Mapping, Navigation, and Learning for Off-Road Traversal

Kurt Konolige
Willow Garage
Menlo Park, California 94025
e-mail: konolige@willowgarage.com

Motilal Agrawal
SRI International
Menlo Park, California 94025
e-mail: agrawal@ai.sri.com

Morten Rufus Blas
Elektro/DTU University
Lyngby, Denmark
e-mail: mrb@elektro.dtu.dk

Robert C. Bolles
SRI International
Menlo Park, California 94025
e-mail: bolles@ai.sri.com

Brian Gerkey
Willow Garage
Menlo Park, California 94025
e-mail: gerkey@willowgarage.com

Joan Solà
LAAS-CNRS
Toulouse, France
e-mail: joan.scat@gmail.com

Aravind Sundaresan
SRI International
Menlo Park, California 94025
e-mail: aravind@ai.sri.com

Received 8 April 2008; accepted 11 November 2008

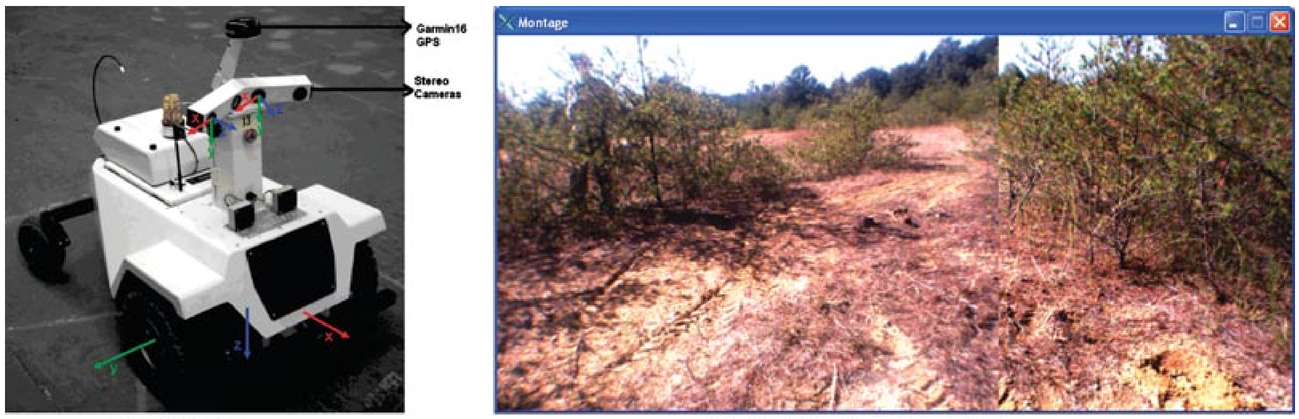
The challenge in the DARPA Learning Applied to Ground Robots (LAGR) project is to autonomously navigate a small robot using stereo vision as the main sensor. During this project, we demonstrated a complete autonomous system for off-road navigation in unstructured environments, using stereo vision as the main sensor. The system is very robust—we can typically give it a goal position several hundred meters away and expect it to get there. In this paper we describe the main components that comprise the system, including stereo processing, obstacle and free space interpretation, long-range perception, online terrain traversability learning, visual odometry, map registration, planning, and control. At the end of 3 years, the system we developed outperformed all nine other teams in final blind tests over previously unseen terrain. © 2008 Wiley Periodicals, Inc.

1. INTRODUCTION

The DARPA Learning Applied to Ground Robots (LAGR) project began in spring 2005 with the ambitious goal of achieving vision-only autonomous traversal of off-road terrain. Further, the participating teams were to be tested "blind"—sending in code to be run on a robot at a remote, unseen site. The hope was that by using learning algorithms developed by the teams, significant progress could be made in robust navigation in difficult off-road environments, where tall grass, shadows, deadfall, and other obstacles predominate. The ultimate goal was to achieve better than 2 × performance over a Baseline system already developed at the National Robotics Engineering Center (NREC) in Pittsburgh, Pennsylvania. All participant teams used the same robotic hardware provided by NREC [Figure 1(a)]; testing was performed by an independent team on a monthly basis at sites in Florida, New Hampshire, Maryland, and Texas.

Although work in outdoor navigation has preferentially used laser range finders (Bellutta, Manduchi, Matthies, Owens, & Rankin, 2000; Guivant, Nebot, & Baiker, 2000; Montemerlo & Thrun, 2004), LAGR uses stereo vision as the main sensor. One characteristic of the vision hardware is that depth perception is good only at fairly short range: its precision deteriorates rapidly after 7 m or so. Even when good stereo information is available, it is often impossible to judge traversability on the basis of three-dimensional (3D) form. For example, tall grass that is compressible can be traversed, but small bushes cannot, and they might have similar 3D signatures. The robots would often slip on sand or leaves and be unable to climb even small grades if they were slippery. These conditions could not be determined even at close range with stereo vision.

Another area that the testing team was keen on developing was the ability of the robots to make decisions at a distance. Many of the tests had extensive



(a) LAGR robot (b) View from the robot cameras

Figure 1. (a) LAGR robot with two stereo sensors. (b) Typical outdoor scene as a montage from the left cameras of the two stereo devices.

cul-de-sacs, dead ends, or paths that initially led toward the goal but then turned away. Here, the robot could not rely on local information to find a good way out. The expectation was that the teams would cope with such situations using long-range vision sensing, that is, be able to tell from the appearance of the terrain whether it was traversable.

Throughout the life of the project, we evaluated the potential of learning methods and appearance-based recognition. The emphasis was always on general methods that would work well in all situations, not just artificial ones designed to test a particular ability, like bright orange fencing that could easily be recognized by its distinctive color. In the end, the most useful and novel technique we developed was an online method for path finding based on color and texture. Although we also developed algorithms for classifying obstacles at a distance, they did not work reliably enough to be included in a final system.

In addition to appearance-based learning, we had to build improved algorithms for many different aspects of vision-based off-road navigation. The paragraphs below summarize the methods that distinguished the SRI system and contributed to its overall performance.

Online Color and Texture Segmentation
It became clear from the early stages of the project that color-only methods for recognizing vegetation or terrain were not sufficient. We concentrated on developing fast combined color/texture methods that could be used online to learn segmentations of the image. These methods advance the state of the art in appearance-based segmentation and are the key to our online path-finding method. They reliably find paths such as the one in Figure 1(b), even when the particular appearance of the path is new.

Precisely Registered Maps
If the robot's reconstruction of the global environment is faulty, the robot cannot make good plans to get to its goal. After noticing navigation failures from the very noisy registration provided by global positioning system (GPS), we decided to give high priority to precise registration of local map information into a global map. Here, we developed real-time visual odometry (VO) methods that are more precise than existing ones while still being computable at frame rates. To our knowledge, this is the first use of VO as the main registration method in an autonomous navigation system. VO enabled us to learn precise maps during a run and so escape efficiently from cul-de-sacs. In the last stage of the project, we also discovered that the precision of VO made it possible to reuse maps from a previous run, thereby avoiding problem areas completely. This *run-to-run learning*, or map reuse, was unique among the teams and on average halved the time it took to complete a course.

Efficient Planner and Controller
The LAGR robot was provided with a "Baseline" system that used implementations of D* (Stentz, 1994) for global planning and the Dynamic Window Approach (DWA) (Fox, Burgard, & Thrun, 1997) for local control. These proved inadequate for real-time control: for example, the planner could take several seconds to compute a path. We developed an efficient global planner based on previous gradient techniques (Konolige, 2000), as well as a novel local controller that takes into account robot dynamics and searches a large space of robot motions. These technqiues enabled the robot to compute optimal global paths at frame rates and to average 85% of its top speed over most courses.

1.1. Performance

It is hard to overemphasize the contribution of consistent map construction and map reuse. Without well-registered maps, the robot would often spend large amounts of time getting cornered as badly remembered obstacles filled in open spaces, or it would reenter dead-end areas that had shifted in the map. Because the testing team emphasized cul-de-sacs and garden path scenarios, it was critical to have accurate representations of areas that were no longer within the short stereo range. As it turned out, an accurate map was a more robust way to deal with these scenarios than unreliable long-range sensing. Further, once the map was constructed, map reuse led to very efficient runs: if you memorize the route, there is no need to repeat your mistakes. Although the idea is simple, the execution was difficult, requiring very precise localization based on VO.

At the end of the project, the teams were tested in a series of courses (Tests 25–27) with a variety of challenges (see Section 6.2). We chose these last tests for inclusion here because our system was complete, having just added the map reuse feature. Over these tests, we averaged about 4 $\times$ the score of Baseline, the best of any team. In each of these tests, our score beat or tied that of the best other team, and in the aggregate, we scored 60% higher than the best other team. These results validate the applicability of our techniques to autonomous navigation.

In this paper we show how we built a system for autonomous off-road navigation that embodies the methods described above and in particular performs online path learning and run-to-run map learning to increase its performance. In the following sections, we first discuss local map creation from visual input, with a separate section on learning color models for paths and traversable regions. Then we examine VO and registration in detail and show how consistent global maps are created and reused. The next section discusses the global planner and local controller. Finally, we present performance results for the last series of tests at the end of the project.

1.2. Related Work

There has been an explosion of work in simultaneous localization and mapping (SLAM), most of it concentrating on indoor environments (Gutmann & Konolige, 1999; Leonard & Newman, 2003). Much of the recent research on outdoor navigation has been driven by DARPA projects on mobile vehicles (Bellutta et al., 2000). The sensor of choice is a laser range finder, augmented with monocular or stereo vision. In much of this work, high-accuracy GPS is used to register sensor scans; exceptions are Guivant et al. (2000) and Montemerlo and Thrun (2004). In contrast, we forgo laser range finders and explicitly use image-based registration to build accurate maps. Other approaches to mapping with vision are those of Rankin, Huertas, and Matthies (2005) and Spero and Jarvis (2002), although they are not oriented toward real-time implementations. Obstacle detection using stereo has also received some attention (Rankin et al., 2005).

VO systems use structure-from-motion methods to estimate the relative position of two or more camera frames, based on matching features between those frames. There have been a number of recent approaches to VO (Johnson, Goldberg, Cheng, & Matthies, 2008; Maimone, Cheng, & Matthies, 2007; Nister, Naroditsky, & Bergen, 2006), including motion estimation on the Mars vehicles (Matthies et al., 2007). Other teams in LAGR also developed VO systems to aid in navigation (Howard, 2008). Our system (Agrawal & Konolige, 2006, 2007; Konolige, Agrawal, & Solà, 2007) is most similar to the recent work of Mouragnon, Lhuillier, Dhome, Dekeyser, and Sayd (2006) and Sunderhauf, Konolige, Lacroix, and Protzel (2005), which exploits bundle adjustment techniques to obtain increased precision. One difference is the introduction of a new, more stable keypoint detector and the integration of an inertial measurement unit (IMU) to maintain global pose consistency. Our system is also distinguished by realtime implementation and high accuracy using a small baseline in realistic terrain. It has been in regular use in demonstrations for more then 2 years as the primary mode of localization and map registration. In addition, the system has been tested over trajectories of up to 9 km with a ground truth RTK-GPS data set and has achieved accuraces of under 1% error (see Section 3.4).

Our segmentation algorithm uses a compact descriptor to represent color and texture. In a seminal paper, Leung and Malik (2001) showed that many textures could be represented and recreated using a small number of basis vectors extracted from the local descriptors; they called the basis vectors *textons*. Whereas Leung and Malik used a filter bank, Varma and Zisserman (2003) later showed that small local texture neighborhoods may be better than using large filter banks. In addition, a small local neighborhood vector can be much faster to compute than multichannel filtering such as Gabor filters over large neighborhoods.

Our planning approach is an enhanced reimplementation of the gradient technique (Konolige, 2000), which computes a global navigation function over the cost map. A similar approach is used in wavefront planning (Latombe, 1991), although wavefront planners usually minimize Manhattan or diagonal distance, whereas we minimize Euclidean distance. Level sets (Kimmel & Sethian 1998) offer an equivalent method for computing paths that minimize Euclidean distance. The underlying computation for all such planners is a variation on dynamic programming (Bellman, 1957). For reasons of efficiency, our planner treats the robot as a holonomic cylinder with no kinodynamic constraints. These constraints could be incorporated into the planner by use of sampling-based algorithms such as rapidly exploring random trees (RRTs) (LaValle, 2006).

