A Multirange Architecture for Collision-Free Off-Road Robot Navigation

Pierre Sermanet
Courant Institute of Mathematical Sciences
New York University
New York, New York 10003
Net-Scale Technologies
Morganville, New Jersey 07751
e-mail: pierre.sermanet@gmail.com

Raia Hadsell
Courant Institute of Mathematical Sciences
New York University
New York, New York 10003

Marco Scoffier
Courant Institute of Mathematical Sciences
New York University
New York, New York 10003
Net-Scale Technologies
Morganville, New Jersey 07751

Matt Grimes
Courant Institute of Mathematical Sciences
New York University
New York, New York 10003

Jan Ben
Net-Scale Technologies
Morganville, New Jersey 07751

Ayse Erkan
Courant Institute of Mathematical Sciences
New York University
New York, New York 10003

Journal of Field Robotics 26(1), 52–87 (2009)
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/rob.20270

Chris Crudele and Urs Miller
Net-Scale Technologies
Morganville, New Jersey 07751

Yann LeCun
Courant Institute of Mathematical Sciences
New York University
New York, New York 10003

Received 7 April 2008; accepted 10 October 2008

We present a multilayered mapping, planning, and command execution system developed and tested on the LAGR mobile robot. Key to robust performance under uncertainty is the combination of a short-range perception system operating at high frame rate and low resolution and a long-range, adaptive vision system operating at lower frame rate and higher resolution. The short-range module performs local planning and obstacle avoidance with fast reaction times, whereas the long-range module performs strategic visual planning. Probabilistic traversability labels provided by the perception modules are combined and accumulated into a robot-centered hyperbolic-polar map with a 200-m effective range. Instead of using a dynamical model of the robot for short-range planning, the system uses a large lookup table of physically possible trajectory segments recorded on the robot in a wide variety of driving conditions. Localization is performed using a combination of global positioning system, wheel odometry, inertial measurement unit, and a high-speed, low-complexity rotational visual odometry module. The end-to-end system was developed and tested on the LAGR mobile robot and was verified in independent government tests.

1. INTRODUCTION

Walking through a park, a forest, or a mountain trail, humans and animals combine information at many different spatial resolutions and on many different time scales to help them plan a trajectory. The general direction is determined by the heading of the goal, combined with cues from long-range vision, and sometimes constrained by the direction of the trail being followed. Long-range visual cues are refined over time on a relatively slow time scale, by combining information from multiple viewpoints as the subject moves along. The long-range plan is updated periodically, whenever the accumulated evidence calls for it. Short-range planning, on the other hand, requires constant attention and fast reaction times. Nearby obstacles must be detected quickly, so that the internal representation of the immediate surroundings can be updated on a fast time scale. The map of the immediate surroundings must have considerably higher spatial resolution than the map of far away locations, but this high resolution is somewhat ephemeral: we quickly forget the details of all the obstacles we circumvented. This accords with studies of human subjects, who have been shown to focus on nearby objects much more frequently than on distant areas (Wagner, Baird, & Barbaresi, 1980). Occasional distant gazing suffices to maintain a global trajectory, but frequent nearby gazing is necessary for obstacle avoidance.

This motivates the design of navigation systems for autonomous robots in which a number of perception–mapping–planning–control loops operate at different ranges, different spatial resolutions, different image resolutions, and different time scales. A fast control loop with low latency and high frame rate constructs a short-range map with high spatial resolution from short-range vision at low image resolution, whereas a slower control loop with higher latency and lower frame rate constructs a long-range map with low spatial resolution using long-range vision at maximum image resolution. The short-range planning is updated on a fast-time scale and takes the vehicle dynamics into account, whereas the long-range planning is updated rarely and ignores dynamical constraints.

This paper describes such a multi-time scale, multiresolution, multirange architecture for vision-based off-road robot navigation. The system was implemented and tested on the LAGR platform. The present paper concentrates on the architecture and system-level issues, as well as on the fast-reacting, short-range vision system, the learning-based

dynamical trajectory control, the probabilistic mapping, the multirange planning strategy, and the localization and odometry subsystems. The adaptive long-range vision is described in full detail in a companion paper (Hadsell et al., 2009) and very briefly described in the current paper.

1.1. Overview

The LAGR robot platform, shown in Figure 1, is somewhat unique in the world of outdoors mobile robots in that it uses no active sensors and relies solely on cameras to sense its environment. This was a deliberate choice in the program designed to promote new advances in robot vision, as well as in the application of machine learning to mobile robotics. The LAGR robot has two forward-looking stereo camera pairs with a combined field of view of 120 deg, a global positioning system (GPS), a low-cost inertial measurement unit (IMU), wheel encoders, and a front bumper with bump sensors. It can travel at 1.3 m/s (fast walking speed).

The implementation of a fully functional, reliable, and efficient vision-based navigation system for off-road robots such as the LAGR platform is a very difficult challenge for robotics research. In contrast with indoor environments and constrained outdoor environments (roads and trails), off-road scenes and natural obstacles can vary widely in shape, color, texture, and lighting.

Figure 1. The LAGR platform has no active external sensors; it relies solely on two stereo camera pairs (four cameras). A GPS, a low-cost IMU, and wheel encoders provide odometry data. It is a differential drive (nonholonomic) off-road vehicle measuring 1.2 m in length × 0.74 m in width × 1.02 m in height. It weighs 109 kg and can travel at up to 1.3 m/s. The robot contains three computers with dual-core CPUs: two "eye" machines for vision processing of each stereo pair and one "planner" machine for planning. A fourth computer takes care of low-level control tasks.

A key question is how to construct a long-range vision system that can reliably label far away objects and regions as traversable or nontraversable and adapt to new environments. This question is addressed in the companion paper (Hadsell et al., 2009). With the limited resolution and small stereo baseline of the camera pairs, the practical range of stereo-based algorithms is limited to about 15–20 m. With such shortsightedness, a pure stereo-based system can get trapped in cul-de-sacs and waste a considerable amount of time trying to find its way around wide obstacles. Furthermore, stereo often produces no depth values (or erroneous ones) for a large proportion of the pixels, when the image is too uniform, too repetitive (like tall grass), or too saturated by the sun or by dark shadows. A long-range monocular vision module is required. Our module is based on a convolutional network that is trained offline to extract features over large sliding windows in the images, followed by a linear classifier fed with these features and adapted online using labels provided by a stereo vision system.

The main topic addressed in the present paper is how to integrate the various long-range and short-range perception, mapping, planning, and control modules into a coherent architecture.

The system uses three visual perception modules. A first stereo-based system operating at 160 × 120 resolution and 5–10 frames per second establishes a traversability map within a 5-m radius from a stereo point cloud. A second stereo system with a 12-m range uses multiple heuristics to establish a medium-range traversability map. Finally, a monocular convolutional network operating at 512 × 384 resolution and 1 frame per second estimates the traversability of every overlapping window in the input image at multiple scales. The last layer of the convolutional network is adapted online using labels produced by the medium-range stereo system, in accordance with the "near-to-far learning" methodology (Stavens & Thrun, 2006).

A particularly important question is how to cope with the uncertainty in the output of the perception modules and how to reduce this uncertainty by accumulating evidence about the traversability from multiple perception modules over long periods of time. The perception modules produce two kinds of uncertainties: uncertainty about the category (or traversability label) of a particular region and uncertainty about the location of that region in relation to the robot.

Distance estimates of faraway objects (e.g., trees) that appear near the horizon in the image are very imprecise. As the robot moves, these distance estimates fluctuate. When using a Cartesian traversability map with fixed resolution, these objects are often smeared out over many map cells. Our solution is to use a robot-centered *hyperbolic-polar map* (h-polar map) in which the radial size of a cell increases hyperbolically (to infinity) with the distance from the robot. Such a map can be seen as a virtual image-plane map, because the radial depth of each map cell corresponds to the growing uncertainty of object distances, which diverge to infinity as they get closer to the horizon in the image plane.

For each frame, and each pixel in the frame, the perception modules produce a vector of confidence values, with one value for each possible category of the region around the pixel. The categories are supertraversable (very smooth terrain), traversable, obstacle foot (the location where an obstacle meets the ground), obstacle, and superobstacle. The confidence vectors produced from a single frame may contain occasional mistakes. Our system accumulates these confidence vectors in the h-polar map as the robot moves and sees the same locations from multiple viewpoints, thereby reducing the uncertainty and the noise of individual classifications. Before planning, the map confidence histograms are transformed into traversability costs.

An important remaining question is how to integrate the dynamical constraints of the robot into fast-reacting, short-range planning. Our system uses a large dictionary (or lookup table) of initial trajectory snippets (also known as *maneuvers*) with a 2.5-m radius that have been collected by driving the robot under human control in a wide variety of conditions.

1.2. Related Work

Learning algorithms allow robots to adapt to new situations and gain experience. Learning can be applied in a number of ways to improve navigation, e.g., learning terrain slippage (Angelova, Matthies, Helmick, & Perona, 2006) and traversability (Angelova, Matthies, Helmick, & Perona, 2007) to improve mobility or to avoid getting stuck, with human supervision and Bayesian estimates (Ollis, Huang, & Happold, 2007), learning depth from monocular images (Saxena, Schulte, & Ng, 2007), or learning natural and man-made pathways from color and texture (Blas, Agrawal, Konolige, & Sundaresan, 2008). Learning algorithms can also be used to extend stereo vision range with near-to-far learning, learning the long-range appearance of terrains and obstacles using color features (Albus et al., 2006) with Markov random fields (Vernaza, Taskar, & Lee, 2008), with suitable features and a linear support vector machine (Bajracharya, Tang, Howard, Turmon, & Matthies, 2008), or using reverse optical flow (Lookingbill, Lieb, & Thrun, 2007). Typically trained offline, features can also be learned online to provide a continuous adaptability (Grudic, Mulligan, Otte, & Bates, 2007).

