complexity of these algorithms may increase before 2009 to improve navigation performance in rough terrain. A possible solution to the position estimation goal is a "visual odometry" algorithm [7], which selects and tracks local terrain features in stereo imagery to estimate the motion of the rover during the traverse. Traverse science applications are not yet well specified; possibilities include analyzing point spectrometer data or multispectral imagery for mineral classification.

Since the Smart Lander/Rover is to include active landing hazard avoidance, the mission design is baselining a radiation hardened primary processor for reliable performance of entry, descent, and landing operations. The current strawman choice for this processor is a 133 MHz rad hard PowerPC 750. Options exist to carry more than one of these, if necessary, to handle the required processing load. We are also examining the possibility of flying a commercial grade microprocessor as a co-processor, to provide additional horsepower for rover

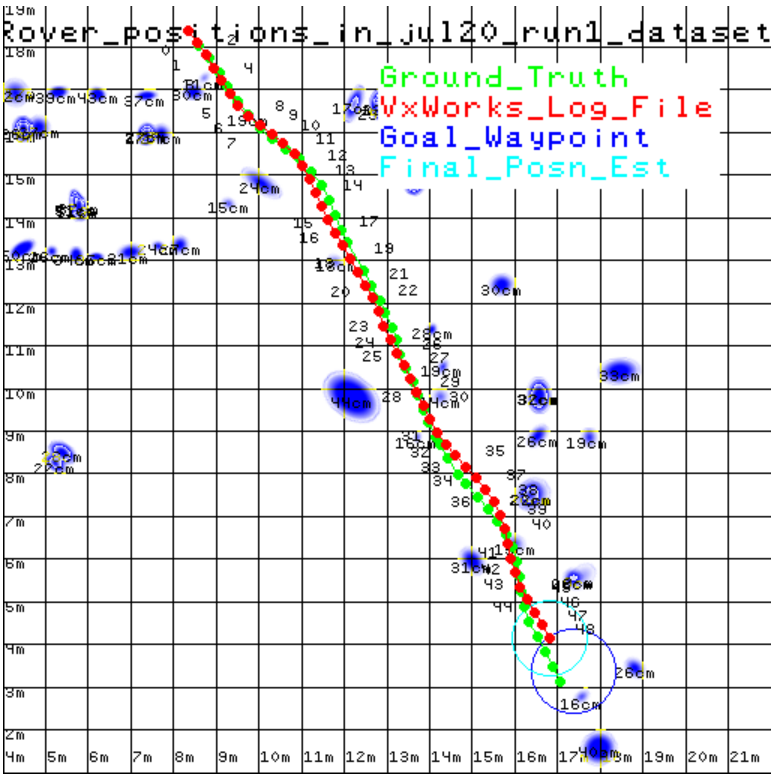

Figure 4. Graphical depiction of a successful 17.5 meter run through VL-2 terrain. The red path indicates the path the rover thought it took, each step of which is numbered; the green path represents the ground truth, as measured by a surveyor's ranging theodolite. Although it actually drove farther than it estimated, the rover did stop within the specified area. Some of the rocks were also measured with the ranging theodolite, and are rendered here as blue ellipses with maximum heights indicated in text.

navigation and science processing. Radiation effects studies of commercial grade PowerPC 750 have concluded that the radiation environment of the surface of Mars will produce one single event upset every 50 hours, and that permanent faults are not expected. To obtain an even greater performance increase, we are considering use of the PowerPC G4 to make available the *Altivec* SIMD instruction unit for acceleration of low level image processing and other mathematical operations. The Altivec unit on the G4 is similar to, but more powerful than the MMX unit on a Pentium 3. We are currently optimizing our stereo code for Altivec, and expect to achieve a speedup even greater than that achieved using MMX on the Pentium.