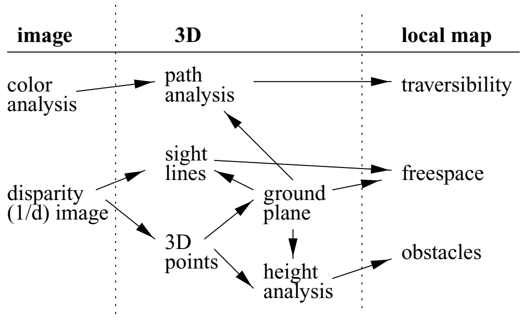
We enforce kinodynamic constraints in our local controller. Control algorithms such as DWA (Fox et al., 1997) compute local controls by first determining a target trajectory in position or velocity space (usually a circular arc or other simple curve) and then inverting the robot's dynamics to find the desired velocity commands that will produce that trajectory. We instead explore the control space directly and simulate and evaluate the resulting trajectories, in a manner reminiscent of the controller used in the RANGER system (Kelly, 1994), with the key differences being the definition of the state space and the trajectory evaluation function. The Stanley controller (Thrun et al., 2006) also rolls out and evaluates possible trajectories but divides them into two categories ("nudges" and "swerves"), based on their expected lateral acceleration. Howard, Green, and Kelly (2007) present a more general approach to constraining the search for controls by first sampling directly in the vehicle's state space.

2. LOCAL MAP CONSTRUCTION

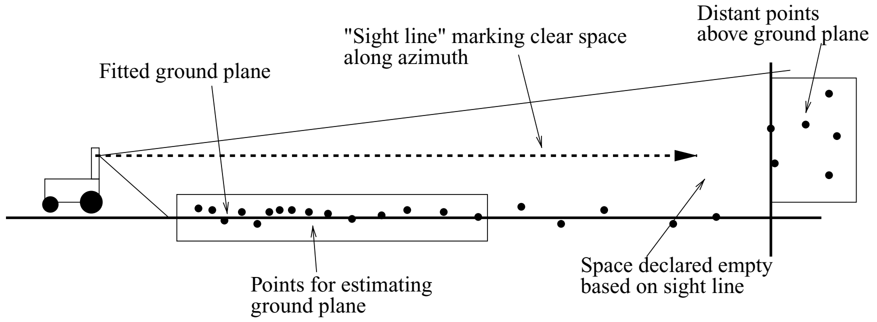
The object of the local map algorithms is to determine, from the visual information, which areas are free space and which are obstacles for the robot: the *local map*. Note that this is not simply a matter of geometric analysis: for example, a log and a row of grass may have similar geometric shapes, but the robot can traverse the grass but not the log.

Figure 2(a) is an outline of visual processing, from image to local map. There are four basic trajectories. From the stereo disparity, we compute a nominal ground plane, which yields free space near the robot. We also analyze height differences from the ground to find obstacles. Via the technique of sight lines we can infer free space to more distant points. Finally, from color, texture, and path analysis, coupled with the ground plane, we determine paths and traversability of the terrain.

All of the processing of local cost maps, with the exception of the color/texture learning, takes place very efficiently. We can run the full algorithm, including stereo computation and obstacle detection, in under 70 ms (15 Hz), enabling very quick response to new features in the environment. Each stereo pair had an associated computer with dual 2 GHz core processors; we used one processor for stereo and cost map analysis, another for visual odometry.



(a) Processing Diagram


Distant points above ground plane
"Sight line" marking clear space along azimuth
Fitted ground plane
Points for estimating ground plane
Space declared empty based on sight line

(b) Stereo point interpretation

Figure 2. Visual processing. In (a), the paths from visual input depict the processing flow in constructing the local map. The interpretation of stereo data points is in (b): nearby points (out to 6 m) contribute to the ground plane and obstacle detection; farther points can be analyzed to yield probable free space ("sight lines") and extended ground planes.

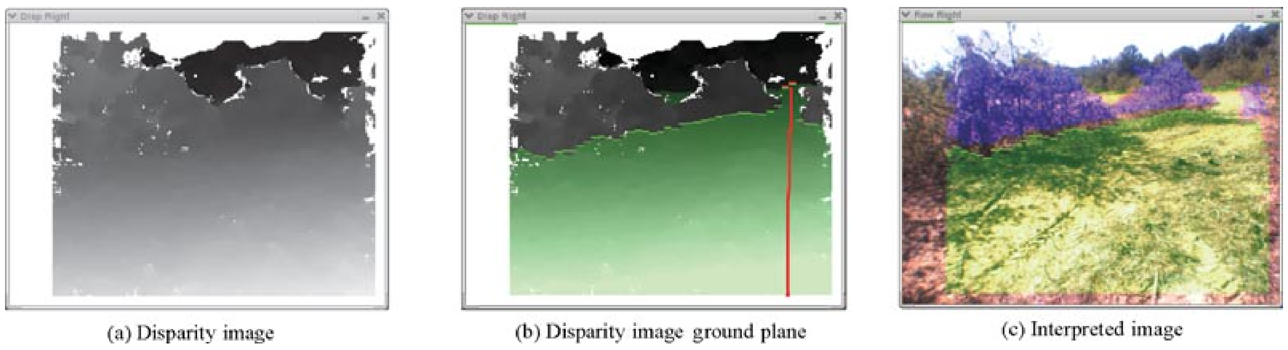


Figure 3. (a) Disparity image from the left stereo pair of the robot in Figure 1. Closer pixels are lighter. (b) Extracted ground plane, in green overlay. Limit of ground plane is shown by green bar; sight line has a red bar. (c) Ground plane overlaid on original image, in green. Obstacles are indicated in purple.

2.1. Stereo Analysis and Ground Plane Extraction

We use a fast stereo algorithm (Konolige, 1997) to compute a disparity image at 512×384 resolution in less than 40 ms[1] [Figure 3(a)]. In typical outdoor scenes, it should be possible to achieve very dense stereo results, The high resolution gives very detailed 3D information for finding the ground plane and obstacles. Each disparity image point $[u, v, d]^\top$ corresponds to a 3D point in the robot's frame ($[x, y, z, w]^\top = R[u, v, d, 1]^\top$ in homogenous coordinates, where R is the reprojection matrix) (Konolige & Beymer, 2007). The matrix multiplication is done for each disparity point as part of stereo processing.

Output from the stereo process is used in a number of ways: the diagram in Figure 2(b) summarizes them. Most of our analysis is biased toward finding free space, especially in areas that are farther from the robot. This strategy stems from the high cost of seeing false obstacles, closing off promising paths for the robot.

The most important geometric analysis is finding the ground plane. Although it is possible to detect obstacles using local variation in height (Happold, Ollis, & Johnson, 2006), using a ground plane simplifies processing and yields more stable results. To extract a ground plane, we use a RANSAC technique (Fischler & Bolles, 1981), choosing sets of three noncollinear points. Hypothesized planes are ranked by the number of points that are close to the plane. Figure 3 shows an example, with a green overlay indicating the inliers. Points that lie too high above the ground plane, but lower than the robot's height, are labeled as obstacles. This method is extremely simple but has proven to work well in practice, even when the ground has modest dips and rises; one reason is that it looks out only to 6 m around the robot. A more sophisticated analysis would break the ground plane into several segments or model more complex shapes. To compute the ground plane efficiently, we subsample the image to 10,000 points and apply the RANSAC algorithm above. Hypothesizing a plane and finding inliers are simple matrix operations, and typical running time is 5 ms.

To find obstacles, a typical algorithm would cluster the 3D points into grid cells on the ground plane. Then, by analyzing the points in each cell, it would be declared ground, obstacle, or unknown. The problem is that there is a geometric mismatch between the 3D cells and the image point density: as the points projected on farther cells become sparse, it is difficult to determine obstacle boundaries. Instead, we find obstacles using algorithms in the disparity plane. The ground plane is projected back onto the disparity points, which are shown in green in Figure 3(b). The disparity image is divided into thin columns, and each column is traversed from the bottom of the image (red line in the image). When the ground plane ends and enough disparity points are found, there must be an obstacle at that endpoint in the ground plane. In practice this technique is much faster and more reliable than ground plane projection, typically consuming only 5 ms.

2.2. Sight Lines

Although we cannot precisely locate obstacles past 6–8 m, we can determine whether there is free

[1]All processing times referenced in this paper are on a 2-GHz Intel CPU.

space, using the following observation. Consider the interpreted image of Figure 3(c). There is a path that goes around the bushes and extends out a good distance. The ground plane extends over most of this area and then ends in a distant line of trees. The trees are too far to place with any precision, but we can say that *there is no obstacle along the line of sight to the trees*. Given a conservative estimate for the distance of the trees, we can add free space up to this estimate; typically we would add free space to at most 25 m. The computation of sight lines is most efficiently accomplished in disparity space, by finding columns of ground plane pixels that lead up to a distant obstacle [red vertical line in Figure 3(b)]. Note that the example sight line follows the obvious path out of the bushes.

2.3. Learning Color and Texture Models

Our learning algorithm uses an online unsupervised segmentation algorithm that uses color and texture to group and cluster similar regions. This segmented image is then used to learn a color and texture model for *path-like* regions in outdoor images. Our segmentation algorithm is based on textons (Leung & Malik, 2001) and is accomplished in two stages. The main design issues have been speed (for real-time segmentation) and robustness (to minimize false-positives).

In the first stage, we cluster color and texture vectors over small local neighborhoods to find a small set of basis vectors (also known as textons) that characterize different scene textures. For reasons of speed, this vector should be as compact as possible without losing appearance characteristics of the region. Angelova, Matthies, Helmick, and Perona (2007) use the three color components over a 5×5 region centered on each pixel. This results in a large 75-dimensional feature vector for each pixel. For speed, we use a 3×3 neighborhood and use the pixel intensity gradients between the surrounding pixels relative to the center pixel to represent texture compactly. We also augment this eight-dimensional feature vector with the 3D color vector of the center pixel in the CIELAB color space. CIELAB has the property that colors are perceptually uniform. The resulting 11-dimensional color/texture feature vector is very compact, and we have found that it still retains the crucial appearance properties to discriminate and segment regions. A detailed comparison with other representations can be found in Blas, Agrawal, Konolige, and Sundaresan (2008).

In the second stage, we cluster histograms of these textons over larger 32×32 regions (which is dependent on the scale of the image) to find more coherent regions with the same mixture of textons using k means as our clustering algorithm. These histograms can be constructed efficiently (irrespective of window size) using integral images (Viola & Jones, 2004). The algorithm is generally set to oversegment the image slightly, as in our case oversegmentation can be dealt with by a subsequent geometrical analysis of the image. Undersegmentation is harder to deal with, as considerable information is lost. The number of clusters was set to eight in the second stage. There are other ways of doing the second stage that provide better results by specifically dealing with boundary conditions, but they are much slower and are thus currently ill-suited for our real-time needs. [See graph-cut (Martin, Fowlkes, & Malik, 2004) and level-sets (Liapis, Sifakis, & Tziritas, 2004).]

2.3.1. Segmentation Results

The University of Southern California (USC) hosts the Brodatz texture database and also provides texture mosaics that are a number of Brodatz textures stitched together in a jigsaw-type pattern. *texmos3* was selected as the texture mosaic for benchmarking our texton descriptors. Figure 4 shows this mosaic along with the ground truth segmentation. This mosaic has eight textures and does not contain color, which tests the descriptors' ability to discriminate textures. Four basic descriptors are tested: a 48-dimensional descriptor composed of the responses from the Leung–Malik (Leung & Malik, 2001) filter bank (LM, 32); a 75-dimensional descriptor of 5×5 raw red–green–blue (RGB, 5×5, 32) values as

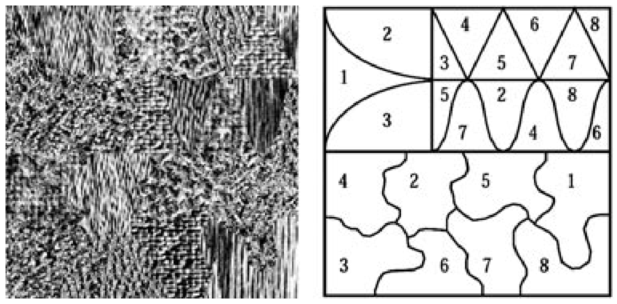


Figure 4. Synthetic texture mosaic used (provided by USC via its website). The left image is the texture mosaic. The right image shows which texture regions belong to which texture.

