Mapping, Navigation, and Learning for Off-Road Traversal

Kurt Konolige
Willow Garage
Menlo Park, CA 94025
konolige@willowgarage.com

Motilal Agrawal
SRI International
Menlo Park, CA 94025
agrawal@ai.sri.com

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

Robert C. Bolles
SRI International
Menlo Park, CA 94025
bolles@ai.sri.com

Brian Gerkey
Willow Garage
Menlo Park, CA 94025
gerkey@willowgarage.com

Joan Solà
LAAS
Toulouse, France
joan.scat@gmail.com

Aravind Sundaresan
SRI International
Menlo Park, CA 94025
aravind@ai.sri.com

Abstract

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. At the end of three years, the system we developed outperformed all 9 other teams in final blind tests over previously-unseen terrain. In this paper we describe the system, as well as the two learning techniques that led to this success: *online path learning* and *map reuse*.

1 Introduction

The DARPA LAGR project began in Spring 2005 with the ambitious goal of achieving vision-only autonomous traversal of rough terrain. Further, the participating teams were to be tested "blind" – sending in code to be run on a robot at a remote, unseen sight. 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 2x performance over a Baseline system already developed at the National Engineering Research Consortium (NERC) in Pittsburgh. All participant teams used the same robotic hardware provided by NERC (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 rangefinders (Montemerlo and Thrun, 2004; Bellutta et al., 2000; Guivant et al., 2000), 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 7m or so. Even where good stereo information is available, it is often impossible to judge traversability on the basis of 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 cul-de-sacs, dead ends, or paths that initially led towards 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 or not.

Throughout the project life, 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, we converged on the following basic capabilities, which constitute our novel contributions to the problem of autonomous navigation with vision.

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 state-of-the-art in appearance-based segmentation, and are the key to our online path-finding method. This method reliably finds 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, it cannot make good plans to get to its goal. After noticing navigation failures from the very noisy registration provided by GPS, we decided to give high priority to precise registration of local map information into a global map. Here, we developed realtime visual odometry (VO) methods that are more precise than existing ones, while still being computable at frame rates. To our

(a) LAGR Robot (b) LAGR Robot

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.

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* 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 Dynamic Window Approach (DWA) (Fox et al., 1997) for local control. These proved inadequate for realtime 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.

In the end, the teams were tested in a series of courses (Tests 25–27). Over these tests, we averaged about 4x the score of Baseline, the best of any team. In every test, our score beat or tied the best other team; and in the aggregate, we score almost twice as high as the best other team.

1.1 System overview

This work was conducted as part of the DARPA Learning Applied to Ground Robotics (LAGR) project. We were provided with two robots (see Figure 1, each with two stereo devices encompassing a 110 degree field of view, with a baseline of 12 cm. The robots are near-sighted: depth information degrades rapidly after 6m. There is also an inertial unit (IMU) with angular drift of several degrees per minute, and a WAAS-enabled GPS. There are 4 Pentium-M 2 GHz computers, one for each stereo device, one for planning and map-making, and one for control of the robot and integration of GPS and IMU readings. In our setup, each stereo computer performs local map making and visual odometry, and sends registered local maps to the planner, where they are integrated into a global map. The planner is responsible for global planning and reactive control, sending commands to the controller.

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 visual odometry and registration in detail, and show how consistent global maps are created. The next section discusses the global planner and local controller. Finally, we present performance results for several tests in Spring 2006.

2 Related work

There has been an explosion of work in mapping and localization (SLAM), most of it concentrating on indoor environments (Gutmann and Konolige, 1999; Leonard and Newman, 2003). Much of the recent research on outdoor navigation has been driven by DARPA projects on mobile vehicles

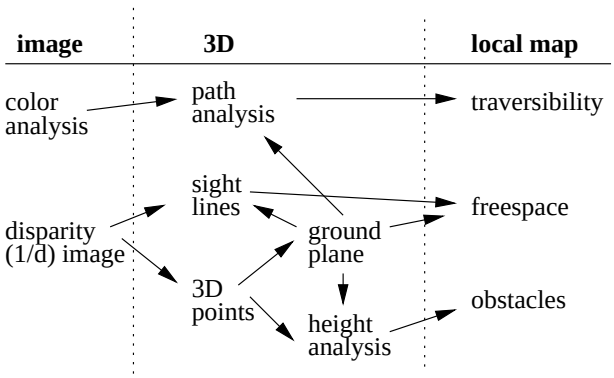
(Bellutta et al., 2000). The sensor of choice is a laser rangefinder, augmented with monocular or stereo vision. In much of this work, high-accuracy GPS is used to register sensor scans; exceptions are (Guivant et al., 2000; Montemerlo and Thrun, 2004). In contrast, we forego laser rangefinders, and explicitly use image-based registration to build accurate maps. Other approaches to mapping with vision are (Rankin et al., 2005; Spero and Jarvis, 2002), although they are not oriented towards realtime implementations. Obstacle detection using stereo has also received some attention (Rankin et al., 2005).