Performance benchmarking over the coming year will compare the runtime of the stereo vision/GESTALT navigation software on the MER flight processor, the 133 MHz PPC 750 that is the current reference processor for the Smart Lander mission, and a commercial grade 500 MHz PPC G4 we are using for development. Work is also in progress to assess the overall computing workload required for the goals of the Smart Lander mission (eg. for obstacle avoidance, position estimation, traverse science, etc.). Considering both clock rate and architectural advantages (i.e., the Altivec unit), we expect that using a commercial grade PPC G4 as a coprocessor will provide an order of magnitude speedup over a rad hard PPC 750, which itself will be an order of magnitude faster than the MER flight processor. We are currently investigating fault protection schemes that would isolate the effects of single event upsets on the commercial grade processor. Even running everything twice, and voting on the results on the rad hard processor, appears to have a performance advantage of 4× over running all software on the rad hard processor.

7. Conclusion

We described our Stereo Vision and Autonomous Navigation algorithms, and presented baseline timing numbers from example implementations. We demonstrated substantial speedups in the stereo vision software by taking advantage of vectorized instructions available in modern commercial CPUs. We argued that use of such processors in future space missions is achievable and can provide significant speedups to future rover mission capabilities.

8. Acknowledgements

We are grateful to the Athena SDM development team, the Urbie TMR team, and the 2003 MER project personnel for their efforts in support of this work. Special thanks to Jeff Biesiadecki, Rich Petras and Ana Saucedo for their help in documenting the performance of GESTALT during the Summer of 2001. Thanks also to Daniel Limonadi for perspectives on requirements of the Smart Lander mission, and

Raphael Some and Robert Ferraro for information about radiation effects on processors on the surface of Mars. And thanks to Reg Willson for useful feedback on earlier versions of the paper.

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract to the National Aeronautics and Space Administration.

REFERENCES

[1] Jeffrey Biesiadecki, Mark Maimone, and Jack Morrison. The Athena SDM rover: A testbed for mars rover mobility. In *International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, Montreal, Canada, June 2001. http://robotics.jpl.nasa.gov/people/mwm/sdm-mobility/.

[2] Donald B. Gennery. *Calibration and Orientation of Cameras in Computer Vision*, chapter Least-Squares Camera Calibration Including Lens Distortion and Automatic Editing of Calibration Points, pages 123–136. Springer Verlag (A. Gruen and T. Huang, ed.), 2001.

[3] P. Grandjean L. Matthies. Stochastic performance modeling and evaluation of obstacle detectability with imaging range sensors. *IEEE Transactions on Robotics and Automation*, 10(6):783–792, December 1994.

[4] M. Maimone, L. Matthies, J. Osborn, E. Rollins, J. Teza, and S. Thayer. A photo-realistic 3-D mapping system for extreme nuclear environments: Chornobyl. In *International Robotics and Systems Conference (IROS)*, pages 1521–1527, Victora B.C., Canada, October 1998. http://robotics.jpl.nasa.gov/people/mwm/pioneer/iros98/.

[5] Mark Maimone, Issa Nesnas, and Hari Das. Autonomous rock tracking and acquisition from a mars rover. In *International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, pages 329–334, Noordwijk, Netherlands, June 1999. http://robotics.jpl.nasa.gov/tasks/pdm/papers/isairas99/.

[6] L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, and S. Goldberg. A portable, autonomous urban reconnaissance robot. In *Intelligent Autonomous Systems*, Venice, Italy, July 2000. http://robotics.jpl.nasa.gov/tasks/tmr/papers/UrbanRobotPaper0700.pdf.

[7] Clark F. Olson, Larry H. Matthies, Marcel Schoppers, and Mark W. Maimone. Stereo ego-motion improvements for robust rover navigation. In *International Conference on Robotics and Automation*, pages 1099–1104, Seoul, Korea, 2001. http://robotics.jpl.nasa.gov/people/mwm/papers/icra01.pdf.

[8] Bill Ross. A practical stereo vision system. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 148–153, New York, 1993.