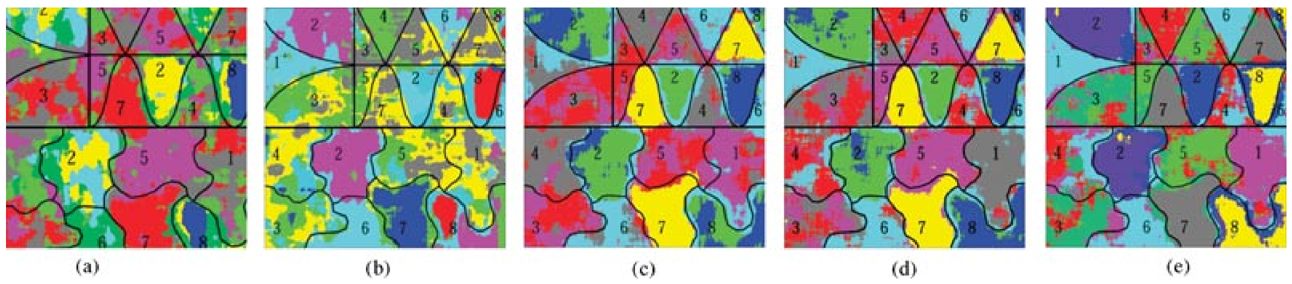


Figure 5. Results for the synthetic texture segmentation. Each color represents a different histogram cluster. An overlay shows which regions should have homogeneous colors. (a) LM filter, 32; (b) RGB 5 × 5, 32; (c) LBP 3 × 3, 32; (d) SRI 3 × 3, 32; and (e) SRI 5 × 5, 64.

used in Angelova et al. (2007) (which in effect is 25-dimensional on gray-scale images); the local binary pattern (Mäenpää & Pietikäinen, 2005) (LBP) in a 3 × 3 neighborhood (LBP, 3 × 3, 32); and two versions of our descriptor: the 3 × 3 neighborhood (11-dimensional with the L,a,b color components set to zero) (SRI, 3 × 3, 32) and a 5 × 5 neighborhood (SRI, 5 × 5, 64) with the descriptor components still being the intensities minus the center intensity.

For the test, the LM filter bank is the only one in which the descriptors are not learned on the image itself. For all other descriptors, 32 textons are learned from the image itself. Our 5 × 5 version used 64 textons illustrating our best possible result. The lack of color information meant that more textons were needed to discriminate the textures. The second stage of clustering is then applied to give the segmentation results. It is important to note that the underlying segmentation algorithm is the same for each of these descriptors.

The actual segmentations obtained for each descriptor can be seen in Figure 5. Each descriptor is then scored using two scores: the detection rate and the confusion rate. The detection rate gives a measure of how much of a given texture it managed to classify correctly. The confusion rate gives a measure of how many correct versus false detections to expect. A good segmentation will have a high detection rate and a low confusion rate.

The total confusion and detection rates are shown in Table I. The LM filter bank performs the worst, as it has higher confusion and lower detection rates than all the other descriptors. The raw intensity value descriptor also performs poorly. LBP has problems discriminating between textures 2 and 8 but is otherwise clearly better than the raw intensities and LM filter bank. Our descriptors do a much better job at discriminating between textures 2 and 8, which indicates that the intensity gradients are necessary to do this and that it is not enough to rely just on the gradient direction. All the methods find it hard to discriminate between textures 3 and 4 except the LBP, which aids it greatly in the total scores. The results for our descriptor are on average better than those of the other methods on this data set. Interestingly, for our descriptors the 3 × 3 version actually gets a better total detection rate than the 5 × 5 version at the cost of a higher total confusion score.

Table I. Total confusion and detection rates for different types of descriptors.

%	LM, 32	RGB	LBP	SRI, 3 × 3	SRI, 5 × 5
Total confusion rate	50	56	46	38	34
Total detection rate	40	53	68	79	68

2.3.2. Learning Paths

We use our segmentation algorithm to learn and subsequently recognize both natural and man-made *paths* in outdoor images. Paths are characterized by their color, texture, and geometrical properties. Training samples for a path can come from teleoperation or from a priori knowledge that the robot is starting on a path. The robot can also search for paths by trying to identify image clusters that have the geometry of a path. We deal with oversegmentation of the path (wherein a path is split into multiple segments due to possibly differing textures) by grouping multiple segments based on their overall geometry. We compute geometrical properties of the path that could be composed of a single or multiple segments. The properties include width, length, and spatial continuity of the path in order to verify whether it geometrically resembles a path. These geometrical properties are computed in three dimensions using the ground plane information available from the stereo cameras and are hence not affected by the perspective projection. We assume a fixed path width (but allow a certain deviation from this assumption). Once a path is identified, the robot learns the texton histograms of the component segments as a model for the path. This model can be used to identify even paths that are partially outside the field of view (FOV) by allowing the path to end prematurely at the image boundary.

For classification, each pixel is first labeled using the shortest Euclidean distance on the color/texture vector at this pixel to the clustered textons. Likewise histograms of textons at each pixel are classified using Euclidean distance to the clustered histograms. The texton histograms from our training provide positive examples (the histograms that belong to a path) as well as negative examples (the histograms that do not belong to a path). This helps prevent false-positives. A final geometrical analysis of the labeled histograms makes sure that potential path regions have the right geometry.

The learning process runs at 1 Hz for training on a single image and is typically performed at the beginning of a run (although it could be performed at regular intervals to update the path model). Classification based on the learned model runs at around 5 Hz. Figure 6 shows the various steps of our algorithm on one of our test runs. The path between bushes is identified in yellow in Figure 6(d). Details on this algorithm can be found in Blas et al. (2008).

2.4. Results of Local Map Construction

The combined visual processing results in local maps that represent traversability with a high degree of fidelity. Figure 7 shows the results of an autonomous run of about 130 m, over a span of 150 s. We used offline learning of mulch paths on a test site and then used the learned models on the autonomous run. The first part of the run was along a mulch path under heavy tree cover, with mixed sunlight and deep shadows. Cells categorized as path are shown in yellow; black is free space. Obstacles are indicated by purple (for absolute certainty) and white-to-gray for decreasing certainty. We did not use sight lines for this run.

The path did not lead directly to the goal, and there were many opportunities for the robot to head cross country. About two-thirds of the way through the run, no more paths were available, and the robot went through heavy grass and brush to the goal. The robot's pose, as estimated from filtered VO (see Section 3.2), is in green; the filtered GPS path is in yellow. Because of the tree cover, GPS suffered from high variance at times.

A benefit of using VO is that wheel slips and stalls are easily detected, with no false positives (Section 5.4). For example, at the end of the run, the robot was caught on a tree branch, spinning its wheels. The filtered GPS, using wheel odometry, moved far off the global pose, while the filtered visual odometry pose stayed put.

3. CONSTRUCTING CONSISTENT GLOBAL MAPS

In this section we provide solutions to two problems: representing and fusing the information provided by visual analysis, and registering local maps into a consistent global map.

3.1. Map Representation

For indoor work, a standard map representation is a two-dimensional (2D) *occupancy grid* (Moravec & Elfes, 1985), which gives the probability of each cell in the map being occupied by an obstacle. Alternatives for outdoor environments include 2.5-dimensional elevation maps and full 3D voxel maps (Iagnemma, Genot, & Dubowsky, 1999). These representations can be used to determine allowable kinematic and dynamic paths for an outdoor robot in rough terrain. We choose to keep the simpler 2D occupancy grid, foregoing any complex calculation of the robot's interaction with the terrain. Instead, we abstract the geometrical characteristics of terrain into a set of categories and fuse information from these categories to create a *cost* of movement.

We use a grid of 20×20 cm cells to represent the global map. Each cell has a probability of belonging to each of the four categories derived from visual analysis (Section 2): obstacle, ground plane free space, sight line free space, and path free space. Note that these categories are not mutually exclusive because, for example, a cell under an overhanging branch could have both path and obstacle properties. We are interested in converting these probabilities

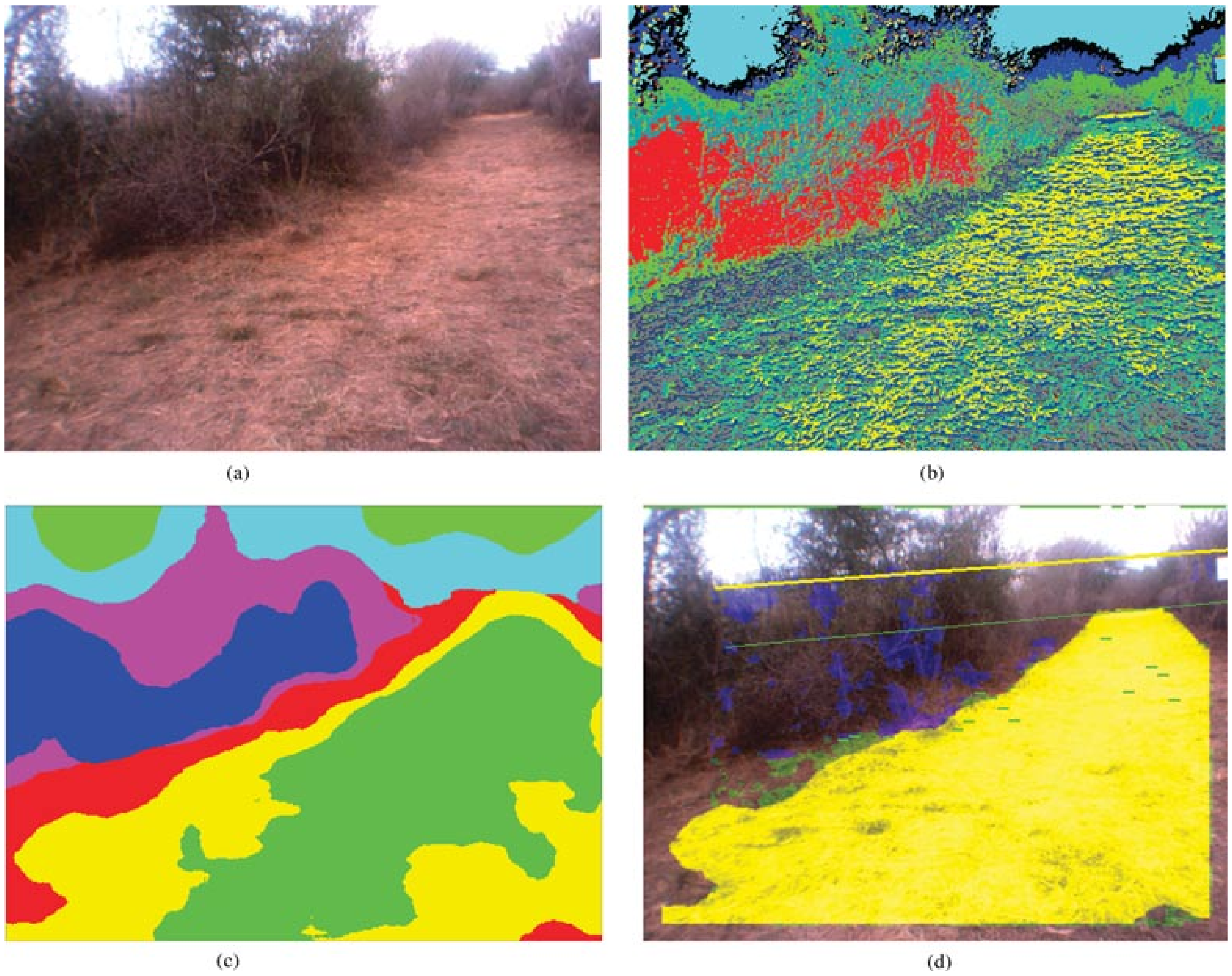


Figure 6. Various steps of our segmentation algorithm on a typical outdoor image. (a) The image from one of the stereo cameras. (b) Each pixel assigned to a texton. (c) Each histogram of textons gets assigned to a histogram profile. In this particular example, the path is composed of two segments (green and yellow). (d) A path is recognized (in yellow).

into a cost of traversing the cell. If the probabilities were mutually exclusive, we would simply form the cost function as a weighted sum. With nonexclusive categories, we chose a simple prioritization schedule to determine the cost. Obstacles have the highest priority, followed by ground plane, sight lines, and paths. Each category has its own threshold for significance: for example, if the probability of an obstacle is low enough, it will be ignored in favor of one of the other categories. The combination of priorities and thresholds yields a very flexible method for determining costs. Figure 7 shows a color-coded version of computed costs.

3.2. Registration and Visual Odometry

The LAGR robot is equipped with a GPS that is accurate to within 3–10 m in good situations. GPS information is filtered by the IMU and wheel encoders to produce a more stable position estimate. However, because GPS drifts and jumps over time, it is impossible to differentiate GPS errors from other errors such as wheel slippage, and the result is that local maps cannot be reconstructed accurately. Consider the situation of Figure 8 and 9. Here the robot goes through two loops of 10-m diameter. There is a long linear feature (a low wall) that is seen as an obstacle at the

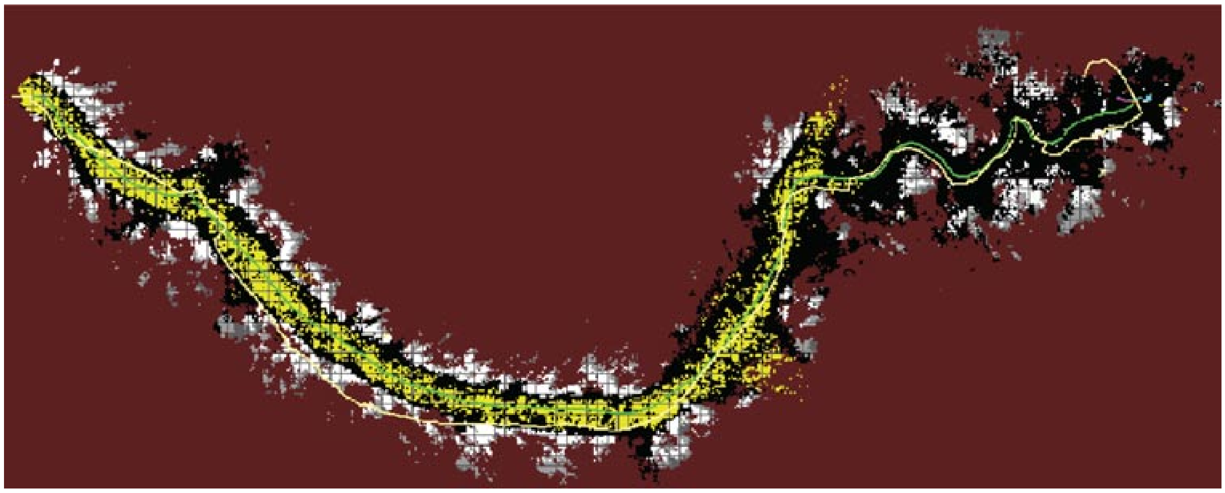


Figure 7. Reconstruction on a 130-m autonomous run. Yellow is recognized path, black is free space, and white and gray are obstacles.