Visual odometry 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 visual odometry (Nister et al., 2004; Olson et al., 2000). Our visual odometry system (Konolige et al., 2007; Agrawal and Konolige, 2006; Agrawal and Konolige, 2007) is most similar to the recent work of Mouragnon et al. (Mouragnon et al., 2006) and Sunderhauf et al. (Sunderhauf et al., 2005). The main difference is the precision we obtain by the introduction of a new, more stable feature, and the integration of an 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 is the only system known to the authors that has been in regular use in demonstrations for over two years.

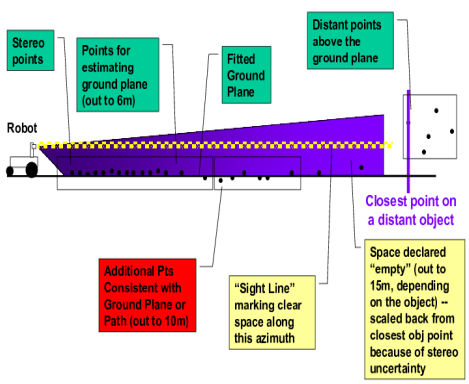
Our segmentation algorithm uses a compact descriptor to represent color and texture. In a seminal paper, Leung and Malik (Leung and Malik, 2001) showed that many textures could be represented and re-created using a small number of basis vectors extracted from the local descriptors; they called the basis vectors *textons*. While Leung and Malik used a filter bank, later Varma and Zisserman (Varma and Zisserman, 2003) 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 and 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), 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 et al. (Howard et al., 2007) present a more general approach to constraining the search for controls by first sampling directly



(a) Processing Diagram



(b) Stereo Points

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

in the vehicle's state space.

3 Local map construction

The object of the local map algorithms is to determine, from the visual information, which areas are freespace 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 freespace to more distant points. Finally, from color and path analysis, coupled with the ground plane, we determine paths and traversability of the terrain.

3.1 Stereo analysis and ground plane extraction

We use a fast stereo algorithm (Konolige, 1997) to compute a disparity image at 512x384 resolution (Figure 3, left). In typical outdoor scenes, it is 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 $[u, v, d]$ corresponds to a 3D point in the robot's frame.

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 towards finding freespace, especially in areas that are further 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

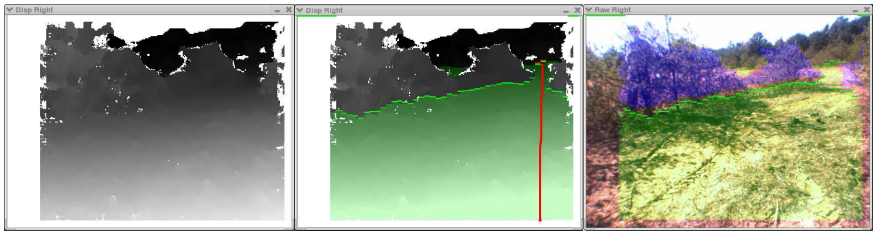


Figure 3: Left: disparity image from the left view of the robot in Figure 1. Closer pixels are lighter. Middle: extracted ground plane, in green overlay. Limit of ground plane is shown by green bar; sight line has a red bar. Right: Ground plane overlayed on original image, in green. Obstacles are indicated in purple.

detect obstacles using local variation in height [ref], using a ground plane simplifies processing and yields more stable results. To extract a ground plane, we use a RANSAC technique (Fischler and Bolles, 1981), choosing sets of 3 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 only looks out to 6m around the robot. A more sophisticated analysis would break the ground plane into several segments or model more complex shapes.

3.2 Sight lines

Although we cannot precisely locate obstacles past 6-8m, we can determine if there is freespace, using the following observation. Consider the interpreted image of Figure 3, middle. There is a path the 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 freespace up to this estimate. 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 line in Figure 3 middle). Note that the example sight line follows the obvious path out of the bushes.

3.3 Learning color and texture models

Our learning algorithm for color and texture models consists of two stages. In the first stage, we cluster color and texture vectors over small local neighborhoods to find a small set of basis vectors (textons (Leung and Malik, 2001)) that characterize different scene textures. In the second stage, we cluster histograms of these textons over larger areas to find more coherent regions with the same mixture of textons using k-means as our clustering algorithm. We use the CIE*LAB colorspace to represent colors because it is more perceptually uniform. Texture information is incorporated by taking the difference between a center pixel intensity and surrounding pixel intensities in a local 3x3 neighborhood. For the second stage, histograms can be constructed efficiently (irrespective of window size) using integral images (Viola and Jones, 2001).