A navigation software developed by Carnegie Mellon University (CMU) and the National Robotics Engineering Center (NREC), called "Baseline," was used by the government testing team to measure improvements during monthly tests. The Baseline system can be seen as a benchmark representing the state of the art around the end of 2004. It is based on medium-range stereo vision, a Cartesian/grid map containing traversability scores, and an incremental long-range planning algorithm called D* (Stentz, 1994). In the PerceptOR program developed at CMU, which preceded the LAGR program, a number of the issues that plagued the Baseline system were addressed (Kelly et al., 2006). A multirange architecture was proposed, but it was used only for planning. A similar maneuver-based local planner was used. Finally, only two levels of planning were used, local and global grid-based planning, whereas we advocate for an additional intermediate level of long-range but local vision and planning coupled with a global topological planning. A key shortcoming of Cartesian traversability map approaches is that they are very sensitive to the accuracy with which the robot is localized in the map. This is rather unfortunate, because GPS fixes can be quite inaccurate and even nonexistent (e.g., in heavily wooded areas).

2. SYSTEM ARCHITECTURE

The overall system architecture is depicted in Figure 2. Each level operates at lower frequency and higher level than the level below. The left half of the figure contains the perception modules, and the right part contains the planning and control modules.

Layer 1 contains a shortsighted (5 m) but robust and fast perception module (1-a, called Fast-OD for fast obstacle detection). The module populates a robot-centered Cartesian map (1-b) with a 5-m radius and a 20-cm resolution. The module runs a simple stereo algorithm on 160×120 image pairs. The ground plane is first detected, and a region is marked as obstacle if many points within that region are dangerously above the ground plane. The map is used by the maneuver-based trajectory planning subsystem. The best path from each point on a robot-centered circle with 2.5-m radius to each point on the 5-m radius is computed using Dijkstra. Then for each point on the 5-m radius, the Layer-2 planning

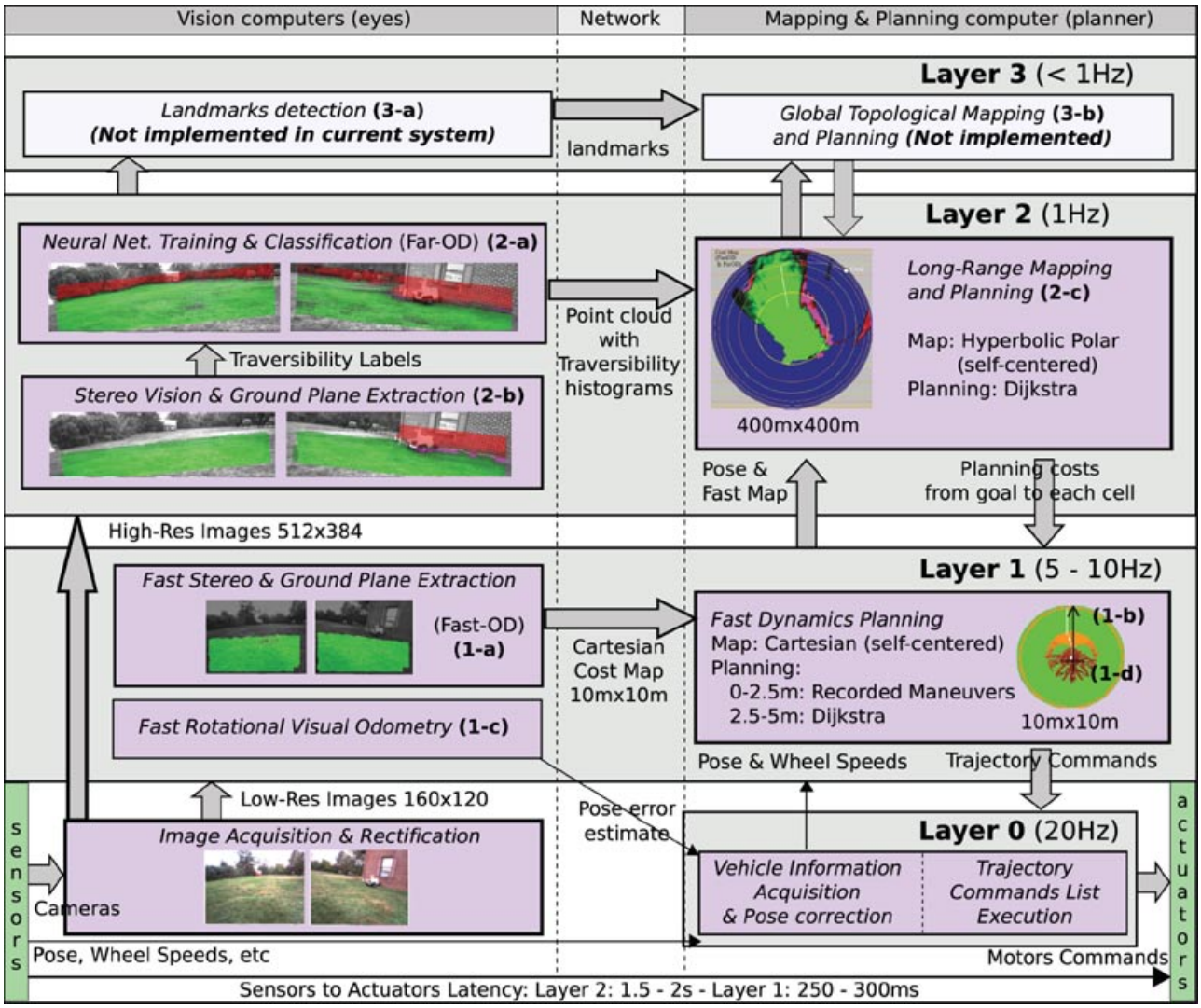


Figure 2. Overall system architecture. Each layer contains perception, mapping, and planning modules (except for Layer 0). From bottom to top, sensor–actuator latency and period increase while resolutions decrease. Layers 1 and 2 are self-centered, and Layer 3 is global. Layer 0, also known as the control loop, guarantees the flow of motor commands required at 20 Hz. Layer 3 was not implemented in the current system because the scale of the LAGR tests did not require global mapping when our local mapping can handle a 200-m radius.

module computes a cost to the goal. The short-range planner searches for maneuvers in the repertoire whose initial states match the current dynamical state (wheel speeds) of the robot (1-d). It then picks the maneuver that has the lowest overall cost to the goal while staying clear of lethal obstacles. Accurately estimating the orientation of the robot within the map is paramount. An error in the orientation has catastrophic effects on the crispness of the map, as obstacles get smeared across the map as the robot rotates. Because the wheels often slip on the ground, and the IMU drifts quickly, a simple and efficient rotational visual odometry (VO) module estimates the angle by which the robot has rotated since the last frame (1-c). The VO module tracks a number of distinctive windows on faraway objects near the

horizon from one frame to the next. Layer 1 operates at 5–10 frames per second.

Layer 2 contains the long-range obstacle detection system (called Far-OD), which is built around a large convolutional network (Hadsell et al., 2009) operating on horizon-leveled image bands extracted from input images at 512 × 384 resolution (2-a). The advantages of convolutional networks for this application is that they automatically learn appropriate features representations through supervised or unsupervised training methods, and they are relatively fast and simple. More importantly, they can compute a label for each pixel based on a large input window surrounding the pixel. Therefore, they can go beyond the popular but limited approach to near-to-far learning that uses simple color histograms as inputs. The last layer of the convolutional network, which can be seen as a logistic regression classifier, probabilistically classifies the image regions into five traversability labels. This last layer automatically adapts to new environments as the robot runs. The desired labels are provided to the classifier by a stereo-based obstacle detection system (2-b) with a range of 12 m. This system uses multiple ground plane fitting and a number of point-cloud heuristics to produce reliable labels. The vectors of confidences over labels are mapped from their position in the image plane to their corresponding location in the world (or in the h-polar map) using a ground-plane assumption. The confidence vectors are accumulated into histograms stored at their corresponding cell in the h-polar map (2-c). After each frame, the accumulated histograms are mapped to traversability costs, and the Dijkstra planning algorithm is launched. Although the radius of the h-polar map and the long-range vision system is infinite in principle, its effective radius is roughly 100–200 m. Overall, Layer 2 operates at one frame per second.

Layer 3 is included in the figure for completeness but is not implemented in the system described in this paper. Its presence would be necessary for the robot to navigate long courses whose extent goes beyond the effective radius of the h-polar map. It would contain a very long-range topological map (3-b) in which visually distinctive landmarks are detected (3-a) and saved.

By analogy with the navigation mechanism in humans, we know which streets to take to get to a restaurant (Layer 3). We can see the corner to turn two blocks away, and we know which way to turn (Layer 2). We walk there while avoiding bumping into other pedestrians (Layer 1). We occasionally think about the global-level route planning (Layer 3); we do not need to look often to the far away end of the block (slow Layer 2) but we will look very often to avoid other pedestrians (fast Layer 1).

3. MULTIRANGE PERCEPTION AND PLANNING

Multilayer architectures (Figure 2), first introduced by Brooks (1986), provide a crucial latency and frequency independence between different levels of planning but also different levels of perception. Key components of each layer are further described. We will first discuss motivation and implementation details for the multilayer, multirange architecture.

3.1. Motivation and Initial Experiments

As the sophistication and computation load of our advanced vision module was increasing, our initial sequential navigation architecture would cause the robot to collide with an increasing number of obstacles. To quantify the relationship between control-loop period and system responsiveness in a traditional sequential architecture, we set up a field experiment that would compare the responsiveness of our sequential system as a function of different control-loop periods and measure the number of collisions on a fixed course. To ensure that the performance of the system was affected only by the control-loop period, we use a single, fixed, midrange obstacle avoidance module. For each test, the control-loop period was artificially increased, but the obstacle avoidance processing and the test course were kept fixed.