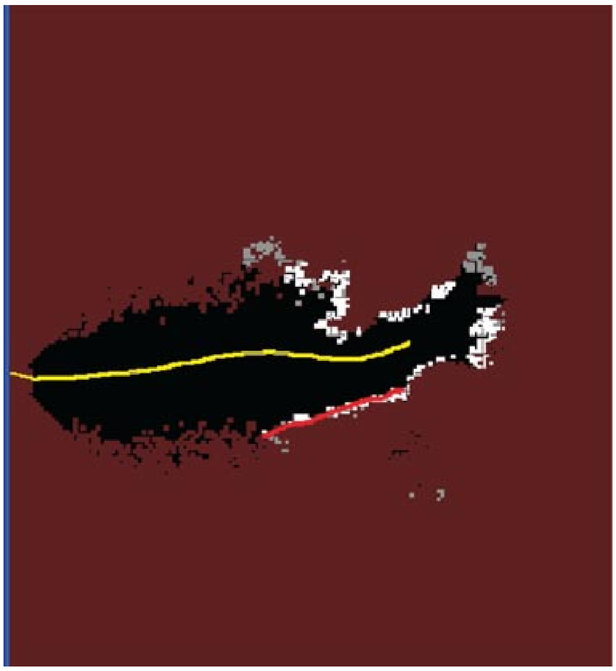


Figure 8. Three stages during a run using GPS filtered pose. Obstacle points are shown in white, free space in black, and the yellow line is the robot's path. The linear feature is marked by hand in red in all three maps, in its initial pose. Map extent is 35 m on a side.

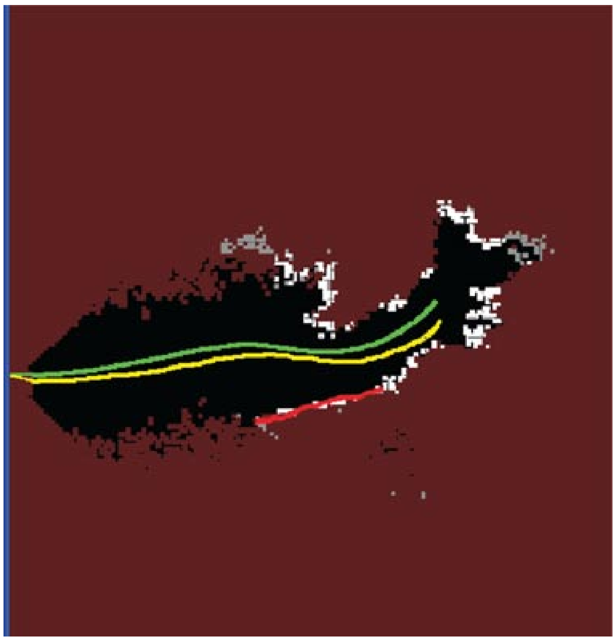


Figure 9. VO in the same sequence as Figure 8. GPS filtered path in yellow; VO filtered path is in green.

beginning and end of the loops. Using the filtered GPS pose, the position of the wall shifts almost 2 m during the run, and obstacles cover the robot's previous tracks.

Our solution to the registration problem is to use VO to ensure local consistency in map registration. Over larger regions, filtering VO with GPS information provides the necessary corrections to keep errors from growing without bounds. We describe these techniques in the next two sections.

The LAGR robot presents a challenging situation for visual odometry: wide FOV and short baseline make distance errors large, and a small offset from the ground plane makes it difficult to track points over longer distances. We have developed a robust VO solution that functions well under these conditions. We briefly describe it here; for more details consult Agrawal and Konolige (2006) and Konolige et al. (2007).

For each new frame, we perform the following process.

1. Distinctive features are extracted from each new frame in the left image. Standard stereo methods are used to find the corresponding point in the right image.
2. Left-image features are matched to the features extracted in the previous frame using our descriptor. We use a large area, usually around 1/5 of the image, to search for matching features.
3. From these uncertain matches, we recover a consensus pose estimate using a RANSAC method (Fischler & Bolles, 1981). Several thousand relative pose hypotheses are generated by randomly selecting three matched noncollinear features and then scored using pixel reprojection errors.
4. If the motion estimate is small and the percentage of inliers is large enough, we discard the frame, because composing such small motions increases error. A kept frame is called a *key frame*. The larger the distance between key frames, the better the estimate will be.
5. The pose estimate is refined further in a sparse bundle adjustment (SBA) framework (Engels, Stewnius, & Nister, 2006; Triggs, McLauchlan, Hartley, & Fitzgibbon, 2000). SBA is a nonlinear batch optimization over camera poses and tracked features. An incremental form of SBA can reduce the error in VO by a large factor at very little computational overhead. A feature that is long lived, that is, that can be tracked over more frames, will give better results.

Precise VO depends on features that can be tracked over longer sequences. Hence, the choice of a feature detector can have a large impact in the performance of such a VO system. Harris corner features are widely used for VO. We have found that although Harris corners give good results and are very efficient to compute, they fail in many situations in outdoor environments. In addition, these features are not very stable, resulting in very short track lengths. Other widely used feature detectors such as SIFT (Lowe, 2004) and SURF (Bay, Tuytelaars, & Gool, 2006) work well but are not suitable for a real-time system. We have developed a novel feature (named CenSurE) (Agrawal, Konolige, & Blas, 2008) that has improved stability and is inexpensive to compute. Whereas the basic idea of CenSurE features is similar to that of SIFT, the implementation is extremely efficient, comparable to Harris. Just as SIFT approximates the Laplacian of Gaussian with difference of Gaussians, CenSurE features approximate the LOG with bilevel center-surround filters. The extreme simplicity of these filters makes them extremely fast to compute but without sacrificing performance. Figure 10 shows a progression of bilevel filters with various degrees of symmetry. The circular filter is the most faithful to the Laplacian but hardest to compute. The other filters can be computed rapidly with integral images with decreasing cost from octagon to hexagon to box filter. Further details of our CenSurE feature detector are described in Agrawal et al. (2008). Figure 11 shows the CenSurE features tracked over several frames.

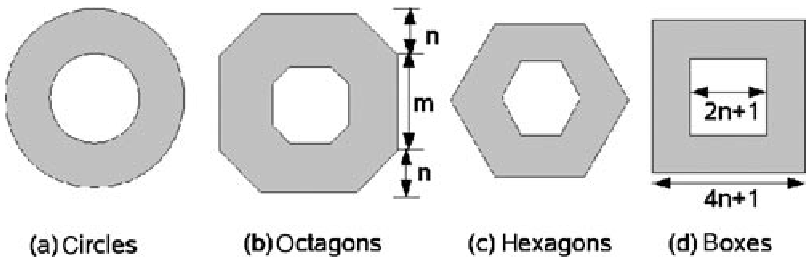


Figure 10. Progression of center-surround bilevel filters. (a) Circular symmetric BLOG (bilevel LOG) filter. Successive filters (octagon, hexagon, box) have less symmetry.

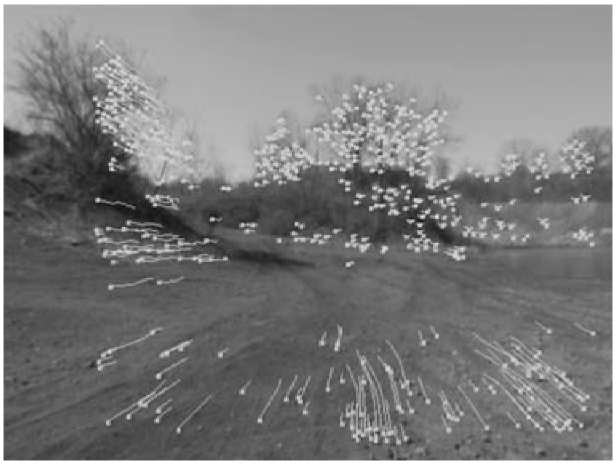


Figure 11. CenSurE features tracked over several frames.

The IMU and the wheel encoders are used to fill in the relative poses when VO fails. This happens due to sudden lighting changes, fast turns of the robot, or lack of good features in the scene (e.g., blank wall).

3.3. Global Consistency

Bundle-adjusted incremental motions between consecutive frames are chained together to obtain the absolute pose at each frame. Obviously, this is bound to result in accumulation of errors and drifting. We use GPS and the IMU to correct the pose of the vehicle. We perform two types of filtering:

1. Gravity normal: The IMU's accelerometers measure the gravity normal in vehicle frame together with vehicle accelerations. Vehicle accelerations have zero mean in the long run and can therefore be considered as white perturbations of the gravity measurements. We apply regular extended Kalman filter (EKF) corrections to the tilt and roll angles. By assigning very large noise values to the perturbing accelerations (we used $10g$ standard deviation), the effect is imperceptible in the short term but sufficient to cancel the long-term angular drifts, otherwise unbounded.
2. GPS yaw: The IMU yaw data are very bad and cannot be used for filtering (for example, over the 150-m run, they can be off by 60 deg). Instead, we used the yaw estimate available from the LAGR GPS. These yaw estimates are comparable to a good-quality IMU. Over a very long run, the GPS yaw does not have an unbounded error, as would an IMU, because it is globally corrected.

To maintain globally consistent maps, we have turned off any position filtering based on GPS. We completely ignore position estimates from the GPS in calculating our pose. In addition, to limit the effect of velocity noise from GPS on the heading estimate, GPS yaw is used only when the GPS receiver has at least a 3D position fix and the vehicle is traveling 0.5 m/s or faster. Our filter is a simple linear filter that nudges the tilt/roll (for gravity normal) and yaw (for GPS yaw) toward global consistency, while maintaining local consistency.

The quality of the registration from filtered VO, shown in Figure 9, can be compared to the filtered GPS of Figure 8. The low wall, which moved almost 2 m over the short loops when using GPS, is much more consistent when VO is employed. And in cases in which GPS is blocked or degraded, such as under heavy tree cover in Figure 7, VO still produces maps that are locally consistent. It also allows us to determine wheel slips and stalls with almost no false positives; note the end of the run in Figure 7, where the robot was hung up and the wheels were slipping and wheel odometry produced a large error.

3.4. Results of Visual Odometry

In Test 17, the testing team surveyed a course using an accurate RTK-GPS receiver. The "Canopy Course" was under tree cover, but the RTK GPS and the LAGR robot GPS functioned well. Sixteen waypoints were surveyed, all of which were within 10-cm error according to the RTK readout. (One waypoint was deemed inaccurate and not included.) The total length of the course was about 150 m. Subsequently, the LAGR robot was joysticked over the course, stopping at the surveyed points. The robot was run forward over the course and then was turned around and sent backward to the original starting position.