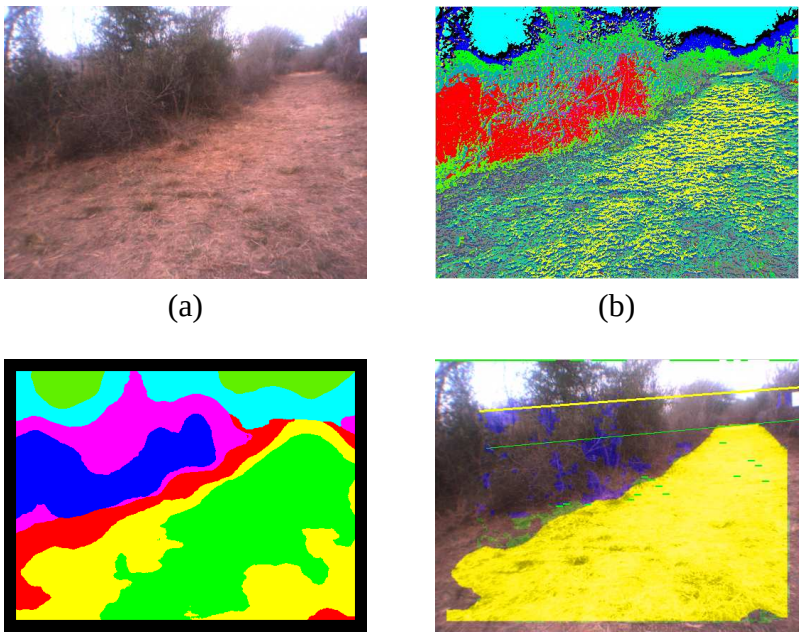


Figure 4: Various steps of our segmentation algorithm on a typical outdoor image. (a) The image from one of the stereo cameras. (b) Each pixel assigned to a texton. (c) Each histogram of textons gets assigned to a histogram profile. In this particular example, the path is composed of two segments. (d) A path is recognized (in yellow)

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 tele-operation 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 over-segmentation in the image (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 composed of grouped segments such as width, length and spatial continuity in order to verify if it geometrically resembles a path. Once a path is identified, the robot learns the texton histograms of the component segments as a model for the path.

For classification, the different clusters of the segmented image are compared to the learnt model of the path using Euclidean distance on the cluster histograms. Clusters that are within a certain threshold are identified as potential paths. A final geometrical analysis makes sure that these potential path regions have the right geometry.

The learning process runs at 1Hz for training on a single image and is typically performed at the beginning (although it could be performed at regular intervals to update the path model). Classification based on the learnt model runs at around 5Hz. Figure 4 shows the various steps of our algorithm on one of our test runs. The path between bushes is identified in yellow in Figure 4(d). For more details, please consult (Blas et al., 2008).

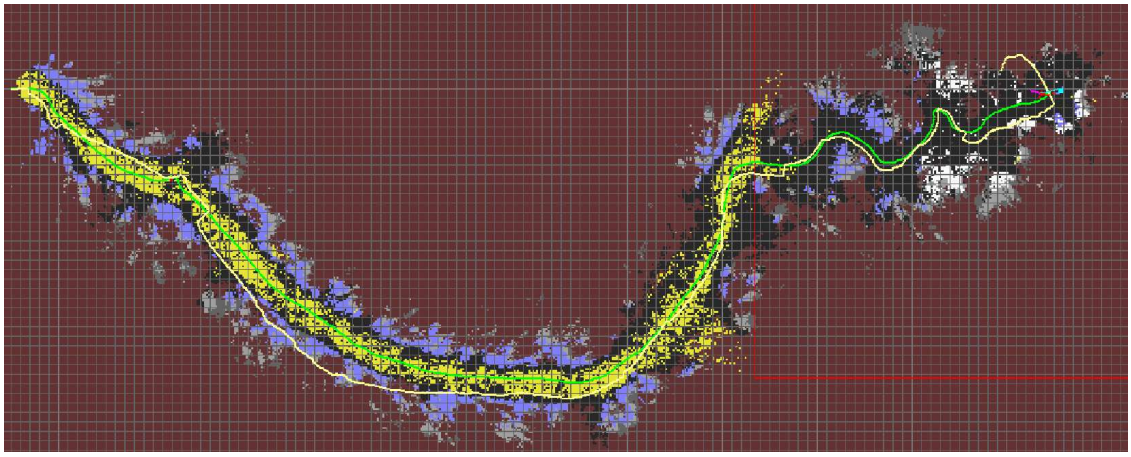


Figure 5: Reconstruction on a 130m autonomous run. Yellow is recognized path, black is freespace, and purple, gray and white are obstacles.

3.4 Results of local map construction

The combined visual processing results in local maps that represent traversability with a high degree of fidelity. Figure 5 shows the results of an autonomous run of about 130m, over a span of 150 seconds. We used offline learning of mulch paths on a test site, 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 freespace. 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 visual odometry (see Section 4.2), 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 visual odometry is that wheel slips and stalls are easily detected, with no false positives (Section 6.4). 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.

4 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.

4.1 Map representation

For indoor work, a standard map representation is a 2D *occupancy grid* (Moravec and Elfes, 1985), which gives the probability of each cell in the map being occupied by an obstacle. Alternatives for outdoor environments include 2.5D elevation maps and full 3D voxel maps (Iagnemma et al., 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 20cm x 20cm cells to represent the global map. Each cell has a probability of the belonging to the four categories derived from visual analysis (Secion 3): obstacle, ground plane freespace, sight line freespace, and path freespace. Note that these categories are not mutually exclusive, since, for example, a cell under an overhanging branch could have both path and obstacle properties. We are interested in converting these probabilities 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 non-exclusive 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 5 shows a color-coded version of computed costs.