The LAGR vehicle was tested in a picnic area at Holmdel Park, New Jersey. The course had eight large trees, two barrels, and two picnic tables, and the goal was 30 m away. The start position faced 90 deg away from the goal, and there was a barrel 3 m from the start. When a run was started, the barrel would not be seen until the robot turned toward the goal, at which point the robot would have to quickly turn to avoid the barrel. The vehicle was tested using four different control-loop periods (time from frame timestamp to issuing drive commands): 360, 510, 660, and 960 ms.

Results of the experiment are summarized in Figure 3. The shortest loop period performed significantly better than any of the other test modes, particularly in avoiding the first obstacle. The intermediate loop periods produced poorer performance: the vehicle usually crashed into the first barrel but could often navigate around other obstacles (although not as consistently as in the shortest loop). The longest period (960 ms) produced abysmal driving, repeated crashes into obstacles.

In addition, the latency plots presented later in the Timing Results section show that an incompressible sensor latency of 190 ms already handicaps the robot's reactivity. Similarly, inertia of the vehicle as shown by plots of Figure 4 also adds latency to the reactivity. The reactivity latency caused by vehicle

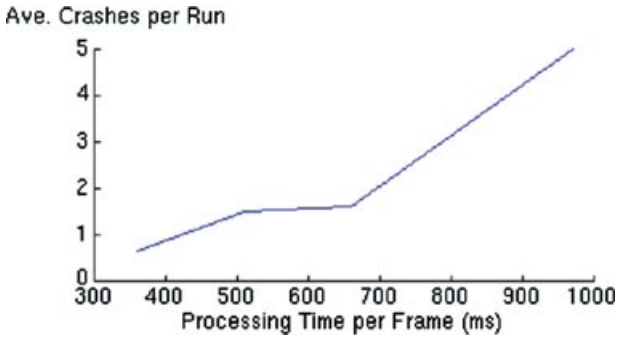


Figure 3. Experimental results: performance of the early sequential system, as measured by the number of collisions per run, for different control-loop periods. The performance is roughly linearly correlated with the length of the control loop.

dynamics is addressed by maneuver-based planning in further sections. Although the sensors and processing latencies can also be addressed by estimating delays before and after planning, the vehicle still gains in reactivity by minimizing latencies.

Layered architectures have been used for a long time, and this experiment confirms once more the importance of reactivity versus vision range in close combat navigation. Wagner et al. (1980) investigated the gaze behavior of walking humans in an outdoor environment. The results indicated that the human gaze was directed at objects close to the observer a majority of the time. Bayouth, Nourbakhsh, and Thorpe (1997) proposed a hybrid human-computer layered architecture for highway driving. Beetz (2001) described a method for integrating perception, planning, and control in a uniform framework and demonstrated the framework with a service robot. Low, Leow, and Ang (2002) presented a method for unified planning and motion control for a mobile robot.

3.2. Implementation Details

Here we specify which characteristics each module should have for different layers. For example, the naming of the obstacle detection modules (OD) indicate their characteristics: Fast-OD must be fast but does not have to see far away, whereas Far-OD has to see far but not necessarily be fast.

3.2.1. Perception Modules Characteristics

The perception processes are run on the "eye" machines and transmit the traversability data over the network to the "planner" machine. Because it has to be fast and does not need to be long range, the resolution of the Fast-OD can be as low as 160×120 compared to traditional systems. Along with the low resolution, stereo processing heuristics producing the traversability map are kept simple in order to guarantee the low latency of the Fast-OD. On the contrary,

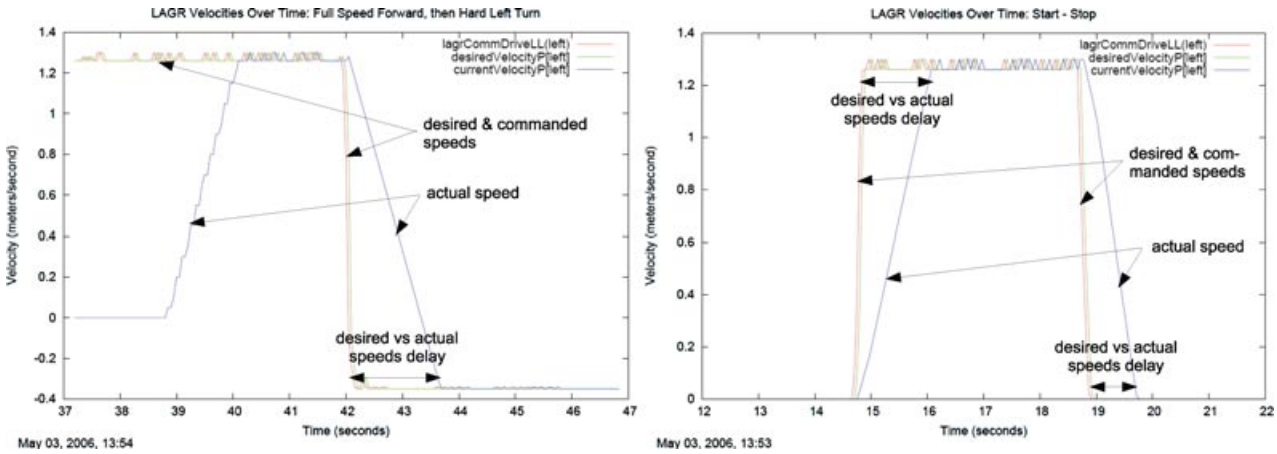


Figure 4. To illustrate the dynamics of the LAGR vehicle (109 kg), the plots show the desired speed (green and red) of the left wheel versus its actual speed (blue). In the left plot, the robot is driven full speed forward followed by a hard left turn. It takes the left wheel about 1.6 s to go from the full forward speed of 1.3 m/s to the full backward speed of −0.35 m/s. Similarly in the right plot, the robot is driven from 0 m/s to full speed forward and then back to 0 m/s. It takes the robot about 1.5 s to reach the full speed and about 1 s to stop.

the Far-OD uses higher resolution input (512 × 384) to reach as far as possible (>100 m) but is allowed to be slow.

3.2.2. Mapping and Planning Modules Characteristics

Each planning layer must be able to operate independently of the layer above for some time and produce data for the lower level. For example, in the Layer-2 (hyperbolic polar) map, all best paths from the goal to each cell are computed once per Layer-2 iteration (1 Hz) and passed to the Layer-1 planner (5–10 Hz). At Layer 1, knowing the current translation and rotation relative to the Layer-2 map, the globally optimal path can be computed using the locally optimal paths reaching all points at a fixed radius (5 m). The transposed local candidates cost into the Layer-2 map are added to the precomputed paths cost, and the minimum sum determines the globally optimal path. When the best local path or trajectory is found, the sequence of motor commands to execute is sent to the Layer-0 control loop (20 Hz). The control loop can now operate independently by just iterating through the commands sequence. Because vehicle-centered maps are used, maps must be recentered at every iteration of the layer before a data update. Thanks to the layered architecture, not only is the planning executed once per level, but also the mapping and recentering, which is a costly process in the case of the hyperbolic polar map. As pointed out earlier, latency from acquisition to planning is minimized as much as possible but is also addressed by recentering maps on the last known pose. Delays after planning are thus not taken into account and must remain nearly null. Another advantage of using layers is that different planning and mapping schemes best suited for different ranges can easily be combined. For example, the hyperbolic polar mapping is suited for far-range data, whereas Cartesian mapping is more adequate for the short range, because it is simple and fast. Similarly, Dijkstra planning is convenient for the long range, whereas dynamics planning is required in the short range. More details about the actual mapping and planning algorithms are given further in the components sections.

3.2.3. Process Priorities

When running different layers on the same CPU, the lowest levels require an increasing execution priority as the latency minimization and frequency maximization requirements increase. This can be achieved by using the priority scheduling features of a real-time operating system or with a regular operating system by giving control of the higher level processes to the lower level processes. For example, when receiving a new frame to process, the Fast-OD pauses the Far-OD process and resumes it when processing is done. Then the Fast-OD can pause itself for a fixed amount of time to allow some processing cycles to the Far-OD. A trade-off must be found depending on the available CPU budget, running speed, and vision distance. At the risk of losing reactivity to moving obstacles, one could extend this idea by giving more CPU cycles to Far-OD when going straight and more to Fast-OD when the robot turns. This custom priority scheduling was successfully tested on the first generation of the LAGR vehicle but was no longer needed after a hardware upgrade to dual-core CPUs.

The timing plots presented later in the Timing Results section show the actual latencies and frequencies obtained for Layers 1 and 2 on the dual-core machines.

4. LAYER 2: LONG-RANGE PERCEPTION, MAPPING, AND PLANNING

We describe in this section the complete Layer 2 of Figure 2, and in particular the mapping and planning approaches used to address the issues inherent to long-range vision: classification (Figure 5) and distance uncertainties (Figure 6) and computational efficiency. The mapping method allows the accumulation of evidence from multiple frames in a principled manner. The geometry of the map accurately reflects the range uncertainty associated with image-plane obstacle labeling. It can furthermore represent an effectively infinite radius with a finite number of cells. Last, the information in the map allows dynamic adjustment of planning policy so as to be more aggressive or more conservative. A key element for long-range mapping is the ability to accurately detect the bottom (or foot) of obstacles, allowing "blocking" obstacles beyond their foot. By using only the bottom of obstacles, we avoid projecting down on the ground plane obstacles for which no elevation information is available and creating false obstacles. We present the high-level mapping and planning in the current section; the learned low-level mapping and planning layer (Figure 7) will be fully described in the next section.

4.1. Long-Range Vision

Our long-range vision learning approach briefly introduced here is fully described in Hadsell et al. (2009). The existing paradigm for vision-based mobile robots relies on hand-tuned heuristics: a stereo algorithm produces a (x, y, z) point cloud, and traversability costs are assigned to points based on their proximity to a ground plane (Goldberg, Maimone, & Matthies, 2002; Kelly & Stentz, 1998).