The course itself was flat, with many small bushes, cacti, downed tree branches, and other small obstacles. Notable for VO was the sun angle, which was low and somewhat direct into the cameras on several portions of the course. Figure 12 shows two images acquired by the robot. The left image shows a good scene in the shadow of the trees, and the right image shows a poor image where the sun washes

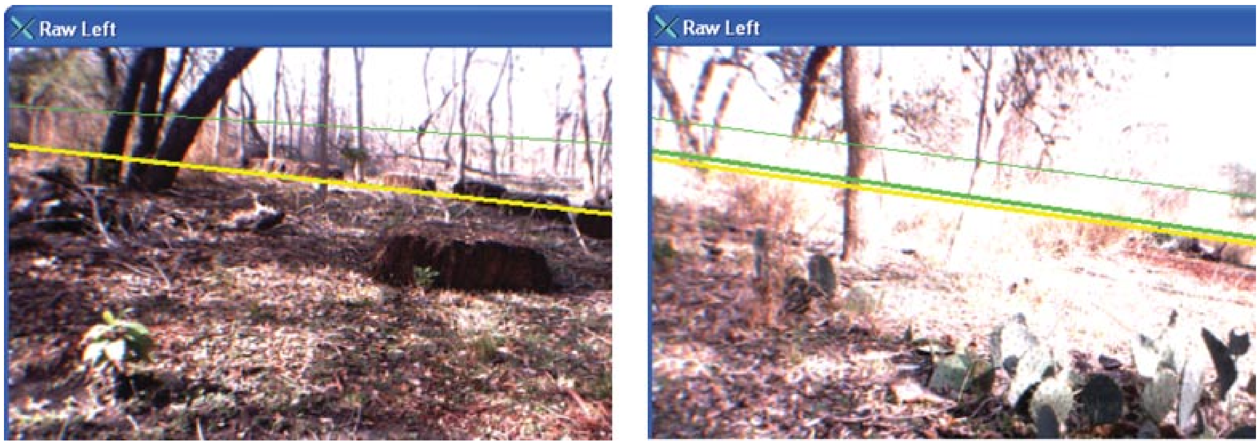
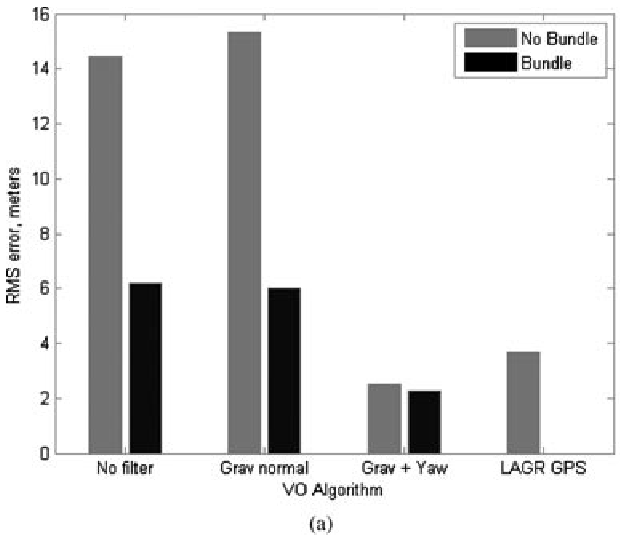


Figure 12. Images from the Canopy data set.

out a large percentage of the scene. (The lines in the images are horizon lines taken from VO and from ground plane analysis.) The uneven image quality makes it a good test of the ability of VO under realistic conditions.

Because the initial heading of the robot is unknown, we used an alignment strategy that assumes that there is an initial alignment error and corrects it by rotating the forward VO path rigidly to align the endpoint as best as possible. This strategy minimizes VO errors on the forward path and may underestimate them. However, for the return path, the errors will be caused only by VO and can be taken as a more accurate estimate of the error.

For this test, our CenSurE features were not ready, and we were able to match frames along the whole route using Harris corners. Figure 13(a) shows the RMS error between VO (with different filters) and the RTK waypoints, on the return path. As noted above, the forward VO path of the robot has been



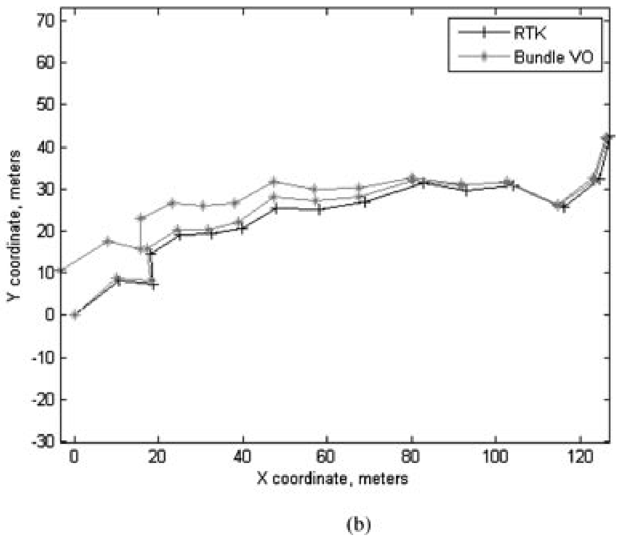


Figure 13. Results of VO on the Canopy data set. (a) RMS error between VO (with different filters) and the RTK waypoints, on the return path. (b) Trajectory of bundle-adjusted VO (without any filtering) compared to RTK groundtruth.

aligned with the RTK path. As can be seen, the best results are obtained using bundle-adjusted VO with gravity normal and GPS yaw filtering. In this case, the errors between waypoints is very small, amounting to <1% of distance traveled. Without filtering, the results are worse [(Figure 13(b)], amounting to about 3% of distance traveled. At some points in the middle of the return trip, the VO angle starts to drift, and at the end of the backward trip there is about a 10-m gap. Note that this effect is almost entirely caused by the error in the yaw angle, which is corrected by GPS yaw. It is also worth mentioning that the use of CenSurE features substantially improves the performance of VO, although we do not have results of using CenSurE on this data set.

We present results of VO with CenSurE features on two other large outdoor data sets. These data sets were collected using a larger tank-like vehicle called Crusher [also developed by the National Robotics Engineering Center (NREC), Pittsburgh, under DARPA's Unmanned Ground Vehicle–PerceptOR Integration (UPI) program]. They have frame-registered ground truth from RTK GPS, which is accurate to several centimeters in XY and 10 cm in Z. For these data sets, the camera FOV is 35 deg, the baseline is 50 cm, and the frame rate is 10 Hz (512×384), so there is often large image motion. We took data sets from Little Bit (9-km trajectory, 47,000 frames) in Pennsylvania and Ft. Carson (4 km, 20,000 frames) in Colorado, to get variety in imagery. The Ft. Carson data set is more difficult for matching, with larger motions and less textured images. In the experiments, we use only CenSurE features, which failed the fewest times (0.17% for Little Bit, 4.0% for Ft. Carson).

The VO angular errors contribute nonlinearly to trajectory error. On the two data sets, we compared RMS and max XYZ trajectory errors. In the case of matching failure, we substituted IMU data for the angles and set the distance to the previous value. In Table II, the effects of bundle adjustment and IMU filtering are compared.

In both data sets, IMU filtering plays the largest role in bringing down error rates. This is not surprising, because angular drift leads to large errors over distance. Even with a noisy IMU, global gravity normal will keep Z errors low. The extent of XY errors depends on how much the IMU yaw angle drifts over the trajectory: in our case, a navigation-grade IMU has 1 deg/h of drift. Noisier IMU yaw data would lead to higher XY errors.

Table II. Trajectory error statistics, in meters and percent of trajectory.

		RMS error in XYZ	Max error in XYZ
Little	VO No SBA	97.41 (1.0%)	295.77 (3.2%)
Bit	VO SBA	45.74 (0.49%)	137.76 (1.5%)
(9 km)	VO No SBA + IMU	7.83 (0.08%)	13.89 (0.15%)
	VO SBA + IMU	4.09 (0.04%)	7.06 (0.08%)
Ft.	VO No SBA	263.70 (6.9%)	526.34 (13.8%)
Carson	VO SBA	101.43 (2.7%)	176.99 (4.6%)
(4 km)	VO No SBA + IMU	19.38 (0.50%)	28.72 (0.75%)
	VO SBA + IMU	13.90 (0.36%)	20.48 (0.54%)

The secondary effect is from SBA. With or without IMU filtering, SBA can lower error rates by one half or more, especially in the Ft. Carson data set, in which the matching is less certain.

3.5. Map Reuse

VO and IMU/GPS filtering enable us to construct consistent maps on a single run. These maps are useful for getting out of traps and cul-de-sacs in the environment, which occurred quite frequently. In fact, the testing team was interested in long-range sensing capabilities and would use natural or constructed traps as a way of rewarding robots that could detect them from a distance. Unfortunately, the vision sensors on the robots were not very capable at a distance [see Section 6 and Figure 18(a)]. So, our strategy was to use map information learned in the first run to compute an optimal path for the second and subsequent runs. This type of learning, *run-to-run learning*, turned out to be the most powerful form of learning for the tests and the key to performing better than any other LAGR team.

Our first successful test of map learning and reuse was in Test 25 at the end of the project [Figure 14 and later in Figure 18(a)]. The direct line to the goal was through a small copse of trees, where there were barriers of deadfall and tall grass. In the first run, the robot wandered through this area, eventually finding a way out to the goal. In the second run, the robot started with the map constructed on the first run and headed around the problem area. Note that the robot actually started into the cul-de-sac and then decided to go around. The planner had a finite horizon of about 40 m and recognized the blockage only at that

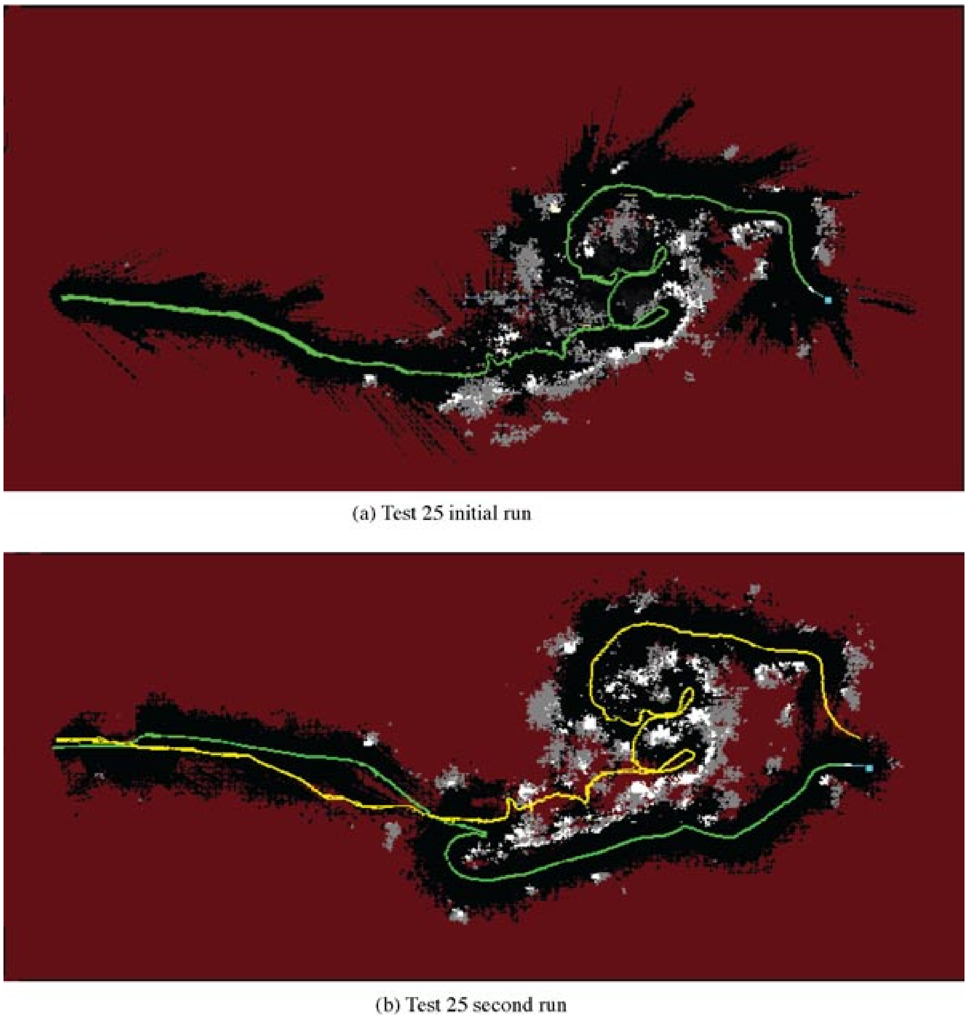


Figure 14. Map reuse during Test 25. The global map in (a) shows the first run: black is free space (including long sight-lines), and white and gray are obstacles. The robot path estimated from VO is the yellow line. The starting position of the robot is the left side of the screen; the goal is on the right at about 80 m. Note the many extended concave obstacles and cul-de-sacs. Image (b) shows the robot's trajectory for the second run in green, bypassing the cul-de-sac obstacles and heading around to the right. The original run is superimposed.

point. In subsequent tests we extended the horizon of the planner to the goal.

Our map-reuse technique is simple: at the start of a run, match the robot's view to the start of the previous run, using the same method as for matching frames in VO. If a good match is found, the map from the previous run is brought in and adjusted to the robot's current position. From this point the robot's position on the old map is "open loop," that is, there is no reregistration or localization of the robot within the map. Because VO performance is generally within 1% over 100 m, this strategy was overwhelmingly successful during the tests. Still, a true visual SLAM algorithm would work better in more difficult conditions, and we have made significant progress here, closing loops over 5-km data sets (Konolige & Agrawal, 2008), but unfortunately this research was done too late to incorporate into the LAGR system.