4.2 Registration and visual odometry

The LAGR robot is equipped with a GPS that is accurate to within 3 to 10 meters 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 6. Here the robot goes through two loops of 10m diameter. There is a long linear feature (a low wall) that is seen as an obstacle at the beginning and end of the loops. Using the filtered GPS pose, the position of the wall shifts almost 2m during the run, and obstacles cover the robot's previous tracks.

Our solution to the registration problem is to use *visual odometry* (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 visual odometry solution that functions well under these conditions. We briefly describe it here; for more details consult (Agrawal and Konolige, 2006; Konolige et al., 2007).

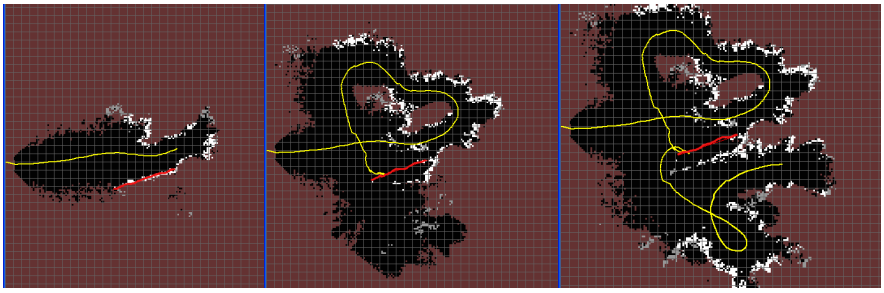


Figure 6: Three stages during a run using GPS filtered pose. Obstacle points are shown in white, freespace in black, and the yellow line is the robot's path. The linear feature is marked by hand in red in all three maps, in its initial pose.

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 and Bolles, 1981). Several thousand relative pose hypotheses are generated by randomly selecting three matched non-collinear 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, since 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 et al., 2006; Triggs et al., 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 litle computational overhead. A feature that is long lived, that is, 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 a lot of 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 et al., 2006) work well but are not suitable for a real time system. We have developed a novel feature (named CenSurE) (Agrawal et al., 2008) that has improved stability and is inexpensive to compute. While the basic idea of CenSurE features is similar to that of SIFT, the implementation is extremely efficient, comparable to Harris. Details of our CenSurE feature detector are described in (Agrawal et al., 2008). Figure 7 shows the CenSurE features tracked over several frames.

The IMU and the wheel encoders are used to fill in the relative poses when visual odometry fails.

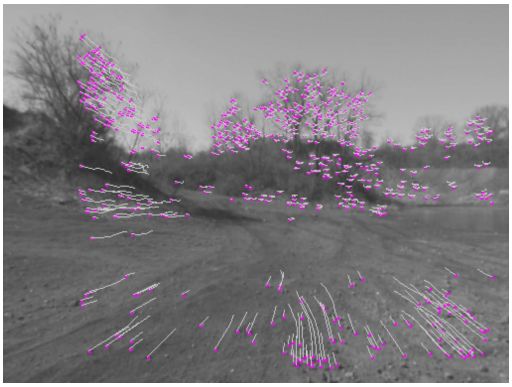


Figure 7: CenSurE features tracked over several frames.

This happens due to sudden lighting changes, fast turns of the robot or lack of good features in the scene(e.g. blank wall).

4.3 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 records tilt and roll based on gravity normal, calculated from the three accelerometers. This measurement is corrupted by robot motion, and is moderately noisy.
2. GPS Yaw – the IMU yaw data is very bad, and cannot be used for filtering (for example, over the 150 m run, it can be off by 60 degrees). 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, since it is globally corrected; but for LAGR test courses it has enough noise that this is not a concern.

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 travelling 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) towards global consistency, while maintaining local consistency.

The quality of the registration from filtered VO, shown in Figure 8, can be compared to the filtered GPS of Figure 6. The low wall, which moved almost 2m over the short loops when using GPS, is much more consistent when VO is employed. And in cases where GPS is blocked or degraded, such as under heavy tree cover in Figure 5, 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 5, where the robot was hung up and the wheels were slipping, and wheel

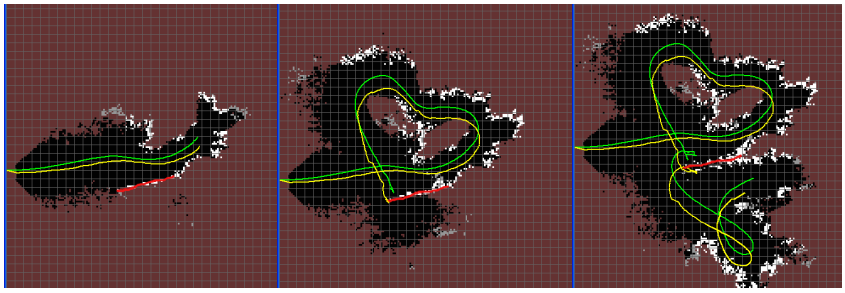


Figure 8: VO in the same sequence as Figure 6. GPS filtered path in yellow, VO filtered path is in green.

odometry produced a large error.