However, stereo algorithms that run in real time often produce cost maps that are short range, sparse, and noisy. Our learning strategy uses these stereo labels for supervision to train a real-time classifier. The classifier then predicts the traversability of all visible areas, from close range to the horizon. For accurate recognition of ground and obstacle categories, it is best to train on large, discriminative windows from the image, because larger windows give contextual information that is lacking in color and texture features. Other research has explored the use of online learning for mobile robots, but their methods have been largely restricted to simple color/texture correspondences (Manduchi, Castano, Talukder, & Matthies, 2003; Sofman, Lin, Bagnelli, Vandapel, & Stentz, 2006).

4.2. Mapping with Categorical Uncertainty: Histograms

4.2.1. Classifier Output

The network produces a floating-point value for each of the five following classes (Figure 5) detected by the stereo module: superground, ground, foot-line, obstacle, superobstacle. Having more visually consistent classes helps the network to produce better classification in comparison to a binary classifier as used in previous versions of the system. In addition, it conveniently suits the histogram scheme used in our planning cost decision algorithm described in the following section.

4.2.2. Label Uncertainty: Cost from Histogram

Label uncertainty can be caused by differences of viewpoints, sensor noise, or learning phase of the online neural network. Mapping with label uncertainty in mobile robotics has been addressed by several approaches from Bayesian techniques (Elfes, 1991) to fuzzy logic (Ulivi & Vendittelli, 1997). Trivial approaches such as using latest labels only or running averages are simple and fast but lack confidence and accuracy and would cause an early fusion of the multiclass network outputs. We use a histogram approach (Figure 8) suited for multiple classes and able to delay the traversability decision to planning time: each cell contains K bins, each bin corresponding to a class (or a range of traversability for single class outputs). Each new label is merged in a cell through a simple addition. Before planning, the histogram is translated into a traversability cost (Figure 9). At that time, the traversability decision can be modulated by the current planning policy: conservative vs. aggressive. As label uncertainty increases with distance, a distance decay must be applied to incoming frames.

4.2.3. Histogram to Cost Transformation

Let $\sum_k \text{bin}_k + c_{\text{uncertain}}$ be the sum of all bins of a histogram with an added constant $c_{\text{uncertain}}$, which brings confidence to 0 when the sum is small. Let $\sum_k w_k \text{bin}_k$ be the weighted sum of all bins. w_k are weights empirically tuned with real examples. The weight values w_k used from superground to foot in the order of Figure 8 are -1.0, -0.7, 0.7, 0.9, and 1.8. The normalized sum S tuned by gain parameters γ is defined by Algorithm 1.

Algorithm 1 Histogram to cost

if $(S <= 0)$ **then**
  $cost = cost_{unexplored} + S \times (cost_{unexplored} - cost_{min})$
else if $(S <= 1)$ **then**
  $cost = cost_{unexplored} + S \times (cost_{lethal} - cost_{unexplored})$
else $cost = cost_{lethal}$
end if
$cost = MAX(cost, cost_{min})$

4.3. Mapping with Spatial Uncertainty: Hyperbolic Polar Map

Our map representation is based on two key concepts: (1) hyperbolic range mapping and (2) representing categorical evidence by accumulated histograms. In the image plane, a single pixel covers a constant angular extent but covers a wildly varying range of distances. A single pixel on nearby ground (near the bottom of the image) covers a few centimeters, whereas a pixel near the horizon covers an essentially infinite range of distances. For a flat ground plane, the mapping of image-plane pixels to distances is hyperbolic from the bottom of the image to the horizon. For this reason, we propose to represent the environment through a robot-centered *hyperbolic-polar map* (h-polar). This representation allows us to map the entire world to a finite number of cells, while being faithful to the type of uncertainty afforded by image-plane labels. To allow the accumulation of evidence for the label of a cell as the robot moves and collects data, each cell in the map contains a *histogram of accumulated probabilities for each category*. At each frame, the classifier produces likelihood values for each category that are accumulated in the histogram of the corresponding map cell.

Previous work on robot-centered, nonuniform mapping includes log-polar representations (Longega, Panzieri, Pascucci, & Ulivi, 2003) and multiresolution grid-based maps (Behnke, 2004). The main advantage of the h-polar approach over these methods is the ability to represent an effectively infinite radius with a finite number of cells. Even more important is a representation of range uncertainty that directly corresponds to that associated with

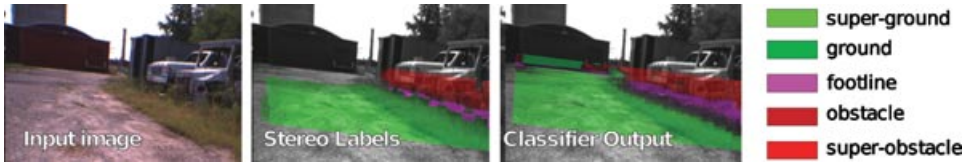


Figure 5. Neural network output: Five classes. The input image (left) is fed into the neural network for classification at resolution 512 × 384, producing traversability labels (overlaid on the right image) up to 100 m. The network is trained both offline and online using the stereo labels (center) up to 12 m for self-supervision.

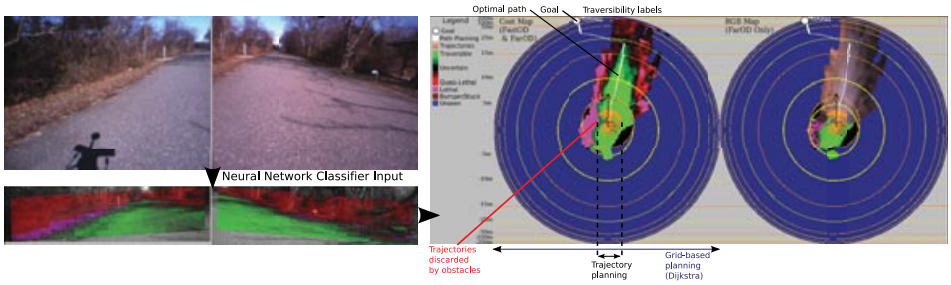


Figure 6. Multilayer planning with trajectories. Long-range (bottom left) and short-range labels are accumulated in the h-polar map (right). The optimal path in white is computed through multiple planning layers, with Dijsktra at Layer 2 and trajectories (orange curves) planning at Layer 1 on a 10 × 10 m area.

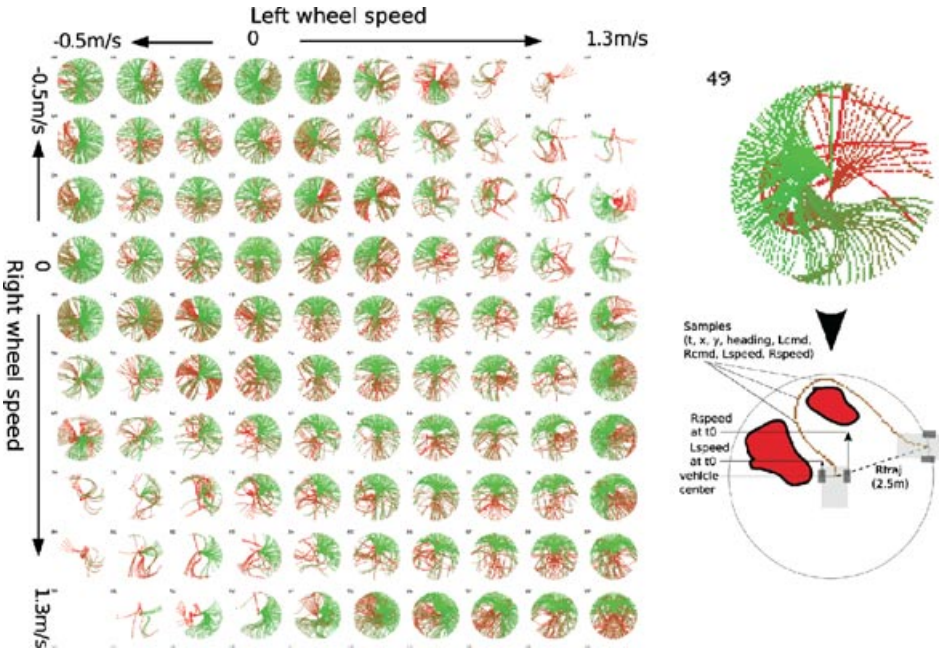


Figure 7. Maneuver dictionary (left) and execution example (right). The maneuver bank shows 100 sets of possible trajectories for each initial speed. The robot initially goes full speed backward at the top left corner and full speed forward at the bottom right. For each set, trajectories are colored from green for the fastest ones to reach the 5-m radius to red for the slower ones, e.g., it is obvious based on colors that turning left when already turning hard left is faster. For example with initial state (4, 9) on the right, i.e., initially turning hard left, the robot can avoid the front obstacle only with a large left turn around it.