4. PLANNING

The LAGR robot was provided with a "Baseline" system that used implementations of D* (Stentz, 1994) for global planning DWA (Fox et al., 1997) for local control. Using this system, we (as well as other teams) had frequent crashes and undesirable motion. The main causes were the slowness of the planner and the failure of the controller to sufficiently account for the robot's dynamics. The D* planner is optimized for very large-scale environments. It uses dynamic programming to compute the minimum-cost potential to the goal at each cell; it needs significant resources to maintain the indices necessary to unravel the minimum-cost computations incrementally. In our environments (100×200 m, 20 cm^2 cells) it would take many seconds to compute a plan, even when only a small portion of the map was filled. For large-scale maps this may be acceptable, but we need much faster response to tactical maneuvers over smaller scales (e.g., cul-de-sacs).

Instead, we reimplemented a gradient planner (Konolige, 2000; Philippsen & Siegwart, 2005) that computes optimal paths from the goal to the robot, given a cost map. The gradient planner is a wavefront planner that computes the cost of getting to a goal or goals at every cell in the workspace. It works by using a local neighborhood to update the cost of a cell. If the cell's cost is higher than the cost of a neighbor cell plus the local transit cost, then it is updated with the new cost. The overall algorithm starts by initializing the goal with a zero cost and everything else with a very large cost. All goal cells are put onto an "open" list. The algorithm runs by popping a cell of the open list and updating each of the cell's neighbors. Any neighbor that has a lowered cost is put back onto the open list. The algorithm finishes when the open list is empty.

There are many variations on this algorithm that lead to different performance efficiences. Our algorithm has several unique modifications:

- Unlike other implementations, it uses a true Euclidean metric, rather than a Manhattan or diagonal metric, in performing the update step (Kimmel & Sethian 1998). The update can be performed on the four nearest neighbors of a cell. Generally speaking, the two lowest-cost neighbors can be used to determine the direction of propagation of the cost potential and the cell updated with an appropriate distance based on this direction.
- The algorithm computes the configuration space for a circular robot and includes safety distances to obstacles. This is one of the interesting parts of the gradient method. Because there is already a method for computing the distance transform from a set of points, the configuration space can be computed efficiently. The obstacle points are entered as goal points, and the update algorithm is run over each of these points, generating a new open list. Each open list is processed fully, leading to a sequence of open lists. At the end of n cycles, the distance to obstacles has been determined up to $n * c$, where c is the cell size. Usually this is done to a distance of three or four times the robot radius, enough to establish a safety cushion to the obstacle. Finally, a cost is associated with the distance: an infinite cost within the robot radius to an obstacle and a decreasing cost moving away from this.
- The queue handling is extremely efficient, using threshold-based queues, rather than a best-first update, which has high overhead for sorting. Instead, we use a two-priority-queue method. A threshold shuttles new cells to one queue or the other, depending on whether their cost is greater or less than the threshold. The low-cost queue is always processed first. When no more cells remain in it, the threshold is increased, the second queue becomes the low-cost queue, and a new high-cost queue is initialized. This queue strategy is the key to the good performance of the algorithm: each update step happens very rapidly. Although the complexity of the algorithm is the order of the area to be covered, and there is no "best-first" search from the goal to the robot position, still the extreme rapidity of each step makes it possible to cover reasonable areas (e.g., 80×80 m) in several tens of milliseconds.
- Rapid switching of global paths is avoided by including hysteresis: lowering the cost along the path. There is a trade-off between sticking to the current path and exploring some new path if current readings indicate it might be better. We lower the cost enough so that it takes a significant amount of new information to turn the path aside.

Typically we run the global planner within a subregion of the whole map because the robot is continuously moving toward the goal and encountering new areas. On longer runs, up to 200 m, we use an 80×80 m area; the global planner runs in about

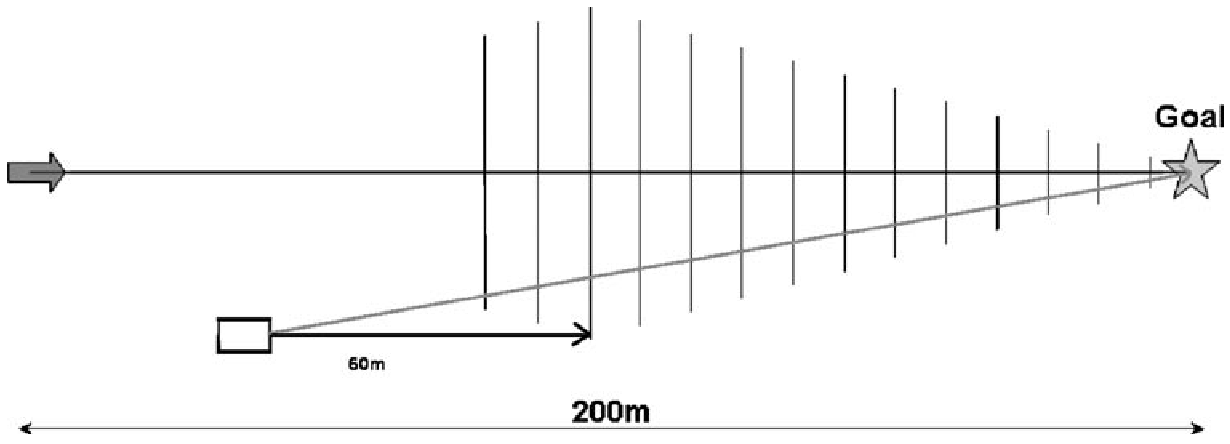


Figure 15. Line goals for a robot in a 200-m environment. The line goal is placed 60 m ahead of the robot, and its extent varies with the distance to the goal.

30 ms in this region. Unless there is a large cul-de-sac, longer than 80 m, this area is sufficient to maneuver the robot tactically around obstacles. For more global planning, which occurs when starting a run with a previously made map, we run the planner over the whole area, which can take up to 100 ms for a large 100×200 m map.

The global planner is optimistic in assuming the robot to be circular, with a diameter equal to the width of the robot. Also, it does not take into account the nonholonomic nature of the robot's motion. Instead, we rely on a local controller to produce feasible driving motions (Section 5).

One of the problems encountered in directing the robot toward a point goal is that the plans tend to constantly urge the robot toward the center of the map. This is not necessarily an efficient strategy because, for example, the robot will prefer to run near vegetation on the side of a path that does not point directly toward the goal. Instead, when the robot is far from the goal, we posit a relaxed *virtual goal line* that allows the robot to pursue more indirect paths to the goal (Figure 15). In a line goal, any point on the line is considered to be a goal, and the robot navigates to the nearest (lowest-cost path) position on the line. For example, in Figure 15 the robot is shown with the direction of travel straight ahead to its line goal, whereas with the goal point at the end it would want to move diagonally.

The line goal is easily implemented in the gradient planner by simply adding all points on the line as goal points. The navigation function then computes the lowest-cost path to any point on the line. The line goal is always placed about 60 m ahead of the robot, and its extent grows in the middle of the run and contracts as it gets nearer to the goal. In experiments, the robot is able to navigate more than 50 m off the center line to the goal and consequently find easily traversed paths that would have been difficult to find if it had headed directly to the goal (Figure 7).

5. CONTROL

Given the global cost information produced by the gradient planner, we must decide what local controls to apply to the robot to drive it toward the goal.

5.1. Trajectory Generation

We take an approach that is opposite to techniques such as DWA. Instead of searching the space of feasible *trajectories*, we search the space of feasible *controls*. As is the case with most differentially driven platforms, the LAGR robot is commanded by a pair $(\dot{x}, \dot{\theta})$ of desired translational and rotational velocities.[2] Thus we have a 2D space of possible commands to consider.

This space is bounded in each dimension by velocity limits that reflect the vehicle's capabilities. Because we are seeking *good*, as opposed to *optimal*,

[2]We could instead work in terms of left and right wheel velocities; the two velocity spaces are equivalent, being related by a simple geometric transformation.

control, we sample, rather than exhaustively search, this rectangular region of allowed velocities. We take a regular sampling (~25 in each dimension, ~625 total) and for each sample simulate the effect of applying those controls to the robot over a short time horizon (~2 s). The simulation predicts the robot's trajectory as a sequence of five-dimensional $(x, y, \theta, \dot{x}, \dot{\theta})$ states with a discrete-time approximation of the vehicle's dynamics.

Of significant importance in this simulation are the vehicle's acceleration limits. Although the LAGR robot can achieve a speed of 1.3 m/s, its low-level motor controller (which we cannot modify) follows a trapezoidal velocity profile that limits the translational acceleration to approximately 0.5 m/s^2 (we determined this value empirically). Thus more than 2 s may elapse between commanding and achieving a desired velocity. We found that the ability to accurately predict the LAGR robot's future state depends vitally on appropriate integration of these acceleration limits. We expect this to be the case for any vehicle with a similarly large ratio of maximum velocity to maximum acceleration.

The generated trajectories, projected into the (x, y) plane, are smooth, continuous 2D curves that, depending on the acceleration limits, may not be easily parameterizable. For the LAGR robot, the trajectories are generally not circular arcs (Figure 16).

5.2. Trajectory Evaluation

Each simulated trajectory t is evaluated by the following weighted cost:

$$C(t) = \alpha \mathrm{Obs} + \beta \mathrm{Gdist} + \gamma \mathrm{Pdist} + \delta \frac{1}{\dot{x}^2}, \qquad (1)$$

where Obs is the sum of grid cell costs through which the trajectory passes (taking account of the robot's actual footprint in the grid); Gdist and Pdist are the estimated shortest distances from the endpoint of the trajectory to the goal and the optimal path, respectively; and $\dot{x}$ is the translational component of the velocity command that produces the trajectory. We choose the trajectory for which the cost (1) is minimized, which leads our controller to prefer trajectories that (a) remain far from obstacles, (b) go toward the goal, (c) remain near the optimal path, and (d) drive fast. Trajectories that bring any part of the robot into collision with a lethal obstacle are discarded as illegal.

Note that we can compute $C(t)$ with minimal overhead: Obs is a simple summation over grid cell costs, Gdist and Pdist were already computed by the planner for all map cells, and $\dot{x}$ is a known constant for each trajectory.

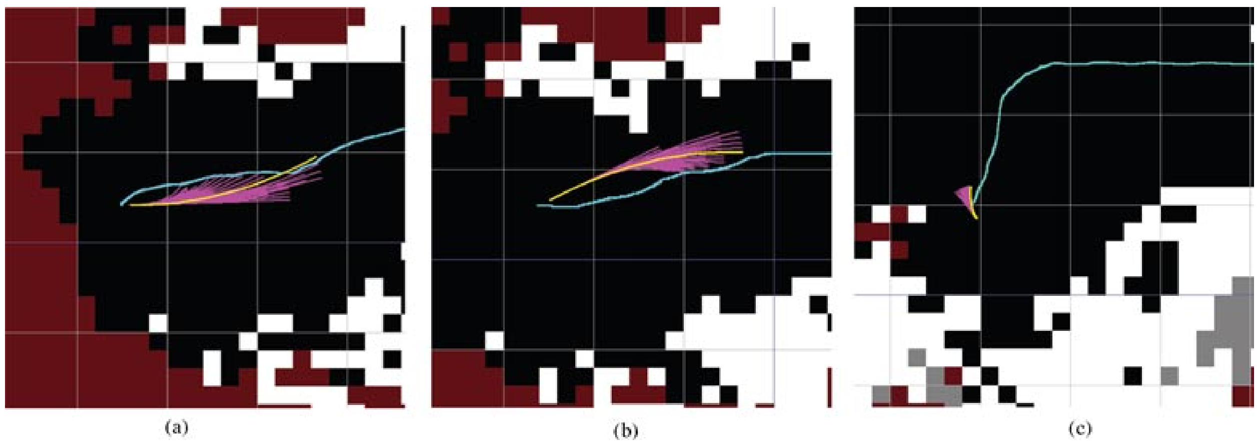


Figure 16. The controller generates trajectories by sampling feasible velocities and simulating their application over a short time horizon. Generated trajectories are purple, the chosen trajectory is yellow, the desired global path is cyan, and obstacles are white. As shown in (a) and (b), the trajectories are smooth but not easily parameterizable as they depend on the vehicle's current velocity and its acceleration limits. When forward motion is not possible, backward trajectories are considered (c): robot is facing down toward obstacles.

5.3. Supervisory Control

We could generate, evaluate, and compare all potential trajectories. However, given the kinematic design (driven wheels in front, passive casters behind) and sensor configuration (forward-facing cameras and forward-mounted bumper) of the LAGR robot, we found it useful to add supervisory logic to direct the order in which candidate velocities are simulated and evaluated.

All forward velocities ($\dot{x} > 0$) are tried first; if any legal forward trajectory is found, the best one is selected. If there are no legal forward velocities, then the controller tries in-place rotations ($\dot{x} = 0$), and then backward velocities ($\dot{x} < 0$). This preference ordering encourages the robot to make forward progress whenever possible and discourages driving backward (during which the robot is essentially blind). If no legal trajectory is found, the default behavior of the robot is to move slowly backward.