4.4 Results of visual odometry

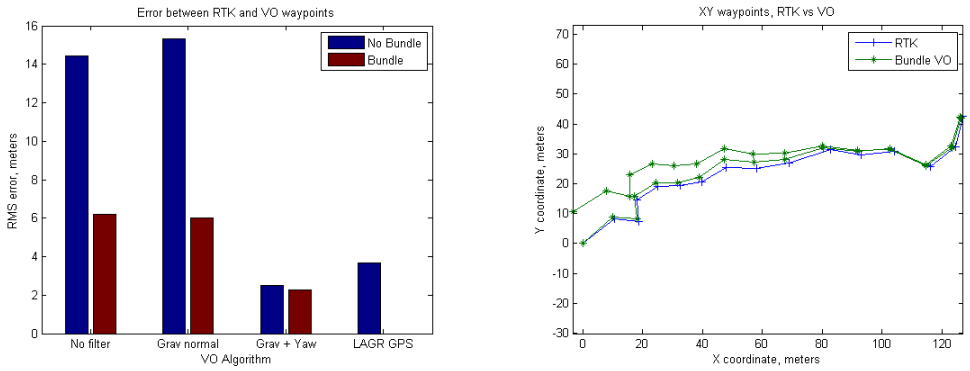
In Test 17, the testing team surveyed a course using an accurate RTK GPS receiver. The 'Canopy Course' was under tree cover, but the RTK GPS and the LAGR robot GPS functioned well. Sixteen waypoints were surveyed, all of which were within 10 cm error according to the RTK readout (one waypoint was deemed inaccurate and not included). The total length of the course was about 150 meters. Subsequently, the LAGR robot was joysticked over the course, stopping at the surveyed points. The robot was run forward over the course, and then turned around and sent backwards 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 9 shows two images acquired by the robot. The left image showa a good scene in the shadow of the trees, and the right image shows a poor image where the sun washes 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.

Figure 9: Images from the Canopy dataset.

Since the initial heading of the robot is unknown, we used an alignment strategy that assumes 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 10 (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 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 10(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 10m 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 dataset.



(a) (b)

Figure 10: Results of VO on the Canopy dataset. (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.

We present results of VO with CenSurE features on two other large outdoor datasets collected with a larger robot. These datasets have frame-registered ground truth from RTK GPS, which is accurate to several cm in XY and 10 cm in Z. For these datasets, the camera FOV is 35 deg, the baseline is 50 cm, and the frame rate is 10 Hz (512x384), so there is often large image motion. We took datasets from Little Bit (9 km trajectory, 47K frames) in Pennsylvania, and Ft Carson (4 km, 20K frames) in Colorado, to get variety in imagery. The Ft Carson dataset 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 datasets, 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 1, the effects of bundle adjustment and IMU filtering are compared.

In both datasets, IMU filtering plays the largest role in bringing down error rates. This isn't surprising, since 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/hr of drift. Noisier IMU

Table 1: Trajectory error statistics, in meters and percent of trajectory

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

yaw data would lead to higher XY errors.

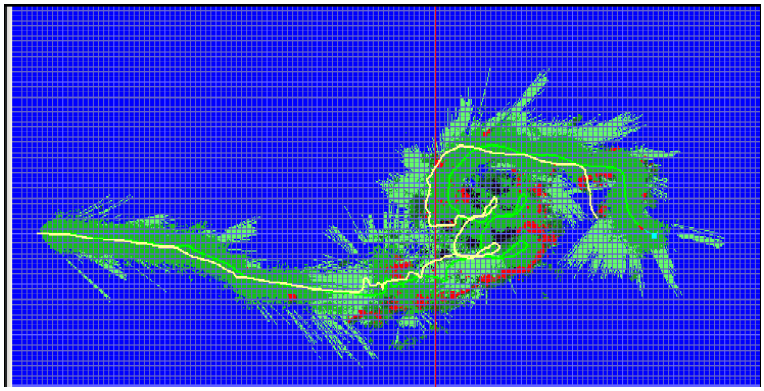
The secondary effect is from SBA. With or without IMU filtering, SBA can lower error rates by half or more, especially in the Ft. Carson dataset, where the matching is less certain.

4.5 Map Reuse

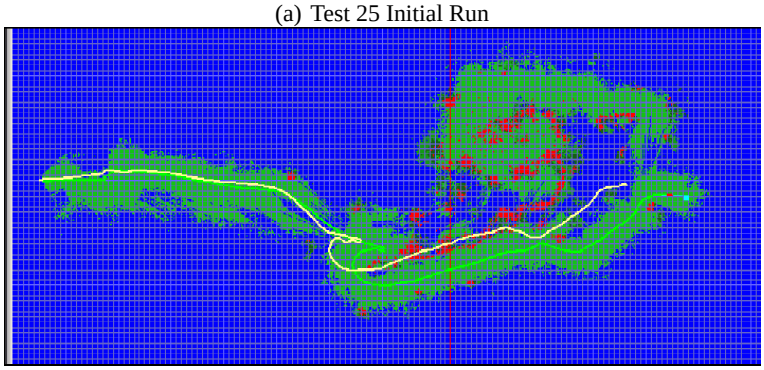
VO and IMU/GPS filtering enable us to construct consistent maps on a single run. These maps are useful for getting out of traps and cul-de-sacs in the environment, which occurred quite frequently. In fact, the testing team was interested in long-range sensing capabilities, and would use natural or constructed 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 7 and Figure 14(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 11 and Figure 14(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, then decided to go around. The planner had a finite horizon of about 40m, and only recognized the blockage at that 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 re-registration or localization of the robot within the map. Since VO performance is generally within 1% over 100m, 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 datasets (Konolige and Agrawal, 2008); but unfortunately this research was done