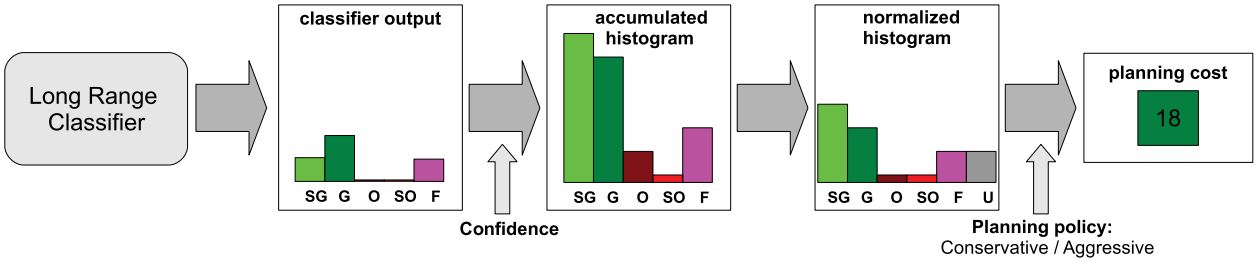


Figure 8. Histogram to cost process. The output of the classifier is multiplied by the current learning confidence and added to an existing histogram. Before converting the histogram to a planning cost, it is normalized. Bin U in the normalized histogram corresponds to constant $c_{\text{uncertain}}$, which brings confidence to 0 when the sum S is small (uncertainty is shown in Figure 9). Finally, the current planning policy modulates the mapping from the normalized histogram to a single planning cost.

image-plane labeling. Pure image-plane labeling and planning (Zhang & Ostrowski, 2002), or visual motion planning, presents the advantage of being free of expensive transformations and pose errors but can operate only with one single frame at the time. The accumulation of evidence over multiple frames can greatly reduce the perception uncertainty and noise that single frame planning is subject to. In addition, visual motion planning relies on the goal being within the current field of view, a dangerous assumption in complicated outdoor scenes. A previous version of our system (Hadsell et al., 2007) used a robot-centered, tactical Cartesian grid mapping with a radius of 30 m. Each time the robot moves and a new camera frame is analyzed, the robot-centered map must be translated and rotated before the results from the new frames are incorporated. This process becomes too expensive for large Cartesian maps. With our new perception system with a range of 100 m, a robot-centered Cartesian map with 20-cm resolution would have required an impractical 500×500 cells. The h-polar representation offers a considerably more efficient use of memory and CPU.

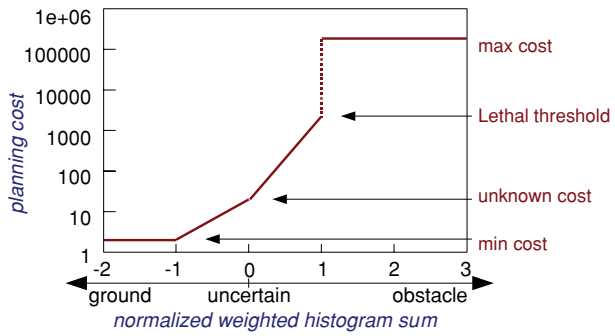


Figure 9. Planning cost function from normalized histogram sum S. An input sum S of 0 represents the full uncertainty, -1 and below are given the minimum ground cost, and 1 or higher the maximum cost.

4.3.1. Overall Process

The overall processing sequence for the h-polar map from image labeling to planning is the following:

1. **Image labeling (Figure 10):** Each pixel of the input images (512×384 in our system) is labeled with a histogram, representing confidences in each class. Each pixel is associated with (r, c, d) coordinates, i.e., row, column, and disparity. The disparity (given by stereo) indicates the depth of the point.
2. **Plane extraction (Figure 10):** A single plane is found using the short-range stereo points (up to 12 m). Remaining points with no disparity information are projected onto the single plane and given a disparity. (r, c, d) is then converted to vehicle-centered (x, y, z).
3. **Point cloud transmission:** The point cloud of the $(x, y, z,$ histogram) is sent over to the h-polar module.
4. **Recentering memory:** Previously accumulated frames are recentered onto the last available pose.
5. **Adding new point cloud:** Each point of the new point cloud is merged into the h-polar internal point cloud.
6. **Produce planning map (Figure 11):** A temporary map for planning and display is produced from the h-polar point cloud.
7. **Planning:** Best paths from the goal to all points of the map are computed and sent to the reactive planning layer. The optimal path is shown in white in Figure 11.

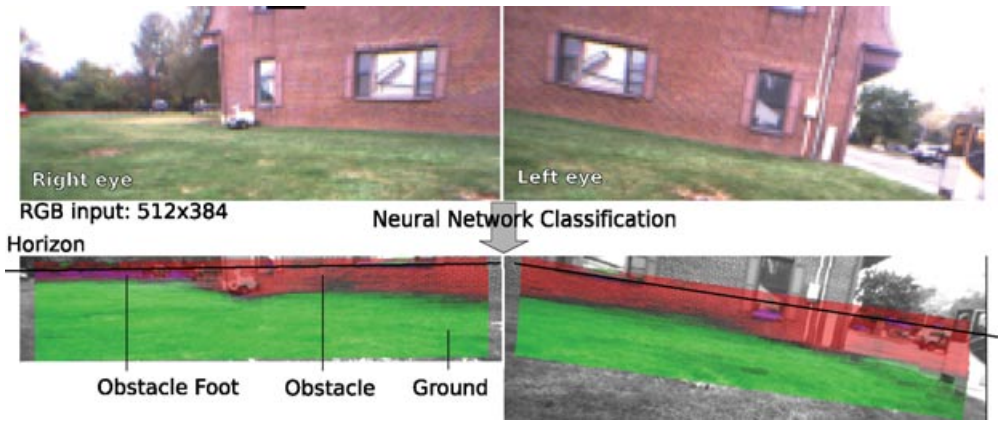


Figure 10. Plane extraction and traversability classification. Left and right images correspond to the left and right camera pairs or "eyes." A single ground plane is extracted from stereo points to generate different vision scales to the horizon, feeding those to the neural network. The resulting classification shows that the building and tree line on the left are classified as obstacle (overlaid red) as well as the parking lot on the right (mistakenly). To the contrary, the grass is accurately labeled as traversable (overlaid green). Note: the overlaid colors can be seen only in the color version.

4.3.2. Geometry Details

The h-polar space is not the exact image space given by the real cameras of the robot, but rather a pseudo-image space defined by the pseudo-camera parameters h_{cam} and $hR_{\min}$ (Figure 12) and able to integrate multiple frames together. As shown in Figure 12, the h-polar uses two different resolution distributions in the radial dimension:

- From the center of the map to the first hyperbolic cell, the resolution is *constant*; each cell has the same radial size.

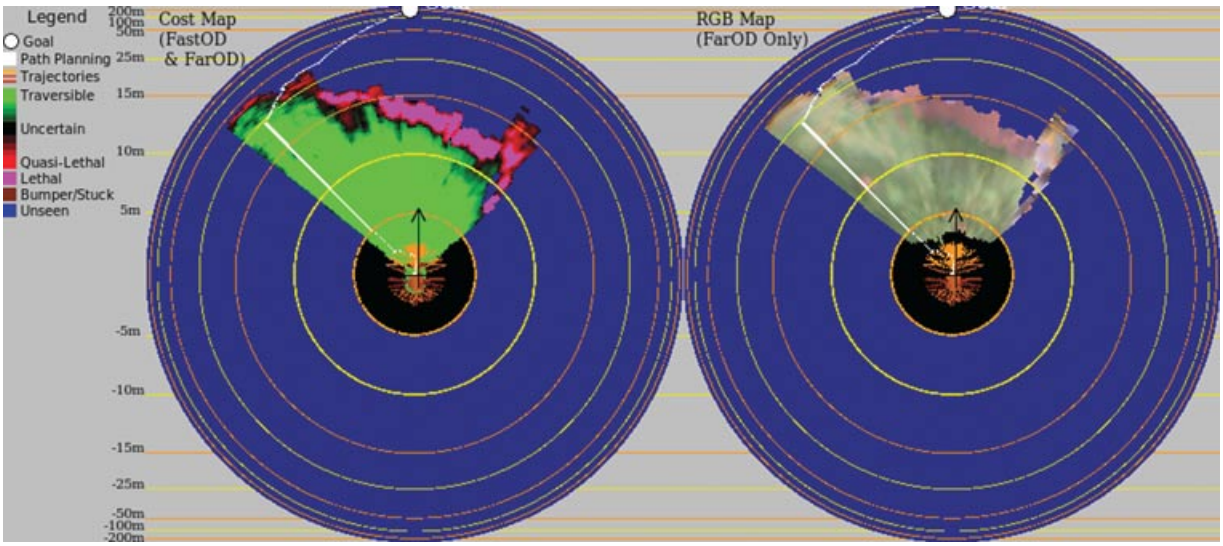


Figure 11. H-polar cost (left) and RGB (right) maps. Those maps are computed from the classification output shown in Figure 10. The optimal path (in white) to reach the goal 200 m ahead accurately leads around the building to the left. The RGB h-polar map (right) is used only for display and is very convenient for humans to figure out what the robot is looking at and how good the classification is. The vehicle is alway facing the top of the map. The orange curves in the 2.5-m radius represent the currently feasible trajectories.

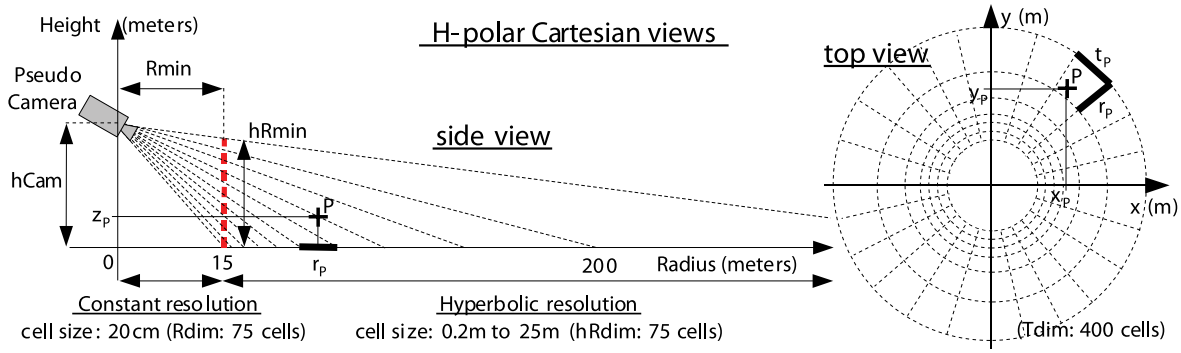


Figure 12. H-polar side and top views. The radial resolution is constant from 0 to 15 m with 20-cm radius. Then from 15 to 200 m and more, cell width ranges from 0.2 to 25 m. Integers r_P and t_P are the h-polar coordinates of the cell into which Cartesian point $P(x_P, y_P, z_P)$ falls.