5.4. Slip Handling

Because the robot may have to traverse rough, steep terrain, it is necessary to detect and react to conditions in which the wheels slip or become stuck. We employ two mechanisms to handle these situations. In both cases, we are comparing the motion reported by the wheels to the motion estimated by VO, which is sufficiently accurate to be treated as ground truth (Section 3.2).

First, the controller continuously compensates for the slip in each wheel by reducing its maximum speed. Our approach is similar to automotive traction control. For each wheel, we monitor the slip ratio s, defined as (Angelova, Matthies, Helmick, Sibley, & Perona, 2006):

$$s = \frac{\omega r - v}{\omega r} \in [0, 1], \qquad (2)$$

where ω is the measured angular velocity of the wheel, r is the wheel radius, and v is the actual linear velocity of the wheel. We obtain ω directly from the wheel encoders. To compute v, we difference sequential VO poses to produce translational and rotational velocities for the vehicle and then use the vehicle geometry to distribute these velocities between the two wheels. When the slip ratio s for a wheel exceeds a minimum threshold (~0.25), we compensate by proportionally reducing the maximum allowable speed for that wheel, which produces better traction on most terrain. Importantly, the controller takes account of the current speed limits, ensuring that predicted trajectories will be achievable under these limits. The slip ratios and speed limits are recomputed at the frequency of VO pose estimation (~15 Hz).

Although continuous slip compensation improves performance, there are situations in which the robot can become truly stuck and require explicit escape mechanisms. The robot usually becomes stuck because of extremely slippery soil (e.g., sand) or ground clutter (e.g., fallen branches). We detect these conditions by looking for significant, time-extended disparities among the velocities that are commanded by the controller, reported by wheel odometry, and estimated by VO (we maintain a running window of each velocity). If a slip or stall is detected, or if the front bumper is triggered, the robot enters a stochastic finite state machine of preprogrammed escape maneuvers (e.g., drive forward, turn in place, drive backward). These maneuvers are executed blindly, on the assumption that the vision system failed to identify the terrain as dangerous and so is unlikely to yield good advice on how to escape it.

One indication of how well slip detection performed was on Test 18, about 2 years into the program. This was a difficult test around small sand dunes, and the robots would spin easily on the sand on small inclines. The slip detection code and escape maneuvers, coupled with VO for localization, allowed us to finish this challenging course, the only team to do so.

6. PERFORMANCE

For the LAGR program, the government testing group (LGT) ran monthly blind demos of the perception and control software developed by the teams and compared their performance to that of a baseline system. Teams were encouraged but not required to send code for each test; they could also send the same code on successive tests. In general the tests would change each month, to expose the teams to different environmental conditions.

There were two major checkpoints for all the teams, the first at the end of Phase I of the program after 1.5 years (Tests 12 and 13) and the second at the end of the program (3 years, Test 27). The goal was to beat the baseline system at the end of Phase I and to do better by a factor of 2 for the second. In this section we show results from all of these tests and additionally the penultimate Tests 25 and 26. At this point, our system was essentially complete, and Tests 25 and 26 presented interesting terrain challenges, whereas Test 27 was somewhat artificial and designed to isolate specific learning capabilities.

On each test, the robot was given four runs, and the best three were taken to give a combined score. The highest achievable score is 1.0, calculated by measuring the shortest possible path to the goal and measuring the time it took a skilled operator to manually drive the robot. There were also penalties for not getting to the goal within a cutoff time.

6.1. Phase I Tests

The tests at the end of Phase I were designed to test the overall ability of the system to handle typical outdoor terrain (make a consistent map, recover from slips, etc.), while focusing on perceptual skills and learning. For these tests, we had completed a basic VO system, the global planner, and the controller. All of the main stereo interpretation algorithms were also in place, including sightlines. However, our appearance-based learning was only a simple supervised color classifier, and there was no map reuse, because the VO system was not precise enough.

In Test 12 [Figure 17(a)], a dark mulch path formed the easiest way to the goal, and teams were invited to submit automatic supervised learning code that would be trained on a similar path from log files. The minimum times for a skilled operator were 69 s along the path (103 m) and 77 s through the maze (distance not measured).

Several teams managed to follow the path after training and generally had good times. We did not; the dark color of the path confused our color-based learner and led us to develop the combined color/texture model described in Section 2.3. We completed the maze course three times, with times of 106, 110, and 112 s. Our best time equaled that of the best team following the mulch path, and our average score of 0.73 was the highest of any team, three times the baseline score (which was also through the maze). Although the LGT expected the maze to be a significant problem for the teams (as it was for the baseline), our combination of consistent map making, fast global planning, and rollout controller combined to make the robot zip through the maze with no hesitation.

In Test 13, the objective was to stay along an old dirt and asphalt road for much of the course, until there was an opening in brush to the left of the road toward the goal [see Figure 17(b)]. The easiest way (most open route) to the goal was along the third route from the left, which required the robot to stray

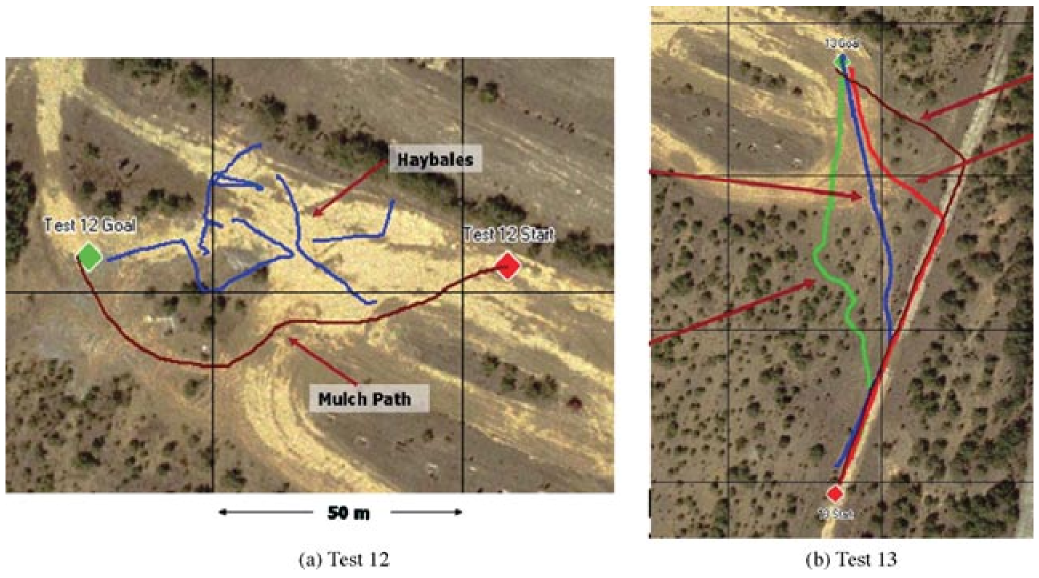


Figure 17. Aerial views of the Phase I tests. In (a), the haybale maze is drawn in blue, and the mulch path leading to the goal is in red. The tree copse leading directly to the goal was not traversable. In (b), four possible routes to the goal are drawn, with the third from the left being the easiest (154 m).

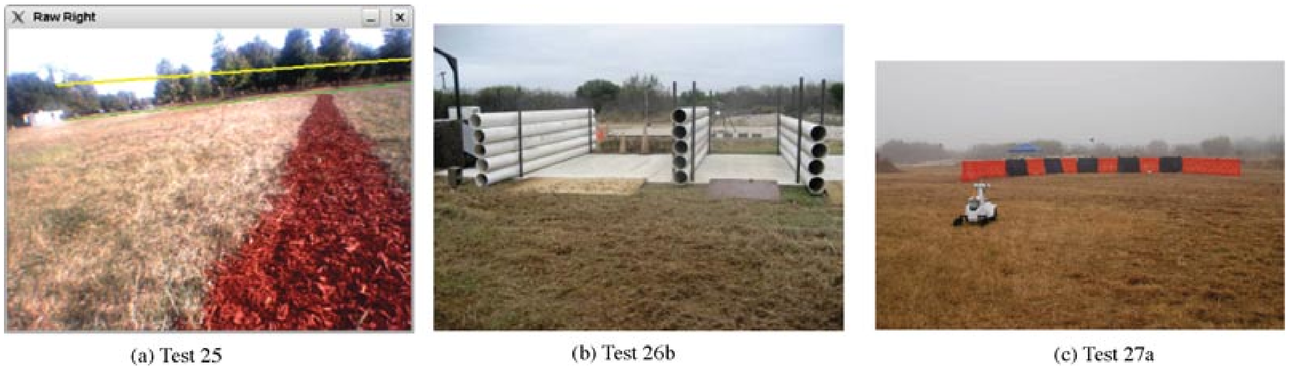


Figure 18. Views of three final tests. In (a), a robot's-eye view of the beginning of Test 25. The copse in the distance could be avoided on the right or left. The yellow line is the robot's horizon from a noisy INS, and the green line is the VO-stabilized horizon. In (b), a pipe corridor from Test 26b: note the blocked left corridor. In (c), Test 27a shows the Jersey barrier, with the goal immediately behind.

far from the direct route to goal, following the open road. The second route was actually the shortest, requiring a skilled operator just 90 s, and the third route took 95 s.

In all three scoring runs, we followed the third route, utilizing sight lines (Section 2.2) to find open space along the road and then the open space along the third route. The times for the three runs were 132, 132, and 148 s. All of our times were faster than the best time of any other team, and our overall score, 0.86, was just under 3× the score of the baseline. We expect that the better online color/texture path learning of Section 2.3 would have found the second route, and a shorter time, but we did not develop this technique until the end of the project.

6.2. End-of-Project Tests

The end-of-project tests were through different types of terrain and with different degrees of difficulty. For these tests, our full system was operational, including online color/texture learning of paths and map reuse. Here is a summary of the courses (Figure 18):

Test 25	83-m straight-line distance to the goal, through a copse of trees with a cul-de-sac and tall grass [Figure 18(a)]. Ideal behavior was to go around the copse, following a mulch path as in Test 12.
Test 26a	(93 m) Narrow paths through tall bushes, with several false turns that might lead more directly to the goal. Desired behavior was to avoid the false turns.
Test 26b	(106 m) A challenging course with man-made and natural obstacles, including a cul-de-sac of parked cars, stacked pipes, hay bales, and rock piles [Figure 18(b)]. The course to the goal was indirect and involved narrow passageways, and finding it was a challenge.
Test 27a	(34 m) A simple course on a grassy field with Jersey barriers stretched directly across the route [Figure 18(c)]. Ideal behavior would be to avoid the barrier without getting close.
Test 27b	(34 m) Similar to 27a, but using low hay bales for obstacles, with two gaps in the barrier containing tall grass. The object was to identify the tall grass and push through it directly to the goal.

The first four tests were designed to reward behavior that could avoid routes that were temptingly direct, but ultimately dead ends. There were two methods of doing this: long-range perception (>10 m) and map memorization and reuse. For Test 26a, the narrow routes through the bushes were easily detected by our online learning algorithms, and the path planner moved the robot quickly along the center of the path. On the first run, the robot turned twice to look briefly at side paths that could have been more direct but then turned back to the main route. Figure 19 shows the scores for this run. The Baseline score is 0.23, and SRI's score is 0.83, which is better by a factor of 3.6. In this test, because the long-range perception of paths worked well, the first run was very

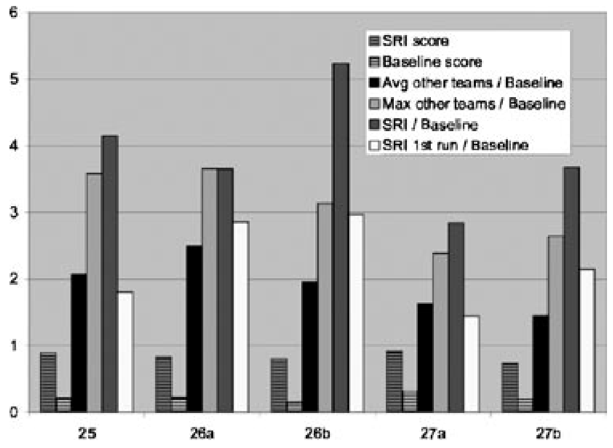


Figure 19. Summary of results from the last three LAGR tests. Raw scores are given for the Baseline software and the SRI system, where 1 is a perfect score (as fast as the robot can go). The other scores are presented as a factor over Baseline; the target performance for the project was 2 × Baseline.

good (2.9 × Baseline), and subsequent map reuse contributed only a modest amount, by not turning to examine the dead-end paths. In fact, our score could have been higher, but the fourth run failed because of a map registration error in the middle of the run, closing off the narrow path.