(a) Test 25 Initial Run



(b) Test 25 Second Run

Figure 11: Map reuse during Test 25. The global map in (a) shows the first run: dark green is freespace, light green is sightlines, red are obstacles. The robot path estimated from VO is the green line; the yellow line is the (noisy) GPS. Starting position of the robot is the left side of the screen; goal is on the right. Note the many extended concave obstacles and cul-de-sacs. Image (b) shows the robot's trajectory for the second run, bypassing the cul-de-sac obstacles and heading around to the right.

too late to incorporate into the LAGR system.

5 Planning

The LAGR robot was provided with a "baseline" system that used implementations of D* (Stentz, 1994) for global planning and Dynamic Window Approach (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 (100m x 200m, 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 re-implemented a gradient planner (Konolige, 2000; Philippsen and 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 and 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. Since 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 3 or 4 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 2-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., 80m x 80m) in several tens of milliseconds.
- Rapid switching of global paths is avoided by including hysteresis - lowering the cost along the path. There is a tradeoff 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, since the robot is continuously moving towards the goal and encountering new areas. On longer runs, up to 200m, we use an 80m x 80m area; the global planner runs in about 30 ms in this region. Unless there is a large cul-de-sac, longer than 80m, this area is sufficient to maneuver the robot tactically around

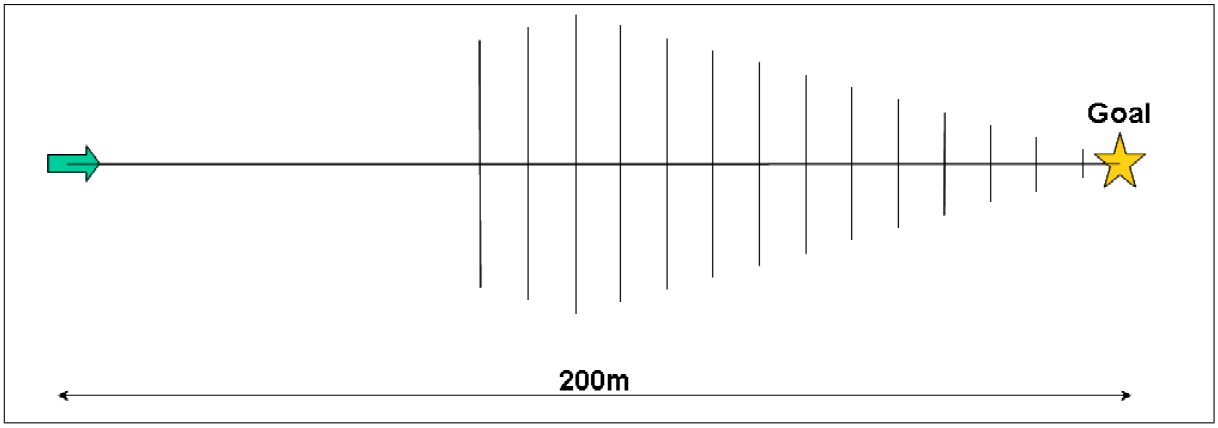


Figure 12: *Line goals for a robot in a 200m environment. The line goal is placed 60m ahead of the robot, and its extent varies with the distance to the goal.*

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 100m x 200m 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 6).

5.1 Line goals

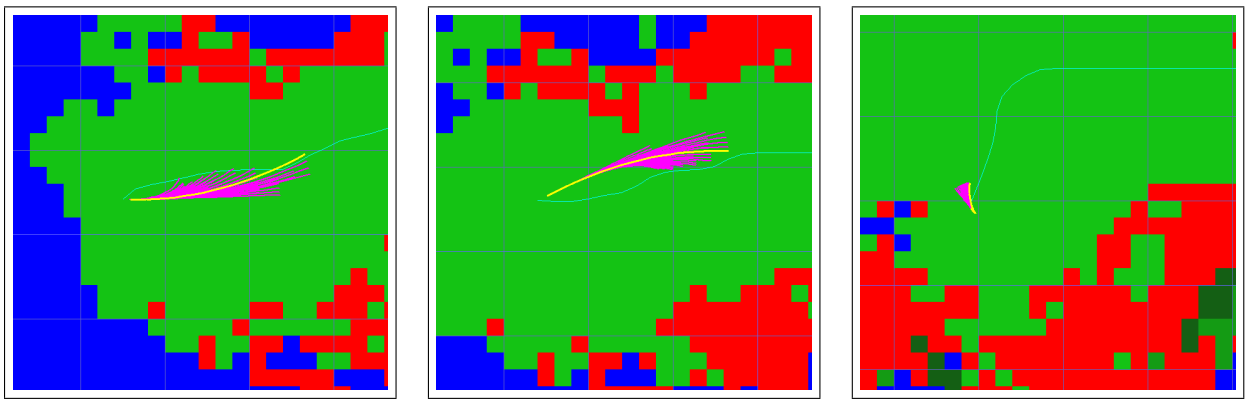
One of the problems encountered in directing the robot towards a point goal is that the plans tend to constantly urge the robot towards the center of the map. This is not necessarily an efficient strategy, because, for example, the robot will prefer to run near vegetation on the side of a path that does not point directly towards 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 (Fig. 12). In experiments, the robot is able to navigate more than 50m 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 (Fig. 5).