- From the first hyperbolic cell to infinity, the radial cell size increases in a *hyperbolic* manner (equation of r_P is of type 1/radius).

The radius of the first hyperbolic cell is determined based on the desired cell size C_{res} in the constant-resolution area. In our system, we estimated that a cell size of 20 cm would be sufficient for our purpose. Thus to keep continuity in the cell sizes between the constant and the hyperbolic area, the first hyperbolic cell must begin at a radius $R_{\min}$ of 15.26 m, given the parameters h_{cam} and $hR_{\min}$ of our pseudo-camera and hypR_{dim}, the hyperbolic radial dimension. $R_{\min}$ is determined by the following equation:

$$R_{\min} = C_{\text{res}} \times \left(\frac{h_{\text{cam}} \times \text{hyp}R_{\text{dim}}}{hR_{\min}} - 1 \right). \qquad (1)$$

The following are the values used in our system in the field tests conducted by the LAGR Government Team in January 2008:

$C_{\text{res}} = 0.2$-m cell radial size in the constant area

$R_{\min} = 15.26$-m radius of the first hyperbolic cell

$h_{\text{cam}} = 1$-m height of the pseudo-camera

$hR_{\min} = 0.97$-m height of the pseudo-image at radius $R_{\min}$

$R_{\text{dim}} = 75$ number of cells in the constant area

hyp $R_{\text{dim}} = 75$ number of cells in the hyperbolic area

$T_{\text{dim}} = 400$ number of cells in the angular dimension

Given these parameters, Figure 13 plots the cell index versus the cell radius. Note that cells range up to 350 m. In practice, however, because uncertainty increases with distance and because only 150 cells are used in the radial dimension, the system is considered to be accurate up to 100 m. But there are no theoretical boundaries to this limit, which can be increased as the computation budget permits.

4.3.3. Cartesian to H-Polar Transformation (x,y,z) to (r,t,z)

Transformation formulas (for the hyperbolic area) from Cartesian coordinates (x_P, y_P, z_P) in the local pose coordinate system to h-polar coordinates (r_P, t_p, z_P) (z_P remains unchanged) of point P (Figure 12) are as follows:

$$r_P = \left(\frac{R_{\min} \times (z_P - h_{\text{cam}})}{\sqrt{{x_P}^2 + {y_P}^2}} + h_{\text{cam}} \right) \times \frac{\text{hyp}R_{\text{dim}}}{hR_{\min}} + R_{\text{dim}}, \qquad (2)$$

$$t_P = \arctan\left(\frac{y_P}{x_P} \right) \times \frac{T_{\text{dim}}}{2\pi}. \qquad (3)$$

These transformations are called many times. To remain efficient, their computation must be optimized: precomputing a lookup table of all the discrete T_{dim} useful values resulting from the arctan function provides a nonnegligible speed-up. Note also that only the integer part of (r_P, t_P) is used to determine which cell a point falls into.

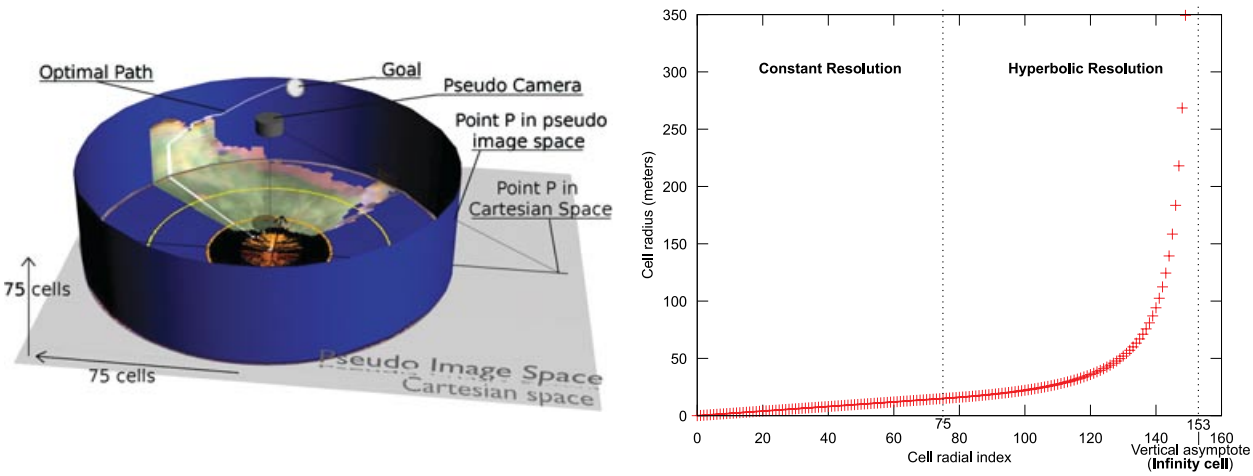


Figure 13. Left: RGB h-polar map in pseudo-image and Cartesian spaces. The center part of the h-polar map has a constant radial resolution, whereas the cylindrical part has a hyperbolic radial resolution. When flattened down into Cartesian space, the cylindrical part covers a bigger area in a hyperbolic manner. To transform coordinates between the pseudo-image space and the Cartesian space, a pixel is projected onto the Cartesian space along the line coming from the omniscient camera at the center of the map at height h_{cam}. The reverse transform projects a point from Cartesian space into the pseudo-image space. Right: h-polar cell radius versus cell radial index with current configuration. Given our pseudo-camera and h-polar parameters, the radial resolution is constant (20 cm) from cell 0 to 74 with radii ranging from 0 to 15.06 m. From cell 75 to 150, the resolution is hyperbolic and radiuses range from 15.26 to 350 m. Cells sizes range from 0.2 to 80 m. If a few more radial cells were added to our map, infinity would lie in cell 153.

4.3.4. H-Polar to Cartesian Transformation (*r,t,z*) to (*x,y,z*)

Transformation formulas (for the hyperbolic area) from h-polar (real) coordinates (r_P, t_p, z_P) of point P (Figure 12) to Cartesian coordinates (x_P, y_P, z_P) in the local pose coordinate system (z_P unchanged) are as follows:

$$\text{radius}_P = \frac{R_{\min} \times (z_P - h_{\text{cam}})}{(r_P - R_{\dim}) \times \frac{h R_{\min}}{\text{hyp} R_{\dim}} - h_{\text{cam}}}, \tag{4}$$

$$\alpha = t_P \times \frac{2\pi}{T_{\dim}}, \tag{5}$$

$$x_P = \cos\alpha \times \text{radius}_P \quad y_P = \sin\alpha \times \text{radius}_P. \tag{6}$$

4.3.5. Hyperbolic-Polar Organized Point-Cloud

Internally, h-polar is a point cloud, each point being defined by $(x, y, z,$ histogram, counter). For practical reasons, the number of points must be bound and it is desirable to keep a uniform distribution of points in the map. Therefore, points are kept using the h-polar distribution, i.e., one point per cell, in an array of points of dimensions $[(R_{\dim} + \text{hyp} R_{\dim}) \times T_{\dim} \times \text{Point}_{\dim}]$. The dimensions of this map do not have to be the same as those of the final cost map used for planning. In fact, there are four times more cells internally than in this final map. To avoid accumulation of integer imprecisions during recentering and frame additions, the exact (x, y, z) coordinates are kept at all times. Only the integer part of (r, t) is used to find the target cell of a point.

4.3.6. Translation and Rotation

The map is internally fixed onto the local pose coordinate system. It never needs to be rotated except for display. The rotation is extremely simple; a single (ring) offset t_{offset} is added to the angular index t before access. When recentering, no rotation is necessary; only translation is applied as described in the following section.

4.3.7. Recentering

Before adding a new frame to the current memory, the latter needs to be recentered to take into account the movement given by the latest available pose. As pointed out above, recentering of the h-polar map requires only translation of the (x, y) components of each point:

1. Compute translation $(x, y)_{\text{trans}}$ between the memory's pose and the last known pose in the

local pose coordinate system (z variations are ignored):

$$(x, y)_{\text{trans}} = (x, y)_{\text{memPose}} - (x, y)_{\text{lastPose}}. \quad (7)$$

2. For all $[(R_{\text{dim}} + \text{hyp}R_{\text{dim}}) \times T_{\text{dim}}]$ points:

   - Add $(x, y)_{\text{trans}}$ to (x, y) components of point.
   - Move point into new h-polar cell: if the new cell exists, merge point into that cell, else delete point.

After recentering, the current angular offset t_{offset} of the heading of the vehicle (α_{lastPose}) must be saved in order to render a map fixed to the vehicle coordinate system:

$$t_{\text{offset}} = \alpha_{\text{lastPose}} \times \frac{T_{\text{dim}}}{2\pi}. \quad (8)$$

Later, t_{offset} simply needs to be added to each t index before accessing the cells for render.

4.3.8. Adding a New Frame

A new frame comes as a cloud of (x, y, z, histogram) points centered on the vehicle's coordinate system at the time of the input image capture. Before adding new points to the memory (already recentered with the last known pose), translation and rotation need to be applied to take into account the last known position of the robot:

$$(x, y)_{\text{trans}} = (x, y)_{\text{imgPose}} - (x, y)_{\text{lastPose}}. \quad (9)$$

The rotation corresponds to the heading α_{imgPose} of the robot in the local pose coordinate system, given by imgPose, the pose of the image. Hence the following transformation matrix (z remains unchanged and can be ignored) produces the centered coordinates $(x, y)'$:

$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} \cos\alpha & -\sin\alpha & x_{\text{trans}} \\ \sin\alpha & \cos\alpha & y_{\text{trans}} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (10)$$

To add the recentered new frame to the recentered memory, one simply needs to loop through all points of the image point cloud and merge each into their corresponding h-polar cell.