In the other three tests (25, 26b, and 27a), map reuse is the primary enabler of good performance: it improved by almost a factor of 2 from the first run. For example, in Test 25, after wandering through the copse and encountering the cul-de-sac and tall grass obstacles, the robot made its way to the goal. On the second run, the robot avoided the copse entirely, choosing a path around it as less costly.

Test 27b was a learning-by-example test. The robots were shown samples of the hay bales and tall grass. Operators would drive the robots into the hay bales and over the grass, to give the robot an idea of the traversability of each. Our online learning algorithms correctly picked out the grass as drivable, based primarily on its texture, because the color was similar to that of the hay bales. We also learned that hay bales were obstacles; however, we had set the suppression of obstacles by drivable objects a little too high, and the robot bumped the hay bales next to the grass area. After a few bumps, it drove through the grass and onto the goal. In subsequent runs, of course, map reuse allowed an optimal plan directly through the grass.

6.3. Analysis

There is no doubt that our system achieves both robustness and good performance on a wide variety of outdoor, unstructured terrain. Map building relies on VO to provide good localization, efficient real-time stereo and robust ground-plane analysis for obstacle detection, and sight lines to identify distant regions that are likely to be navigable. Online path learning helps in the very common situation of tracks through vegetation, or man-made dirt and asphalt roads. Together these techniques allow us to construct well-registered, precise maps that serve well during the first run to get the robot reliably to the goal. Even more importantly, on subsequent runs, the path planner is able to construct an optimal path to the goal from the start of the run.

Moving quickly is very important to achieving good performance, especially because many small obstacles such as branches could be traversed at speed but might hang up the robot if it was moving more slowly. As described in Section 5, the path planner and local controller combined to give the robot a very agile feeling. Our average speed was more than 1.1 m/s, even while exploring unknown terrain (top speed of the robot is 1.3 m/s).

The government team was very interested in creating scenarios to test the long-range perception of the robot. Unfortunately, the robot's vision sensors had very little resolution at distance. Depth information from stereo was very uncertain after about 7 m. Even using monocular information, very few pixels were available for long-range sensing. In Figure 18(a), a high-resolution camera with a longer focal length clearly shows routes around the copse of trees to the left. But looking through the robot cameras, there is very little to show that the copse of trees could be avoided to the left: perhaps there are a few more vertical pixels of brown-colored grass on that side. But this information is insufficient to reliably navigate from the robot's perspective, and teams that tried to do this would as often pick a bad way as a good one.

What we could reliably learn is the map structure from the first run. With this in hand, subsequent runs could be much more efficient. We had this technique working reliably only in the last tests (25–27), and it was difficult for the government team to react and set up tests that would allow long-range perception to do as well as map learning and reuse. It was also difficult for other teams to adopt our technique, because it required very good map registration, and a badly registered map is worse than no map at all. In Test 26a, the narrow paths (≈2 m wide) meant that even small registration errors could cause a prior map to close off the current path, which happened to us in the fourth run. Note that the map reuse was run open loop: after registering with an initial image at the beginning of the run, we relied on VO to keep the robot localized.

We compared our results with the published results of the other teams, both the average and the best for each test (Figure 19). In all these tests, we had the best score (or tied for the best). Typically we outperformed the average team by a factor of two. In the most difficult test, 26b, even our first-run score was almost as good as the best overall team score; map reuse enabled us to do even better. The controller, planner, and VO system were used in the best-in-class NIST system, and in fact NIST was our closest competition in two of the tests, including the difficult Test 26b. We also surpassed the target of $2\times$ baseline performance, achieving almost a $4\times$ improvement, better than the $3\times$ improvement of the Phase I tests. There is no doubt that map reuse was the primary enabler for this performance.

Whereas many teams concentrated on finding obstacles at a distance using color-based learning, we decided that the risk of using this technique in complicated environments was not worth the results. In some simple (and contrived) scenarios, such as Test 27a [Figure 27(c)], it could indeed help, but if it were used all the time, it would be as likely to lead to bad choices as good (dark green areas could be shadows instead of trees) and cause poor overall behavior. Our online path learning, by contrast, used both geometric and color/texture cues to reliably find good paths, with almost no false positives.

7. CONCLUSION

We have demonstrated a complete autonomous system for off-road navigation in unstructured environments, using stereo vision as the main sensor. The system is very robust—we can typically give it a goal position several hundred meters away and expect it to get there. It is also one of the first to demonstrate the practical use of VO as the primary method of registration, with extremely good results. The precision of VO is such that maps can be reused on subsequent runs, doubling the performance of the system.

To be sure, there are hazards that are not dealt with by the methods discussed in this paper: water and ditches are two robot killers. We also were restricted to running open loop in reusing maps; we would like to use visual landmarks to reregister the position of the robot in the map, but this work was not ready at the time of the last tests.

REFERENCES

Agrawal, M., & Konolige, K. (2006). Real-time localization in outdoor environments using stereo vision and inexpensive GPS. In Proceedings of the International Conference of Pattern Recognition (ICPR) (pp. 1063–1068). Washington, DC: IEEE Computer Society.

Agrawal, M., & Konolige, K. (2007). Rough terrain visual odometry. In Proceedings of the International Conference on Advanced Robotics (ICAR), Jeju Island, South Korea.

Agrawal, M., Konolige, K., & Blas, M. R. (2008). CenSurE: Center surround extremas for realtime feature detection and matching. In Proceedings of the European Conference on Computer Vision (ECCV), Marseille, France. Lecture Notes in Computer Science 5305 (pp. 102–115). Springer.

Angelova, A., Matthies, L., Helmick, D., & Perona, P. (2007). Learning and prediction of slip from visual information. Journal of Field Robotics, 24(3), 205–231.

Angelova, A., Matthies, L., Helmick, D., Sibley, G., & Perona, P. (2006). Learning to predict slip for ground robots. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Orlando, FL (pp. 3324–3331).

Bay, H., Tuytelaars, T., & Gool, L. V. (2006). SURF: Speeded up robust features. In Proceedings of the European Conference on Computer Vision (ECCV), Graz, Austria (pp. 404–417).

Bellman, R. (1957). Dynamic programming. Princeton, NJ: Princeton University Press.

Bellutta, P., Manduchi, R., Matthies, L., Owens, K., & Rankin, A. (2000). Terrain perception for DEMO III. In Proceedings of the IEEE Intelligent Vehicles Symposium, Dearborn, MI (pp. 326–331). IEEE Computer Society.

Blas, M. R., Agrawal, M., Konolige, K., & Sundaresan, A. (2008). Fast color/texture segmentation for outdoor robots. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France (IROS) (pp. 4078–4085).

Engels, C., Stewnius, H., & Nister, D. (2006). Bundle adjustment rules. In Proceedings of the Conference on Photogrammatic Computer Vision, Bom, Germany (pp. 266–271).

Fischler, M., & Bolles, R. (1981). Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. Communications of the ACM, 24, 381–395.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics and Automation Magazine, 4(1), 23–33.

Guivant, J., Nebot, E., & Baiker, S. (2000). High accuracy navigation using laser range sensors in outdoor applications. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA (pp. 3817–3822).
Gutmann, J. S., & Konolige, K. (1999). Incremental mapping of large cyclic environments. In Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Monterey, CA (pp. 318–325).
Happold, M., Ollis, M., & Johnson, N. (2006). Enhancing supervised terrain classification with predictive unsupervised learning. In Proceedings of Robotics: Science and Systems, Philadelphia, PA (RSS) (pp. 901–914).
Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France (IROS) (pp. 3946–3952).
Howard, T., Green, C., & Kelly, A. (2007). State space sampling of feasible motions for high performance mobile robot navigation in highly constrained environments. In Proceedings of the International Conference on Field and Service Robotics, Chamonix, France.
Iagnemma, K., Genot, F., & Dubowsky, S. (1999). Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Detroit, MI (pp. 2286–2291).
Johnson, A. E., Goldberg, S. B., Cheng, Y., & Matthies, L. H. (2008). Robust and efficient stereo feature tracking for visual odometry. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Pasadena, CA (pp. 39–46).
Kelly, A. (1994). A feedforward control approach to the local navigation problem for autonomous vehicles (Tech. Rep. CMU-RI-TR-94-17). Pittsburgh, PA: Robotics Institute, Carnegie Mellon University.
Kimmel, R., & Sethian, J. A. (1998). Computing geodesic paths on manifolds. Proceedings of the National Academy of Science, 95, 8431–8435.
Konolige, K. (1997). Small vision systems: Hardware and implementation. In Proceedings of the International Symposium on Robotics Research, Nagoya, Japan (pp. 111–116).
Konolige, K. (2000). A gradient method for realtime robot control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo, Japan (pp. 639–646).
Konolige, K., & Agrawal, M. (2008). FrameSLAM: From bundle adjustment to realtime visual mappping. IEEE Transactions on Robotics, 24(5), 1066–1077.
Konolige, K., Agrawal, M., & Solà, J. (2007). Large scale visual odometry for rough terrain. In Proceedings of the International Symposium on Robotics Research, Hiroshima, Japan.
Konolige, K., & Beymer, D. (2007). SVS reference manual (Tech. Rep.) SRI International. http://www.videredesign.com/docs/smallv4.4d.pdf; accessed July 2007.
Latombe, J.-C. (1991). Robot motion planning. Norwell, MA: Kluwer Academic Publishers.
LaValle, S. (2006). Planning algorithms. New York: Cambridge University Press.
Leonard, J. J., & Newman, P. (2003). Consistent, convergent, & constant-time SLAM. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico (pp. 1143–1150).
Leung, T., & Malik, J. (2001). Representing and recognizing the visual appearance of materials using three-dimensional textons. International Journal of Computer Vision 43(1), 29–44.
Liapis, S., Sifakis, E., & Tziritas, G. (2004). Color and/or texture segmentation using deterministic relaxation and fast marching algorithms. Journal of Visual Communication and Image Representation, 15, 1–26.
Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91–110.
Mäenpää, T., & Pietikäinen, M. (2005). Texture analysis with local binary patterns. In C. Chen & P. Wang (Eds.) Handbook of pattern recognition and computer vision, 3rd ed. (pp. 197–216). Singapore: World Scientific.
Maimone, M., Cheng, Y., & Matthies, L. (2007). Two years of visual odometry on the Mars exploration rovers. Journal of Field Robotics, 24(3), 169–186.
Martin, D., Fowlkes, C., & Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(5), 530–549.
Matthies, L., Maimone, M., Johnson, A., Cheng, Y., Willson, R., Villalpando, C., Goldberg, S., Huertas, A., Stein, A., & Angelova, A. (2007). Computer vision on Mars. International Journal of Computer Vision, 75(1), 67–92.
Montemerlo, M., & Thrun, S. (2004). Large-scale robotic 3-D mapping of urban structures. In Proceedings of the International Symposium on Experimental Robotics (ISER), Singapore (pp. 141–150).
Moravec, H., & Elfes, A. (1985). High resolution maps from wide angle sonar. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (pp. 116–121).
Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., & Sayd, P. (2006). Real time localization and 3D reconstruction. In Proceedings of Computer Vision and Pattern Recognition Conference (CVPR), New York (vol. 1, pp. 363–370).
Nister, D., Naroditsky, O., & Bergen, J. (2006). Visual odometry for ground vehicle applications. Journal of Field Robotics, 23(1), 3–20.
Philippsen, R., & Siegwart, R. (2005). An interpolated dynamic navigation function. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Barcelona, Spain (pp. 3782–3789).
Rankin, A., Huertas, A., & Matthies, L. (2005). Evaluation of stereo vision obstacle detection algorithms for off-road autonomous navigation. In Proceedings of the AUVSI Symposium on Unmanned Systems.
Spero, D. J., & Jarvis, R.A. (2002). 3D vision for large-scale outdoor environments. In Proceedings of the Australasian Conference on Robotics and Automation (ACRA), Auckland, New Zealand (pp. 228–234).

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Diego, CA (vol. 4, pp. 3310–3317).

Sunderhauf, N., Konolige, K., Lacroix, S., & Protzel, P. (2005). Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle. In M. Schanz, R., Lafrenz, P. L., & V. Avrutin (Eds.), Tagungsband autonome mobile systeme (pp. 157–163). Springer Verlag.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., & Mahoney, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics, 23(9), 661–692.

Triggs, B., McLauchlan, P. F., Hartley, R. I., & Fitzgibbon, A. W. (2000). Bundle adjustment—A modern synthesis. In B. Triggs, A. Zisserman, & R. Szeliski, (Eds.), Vision algorithms: Theory and practice, LNCS (pp. 298–375). Springer Verlag.

Varma, M., & Zisserman, A. (2003). Texture classification: Are filter banks necessary? In Proceedings of Computer Vision and Pattern Recognition Conference (CVPR), Madison, WI (pp. 691–696).

Viola, P., & Jones, M. (2004). Robust real-time face detection. International Journal of Computer Vision, 57(2), 137–154.