6 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.

6.1 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 transla-



(a) (b) (c)

Figure 13: *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 red. 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).*

tional and rotational velocities.[1] 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*, control, we sample, rather than exhaustively search, this rectangular region of allowed velocities. We take a regular sampling ($\sim$25 in each dimension, $\sim$625 total), and for each sample simulate the effect of applying those controls to the robot over a short time horizon ($\sim$2s). The simulation predicts the robot's trajectory as a sequence of 5-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. While 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 seconds 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 2-dimensional curves that, depending on the acceleration limits, may not be easily parameterizable. For the LAGR robot, the trajectories are generally not circular arcs (Fig. 13).

6.2 Trajectory evaluation

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

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

[1] 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.

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.

6.3 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.

6.4 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 visual odometry (VO), which is sufficiently accurate to be treated as ground truth (Section 4.2).

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 et al., 2006):

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

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, 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 ($\sim$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 ($\sim$15Hz).

While 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.

7 Performance

For the LAGR program, the government testing group ran monthly blind demos of the perception and control software developed by the teams, and compared their performance to a baseline system (see XXX in this issue). The target for the last series of tests at the end of the 3-year program was to do better than 2x the baseline performance. We show here tests 25 through 27, the last three, because our system was essentially complete at this time. On each test, the robot was given 4 runs, and the best 3 were taken to give a combined score. The highest achievable score is 1.0, calculated by measuring the shortest path to the goal, and assuming the robot could move at maximum speed (1.3 m/s) over this path. There are also penalties for not getting to the goal within a cutoff time.

7.1 End-of-project Tests

The tests themselves were through different types of terrain, and with different degrees of difficulty. Here is a summary of the courses.

25 83m straight-line distance to the goal, through a copse of trees with a cul-de-sac and tall grass (Figure 14(a)). Ideal behavior was to go around the copse.

26a (93m) 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.

26b (106m) 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 14(b)). The course to the goal was indirect and involved narrow passageways, and finding it was a challenge.

27a (34m) A simple course on a grassy field with jersey barriers stretched directly across the route (Figure 14(c)). Ideal behavior would be to avoid the barrier without getting close.

27b (34m) 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.

(a) Test 25 (b) Test 26b (c) Test 27a

Figure 14: 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, while 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.

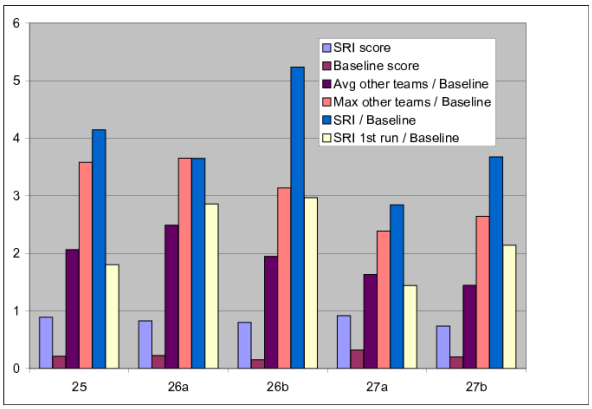


Figure 15: Summary of results from the last 3 LAGR tests. Raw scores are given for the Baseline software and the SRI system, where 1 is a perfect score (as fast as the robot can go). The other scores are presented as a factor over Baseline; the target performance for the project was 2x Baseline.

The first four tests were designed to reward behavior that could avoid paths that were temptingly direct, but ultimately dead-ends. There were two methods of doing this – long-range perception (>10m), and map memorization and re-use. 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 15 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, since the long-range perception of paths worked well, the first run was very good (2.9x Baseline), and subsequent map re-use only contributed 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, 27a), map re-use 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 driveable, based on primarily on its texture, since the color was similar to the hay bales. We also learned that hay bales were obstacles; however, we had set the suppression of obstacles by driveable 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 re-use allowed an optimal plan directly through the grass.

7.2 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 realtime 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 since many small obstacles such as branches could be traversed at speed, but might hang up the robot if it was moving slower. As described in Section 6, the path planner and local controller combined to give the robot a very agile feeling. Our average speed was over 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 7m. Even using monocular information, there were very few pixels available for long-range sensing. In Figure 14(left), a high-resolution camera with a longer focal length clearly shows routes around the barrier. But with a similar distance to the tree on the right image, 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 re-use. 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 ($\approx$2m 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 re-use 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 15. In all these tests, we had the best score (or tied for the best). Typically we out-performed 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 re-use enabled us to do even better. The controller, planner, and visual odometry system were used in the best-in-class NIST system (see XXX in this issue), and in fact NIST was our closest competition in two of the tests, including the difficult Test 26b.