4.3.9. Points Merging

When recentering memory or adding a new frame, several points may fall into a same h-polar cell and need to merge because only one point per cell is kept. There are two different merging operations to consider:

- **Data merging:** Using the histogram scheme, merging is a simple addition. Keeping a counter of the number of points merged can be useful to normalize the red–green–blue (RGB) display and for coordinates merging.
- **Coordinates merging:** The new coordinates can be an interpolation of each point's coordinates weighted by their counter, thus giving more importance to points seen many times. However, in our system, because of a low computation budget, coordinates are simply overwritten by the last added point.

4.3.10. Blocking after Obstacles

A key element to avoid projecting down to the ground the part of obstacles for which no z estimate is available, typically points beyond stereo range, is to block those points after an obstacle. For example, a tree trunk can appear as a giant obstacle in the map if not blocked after its foot. Therefore, all points without z information following an obstacle in the radial direction are ignored when adding a new frame.

4.3.11. Subsampled and Smoothed Mapping for Planning

The internal high-resolution h-polar map is scattered because points can merge during the translation process. This can produce erratic path planning between unknown cells and create holes inside lethal obstacles. To cope with this issue, a lower resolution map is produced for the actual planning: every low-resolution cell is a sum of four neighboring cells. In addition to subsampling, as some holes may subsist, the eight neighbors of each cells are added, thus providing a smooth dense map. Note that no normalization is required because the histogram scheme handles addition of multiple histograms by modulation of confidence.

4.3.12. Registration for Increased Accuracy

To improve mapping accuracy, registration methods can be used before merging new frames into the h-polar memory map. The pseudo-image space provides an easier and more efficient space for patch registration to correct for pose imprecisions. Registration

was not implemented in the current system but should be investigated in further research.

4.3.13. Grid-Based Planning with Precomputed Transition Costs

Any grid-based planning algorithm can be used here by using the precomputed transitions costs between the center of each cell to its eight neighbors. [The transition cost is simply the Euclidean distance of the (x, y) real coordinates of each cell's center.] As described earlier, the multilayer planning is used to guarantee latency independency between processes. Thus the hyperbolic-polar planning must generate all best paths from the goal to all cells in order to allow the lower planning level to operate on its own for few iterations at a higher frame rate. We use the Dijkstra algorithm from the goal to every cell of the h-polar map as our single source to multitargets planner. Every h-polar cell contains the optimal path to the goal. The reactive planner computes the cost from the vehicle center to a finite number of angular candidates at a certain radius (5 m) and then queries and adds the remaining cost for each of those candidates, beforehand translated into h-polar coordinates. The candidate with minimum cost contains the optimal path. Finally, the list of wheel commands of the trajectory that initiated the best path is executed until the next iteration. The optimal paths are drawn in white in Figures 14 and 15.

5. LAYER 1: FAST PERCEPTION AND PLANNING

5.1. Fast Stereo Vision

The short-range module (in Layer 1) follows a standard approach to vision-based obstacle detection (for similar examples, see Goldberg et al. (2002), Kelly and Stentz (1998), and Kriegman, Triendl, and Binford (1989). But it is also meant to process as fast as possible to guarantee a low-latency response to obstacles in the short range as part of the navigation architecture. Results in Tables I and II show that using a resolution as low as 160 × 120 for the stereo vision does not entail collisions with the LAGR vehicle. By decreasing the resolution, we also decrease the vision range (5 m) but increase reactivity. From those results we conclude that reactivity prevails over vision range in the short range.

Along with the low resolution, simple heuristics are used to produce traversability labels from the stereo three-dimensional (3D) points. A single ground plane fitted to the points is first estimated using a Hough transform as described in Hadsell et al. (2009). Points are then projected onto that ground plane and separated into the cells (10-cm squares) of the Cartesian map (10 × 10 m). The number of points falling into each cell is tallied, and a threshold determines whether a cell is an obstacle or traversable (unknown if empty). This small Cartesian map of 100 × 100 cells is then fed to the trajectory planning module described next. This small local map is located at Layer 1 but is also sent up to Layer 2 to be merged into memory in order to increase confidence in the h-polar map and also reduce disagreements between Layer 1 and 2 maps. If Layer 1 sees an obstacle that Layer 2 does not, the robot can go in loops forever in front of a cul-de-sac: Layer 2 sees an opening in the cul-de-sac and leads the robot into it, but getting closer Layer 1 realizes it is actually closed, ordering to go away from it. But going away from it, Layer 2 will bring the robot back when out of the Layer 1 range.

5.2. Fast Dynamics Planning: Recorded Maneuvers

An essential module for collision-free navigation is a fast reactive planner incorporating vehicle dynamics. (Inertia of the LAGR vehicle is measured in Figure 4.) The simple and computationally fast method described below is able to record sophisticated dynamics models while avoiding the complex and potentially inaccurate modeling of traditional methods.

Vehicle dynamics is typically handled by models whose parameters are found through system identification or manually computed from the vehicle's characteristics. Although these methods provide accurate theoretical dynamical models, they may not take into account differences between individual vehicles, lack adaptability to new environments, and may not handle sophisticated models, requiring handcrafted heuristics for backward motion, for example. We propose a simple and computationally efficient method that bypasses the need for system identification or handcrafting heuristics. Our system learns the particularities of individual vehicles and allows on-line adaptation and sophisticated models.

Human-driven or autonomously driven trajectories are recorded and stored. While in learning mode the robot records each traveled trajectory and attempts to place it into a bank indexed by the initial speeds of each left and right wheel and the ending position of the trajectory at a fixed radial distance from the current (starting) position. Only the best trajectories (mainly the fastest ones; other measures are described further) are stored in the trajectory bank. These are reused (essentially played back) during autonomous runs for precise vehicle control. The system knows the state in which it starts and has a recorded set of wheel commands from that state to any point on a circle of a chosen radius.

Optimal control techniques (Bryson & Ho, 1975) for motion planning provide optimal solutions

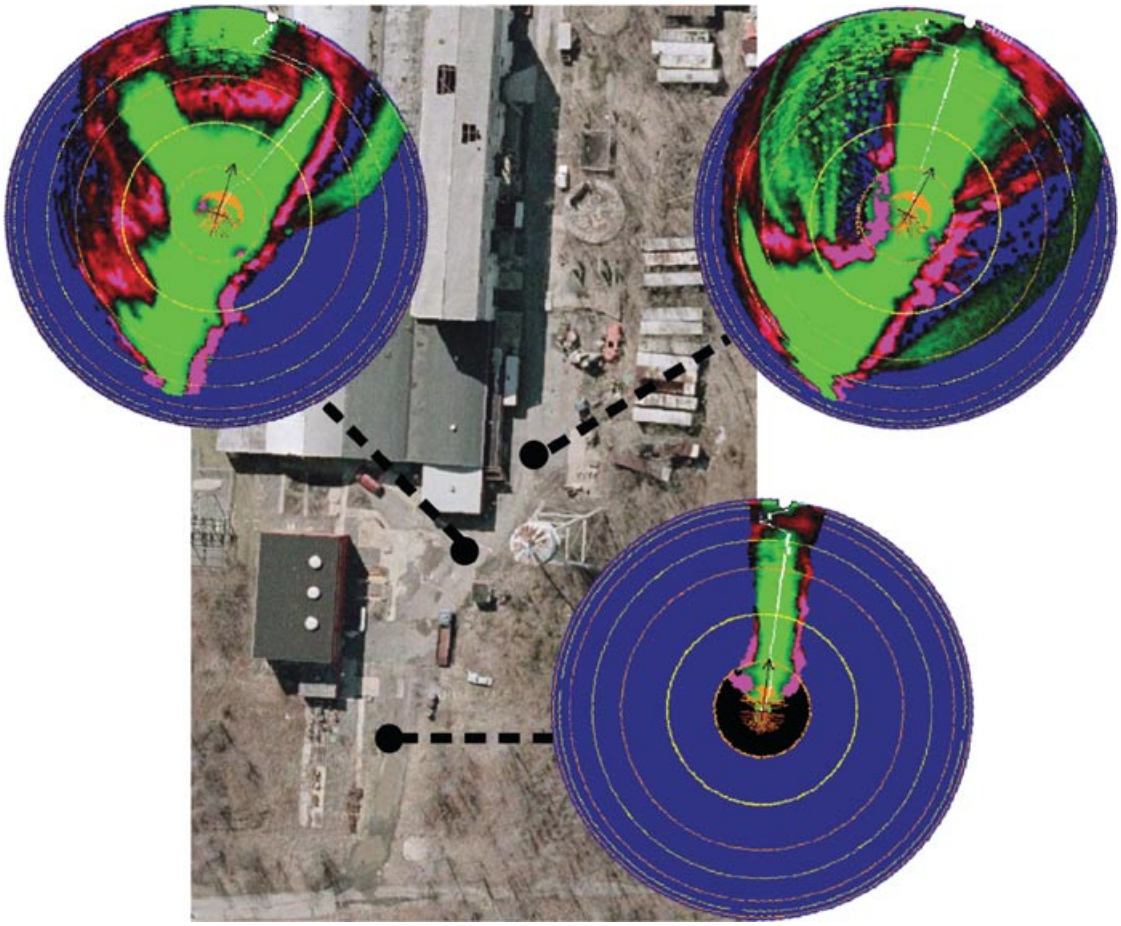


Figure 14. H-polar maps along heterogeneous outdoor course. The robot starts at the bottom of this satellite image going to the top. Each h-polar map accurately represents the real environment and plans around obstacles from an approximate distance of 50 m.