[9] Reid Simmons, Lars Henriksen, Lonnie Chrisman, and Greg Whelan. Obstacle avoidance and safeguarding for a lunar rover. In *AIAA Forum on Advanced Developments in Space robotics*, Madison, WI, August 1996. http://www.cs.cmu.edu/~reids/papers/AIAAobsAvoid.pdf.

[10] Sanjiv Singh, Kurt Schwehr, Reid Simmons, Trey Smith, Anthony Stentz, Vandi Verma, and Alex Yahja. Recent progress in local and global traversability for planetary rovers. In *International Conference on Robotics and Automation*, 2000. http://www.frc.ri.cmu.edu/projects/mars/ publications/global_local_icra2000.ps.gz.

[11] P. H. Smith, J. F. Bell III, N. T. Bridges, D. T. Britt, L. Gaddis, R. Greeley, H. U. Keller, K. E. Herkenhoff, R. Jaumann, J. R. Johnson, R. L. Kirk, M. Lemmon, J. N. Maki, M. C. Malin, S. L. Murchie, J. Oberst, R. J. Reid T. J. Parker, R. Sablotny, L. A. Soderblom, C. Stoker, R. Sullivan, N. Thomas, M. G. Tomasko, W. Ward, and E. Wegryn. Results from the mars pathfinder camera. *Science*, 278(5344):1758, December 1997. http://www.sciencemag.org/cgi/content/full/278/5344/1758.

[12] Richard Volpe. Navigation results from desert field tests of the Rocky 7 mars rover prototype. *International Journal of Robotics Research*, 18(7), Special Issue on Field and Service Robots, July 1999. http://robotics.jpl.nasa.gov/people/volpe/papers/JnavMay.pdf.

[13] David Wettergreen, Deepak Bapna, Mark Maimone, and Geb Thomas. Developing nomad for robotic exploration of the atacama desert. *Robotics and Autonomous Systems*, 26(2–3):127–148, February 1999. http://robotics.jpl.nasa.gov/people/mwm/papers/98ras.nomad.pdf.

[14] Yalin Xiong and Larry Matthies. Error analysis of a real-time stereo system. In *Computer Vision and Pattern Recognition*, pages 1087–1093, 1997. http://www.cs.cmu.edu/~yx/ papers/StereoError97.pdf.

9. BIOGRAPHIES

Steve Goldberg *is a researcher at Indelible Systems in Los Angeles, California. He earned a Bachelors of Computer Engineering from the University of Southern California in 1999 where he was an undergraduate research assistant. He currently is working as a contractor at the Jet Propulsion Laboratory.*

Dr. Mark Maimone *is a Machine Vision researcher at the Jet Propulsion Laboratory. In 1989 he completed the International Space University's summer program, in which he participated in the design of a lunar polar orbiter. He earned his Ph.D. in Computer Science from the Computer Science Department of Carnegie Mellon University in 1996, and was then a Postdoctoral Research Associate at Carnegie Mellon's Robotics Institute, serving as Navigation and Software Lead for the 1997 Atacama Desert Trek. Since starting at JPL in 1997, he has worked on the several Mars Rover research projects, and a vision system for inspection of the Chornobyl reactor. Mark is currently a member of the 2003 Mars Exploration Rover flight software team, and is developing the vision and navigation subsystems for the robots that NASA will send to Mars in 2003. His research interests include robotic navigational autonomy, stereo vision, camera calibration, and software environments.*

Dr. Larry Matthies *obtained a Ph.D. degree in Computer Science from Carnegie Mellon University in 1989, then moved to the Jet Propulsion Laboratory, where he is currently Supervisor of the Machine Vision Group. He has extensive experience in 3-D sensing, world modeling, and path planning for robotic navigation. At JPL, he pioneered the development of real-time algorithms for stereo vision-based obstacle detection for autonomous navigation of robotic vehicles and was a member of the team that developed the Sojourner Mars rover. He currently leads projects on computer vision for robotic vehicles sponsored by NASA, DARPA, and the U.S. Army. He is a member of the editorial board of the Autonomous Robots journal and an adjunct professor of Computer Science at the University of Southern California.*