8 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. But there are hazards that are not dealt with by the methods discussed in this paper: water and ditches are two robot-killers. Finally, we would like to use visual landmarks to augment GPS for global consistency, because it would give finer adjustment in the robot's position, which is critical for following routes that have already been found.

References

Agrawal, M. and Konolige, K. (2006). Real-time localization in outdoor environments using stereo vision and inexpensive gps. In *Intl. Conf. of Pattern Recognition (ICPR)*.

Agrawal, M. and Konolige, K. (2007). Rough terrain visual odometry. In *Proc. International Conference on Advanced Robotics (ICAR)*.

Agrawal, M., Konolige, K., and Blas, M. R. (2008). CenSurE: Center surround extremas for realtime feature detection and matching. In *ECCV, Submitted*.

Angelova, A., Matthies, L., Helmick, D., Sibley, G., and Perona, P. (2006). Learning to predict slip for ground robots. In *ICRA*, pages 3324–3331.

Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *European Conference on Computer Vision*.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

Bellutta, P., Manduchi, R., Matthies, L., Owens, K., and Rankin, A. (2000). Terrain perception for DEMO III. In *Proc. of the IEEE Intelligent Vehicles Symp.*

Blas, M. R., Agrawal, M., Konolige, K., and Sundaresan, A. (2008). Fast color/texture segmentation for outdoor robots. In *IROS, Submitted*.

Engels, C., Stewnius, H., and Nister, D. (2006). Bundle adjustment rules. *Photogrammetric Computer Vision*.

Fischler, M. and Bolles, R. (1981). Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Commun. ACM.*, 24:381–395.

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

Guivant, J., Nebot, E., and Baiker, S. (2000). High accuracy navigation using laser range sensors in outdoor applications. In *ICRA*.

Gutmann, J. S. and Konolige, K. (1999). Incremental mapping of large cyclic environments. In *CIRA*.

Howard, T., Green, C., and Kelly, A. (2007). State space sampling of feasible motions for high performance mobile robot navigation in highly constrained environments. In *Proc. of the Intl. Conf. on Field and Service Robotics*.

Iagnemma, K., Genot, F., and Dubowsky, S. (1999). Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In *ICRA*.

Kelly, A. (1994). A feedforward control approach to the local navigation problem for autonomous vehicles. Technical Report CMU-RI-TR-94-17, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Kimmel, R. and Sethian, J. A. (1998). Computing Geodesic Paths on Manifolds. *Proc. of the National Academy of Science*, 95:8431–8435.

Konolige, K. (1997). Small vision systems: hardware and implementation. In *Intl. Symp. on Robotics Research*, pages 111–116.

Konolige, K. (2000). A gradient method for realtime robot control. In *IROS*.

Konolige, K. and Agrawal, M. (2008). Frameslam: from bundle adjustment to realtime visual mappping. *Transaction on Robotics*, Submitted.

Konolige, K., Agrawal, M., and Solà, J. (2007). Large scale visual odometry for rough terrain. In *Proc. International Symposium on Robotics Research*.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, Massachusetts.

LaValle, S. (2006). *Planning Algorithms*. Cambridge University Press, New York, New York.

Leonard, J. J. and Newman, P. (2003). Consistent, convergent, and constant-time slam. In *IJCAI*.

Leung, T. and Malik, J. (2001). Representing and recognizing the visual appearance of materials using three-dimensional textons. *IJCV*, 43 (1).

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Intl. J. of Computer Vision*, 60(2):91–110.

Montemerlo, M. and Thrun, S. (2004). Large-scale robotic 3-d mapping of urban structures. In *ISER*.

Moravec, H. and Elfes, A. (1985). High resolution maps for wide angles sonar. In *ICRA*.

Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., and Sayd, P. (2006). Real time localization and 3d reconstruction. In *CVPR*, volume 1, pages 363 – 370.

Nister, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *CVPR*.

Olson, C. F., Matthies, L. H., Schoppers, M., and Maimone, M. W. (2000). Robust stereo ego-motion for long distance navigation. In *CVPR*.

Philippsen, R. and Siegwart, R. (2005). An interpolated dynamic navigation function. In *ICRA*.

Rankin, A., Huertas, A., and Matthies, L. (2005). Evaluation of stereo vision obstacle detection algorithms for off-road autonomous navigation. In *AUVSI Symp. on Unmanned Systems*.

Spero, D. J. and Jarvis, R. A. (2002). 3D vision for large-scale outdoor environments. In *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*.

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *ICRA*, volume 4, pages 3310–3317.

Sunderhauf, N., Konolige, K., Lacroix, S., and Protzel, P. (2005). Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle. In *Tagungsband Autonome Mobile Systeme*. 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., and 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., and Fitzgibbon, A. W. (2000). Bundle adjustment - a modern synthesis. In *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag.

Varma, M. and Zisserman, A. (2003). Texture classification: Are filter banks necessary? *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2:691–696.

Viola, P. and Jones, M. (2001). Robust real-time face detection. In *ICCV01*.