but suffer from high computational costs, making them impractical for real-time applications. Most efforts have concentrated on curve fitting for online planning from line segments (Tsumura, Fujiwara, Shirakawa, & Hashimoto, 1981), arcs (Komoriya, Tachi, & Tanie, 1984), clothoids (Kanayama & Miyake, 1985), or cubic spirals (Kanayama & Hartman, 1988). Curve fitting relies on identification of the parameters of a vehicle's dynamical model. Our method provides a simple and automatic identification of model parameters while bypassing the complex curve-fitting problem and resulting in a nearly null online computation cost. PiedMonte and Feron (1999) introduced maneuver-based planning in free environments for aerial vehicles by recording human expert maneuvers and later developed a maneuver-based automaton (Frazzoli, Dahleh, & Feron, 2005) that can concatenate trajectory primitives with a regular language. Whereas the concept of trajectory recording for maneuver-based planning is the same, we extended it to perform obstacle avoidance as well and apply it for ground robot nonholonomic planning. A similar precomputed dynamics planning architecture was introduced by Coombs, Murphy, Lacaze, and Legowik (2000) but using an offline clothoids precomputation method, which still relies on handcrafted modeling. Kelly and Nagy (2003) point out the need for a parametric trajectory representation in contrast to the prohibitively large space required by prestoring methods. The practical results of Coombs et al. (2000) and this paper advocate in favor of trajectory prestoring. Although recording and prestoring can certainly not address all nonholonomic planning problems, its simplicity and efficiency make it worth considering for some systems. Our recording method is currently limited to two-dimensional (2D)

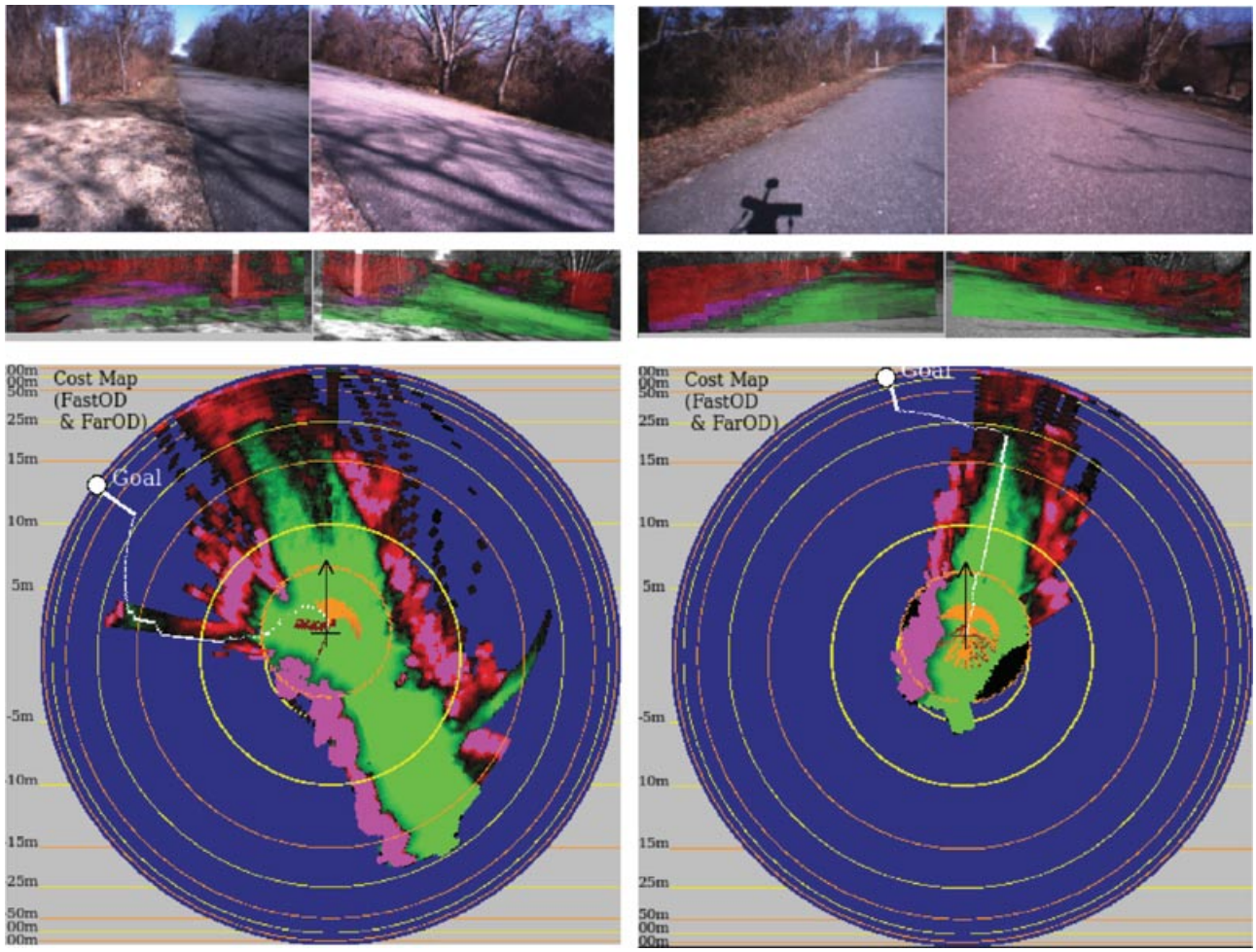


Figure 15. Result runs with long-range vision activated in Sandy Hook. The top images are left and right input images. The middle images show classification for left and right. At the bottom is the corresponding h-polar map. Note: the classification images in the middle for left and right examples were taken a few frames later than the input and the h-polar images.

planning and does not address the 3D challenges treated in Howard and Kelly (2005) and Spenko, Kuroda, Dubowsky, and Iagnemma (2006).

5.2.1. Recording Trajectories

Trajectories can be recorded every time the robot moves, either in supervised mode (human driver) or self-supervised (autonomous). On the LAGR platform, information about the state of the vehicle is available at 20 Hz. This state includes among other information an absolute time-stamp, the current pose, and current speeds of each wheel. Similarly, wheel commands are sent at 20 Hz to the motors in either manual or autonomous mode. These selected state variables along with the wheel commands are grouped into a *sample* every 50 ms (20 Hz). By recording this stream of samples, a feasible trajectory reaching the current position and the corresponding series of wheel commands are known at each time-stamp. In other words, the radius R_{traj} to the current vehicle center can be reached by looking back in the recorded sample stream and executing the wheel commands found in the stream. Every 50 ms a new trajectory is produced, and after some driving, enough trajectories are recorded to reach radius R_{traj} in all directions. The trajectory space has other input dimensions: the initial state (LSpeed_0, RSpeed_0). Composed of both wheel speeds obtained from the vehicle state, the initial state allows the system to select the current

set of feasible trajectories. When disregarding terrain differences (ice vs. asphalt), left and right initial wheel speeds are the main variables determining the current dynamics. For example, in Figure 6, initial state (4, 9) means that the robot is making a hard left turn at $t = 0$. By analyzing the trajectories going to the right, the figure makes clear that the dynamics of the vehicle have been captured. From this start state (turning hard left), in order for the vehicle to reach a candidate 90 deg to the right it must make a large loop where left-turning and forward-moving momentum is transformed into a right turn. This curve has not been computed but simply selected. It is the fastest set of commands seen so far for getting the vehicle from a hard left turn to a hard right.

Of course the accuracy of the trajectories depends on the accuracy of the pose sensor, which in the LAGR platform relies on a combination of the wheel odometers and an IMU. This pose information is assumed to stay accurate enough in the short term. If no visual odometry correction is available, it is preferable to record trajectories only on surfaces with low pose error rate (i.e., asphalt is preferable to ice). Once trajectories are extracted from the sample streams, they need to be sorted, compared, and stored into the trajectory bank (prestoring).

5.2.2. Trajectory Bank

The trajectory bank holds all best recorded trajectories. Each trajectory is indexed by (LSpeed_0, RSpeed_0, $\text{Angle}_{R_{\text{traj}}}$). LSpeed_0 and RSpeed_0 are the left and right speeds at time 0 of each trajectory. The LAGR vehicle drives at speeds ranging from -0.5 to 1.3 m/s. This range is quantized into 10 different bins (0 m/s lies in index 3). There are thus 100 different possible initial speed states. For each speed state, 160 $\text{Angle}_{R_{\text{traj}}}$ candidates are evenly divided around the perimeter at R_{traj} radius. In Figure 6, speed states along the diagonal have similar left and right wheel speeds at t_0, whereas at the top right and bottom left, the robot is steering harder right and left, respectively. Because very hard turns are less frequent, these corners are more sparse than the area around the diagonal. A minimal bank with only 15% of all states filled (extracted from approximately 2 h of human-driven recorded samples) showed great driving performance as demonstrated in the Trajectory Results section. The bank shown in Figure 7, which is 64% full and was obtained using 18 h of human-driven and autonomous-driven log files (recorded during regular testing runs) exhibits similar performances but can deal with a wider range of situations. The bank shown in Figure 6 is 64% full and was obtained using 18 h of human-driven and autonomous-driven log files (recorded during regular testing runs). The latter bank shows similar performance as in Table II but can deal with a wider range of situations.

The discretization of wheel speeds and angular candidates as well as the radial distance of the trajectories does require some tuning based on the maximum driving speed of the vehicle and the available computational budget. The higher the maximum speed of the vehicle, the more wheel speed bins are required and the more the trajectory radius must be increased. With more resolution (more wheel states) in each input dimension, modeling accuracy increases but so do computational requirements. Moreover, the more states in the bank, the more trajectories need to be recorded. It was empirically found that on the LAGR platform, a minimum of five states per wheel was sufficient to obtain decent driving. Ten states provided the best results while keeping a rather small bank. The required number of angular candidates is estimated from the local map's resolution. With a cell size of 10 cm and a radius of 2.5 m, approximately 160 candidates ($2\pi \times 25$) are needed to have one trajectory per cell on the perimeter. The 2.5-m radius was also chosen empirically and is based on the time the vehicle can travel before an update to the maps from the planner. Is it better to keep the radius as small as possible to limit the trajectory sampling space for memory and bank-filling matters.

Because in a race the vehicle is expected to minimize traveling time, the simplest scoring criterion for trajectory selection is the time it takes to reach the radius R_{traj}. When multiple trajectories are available for a same candidate, only the one with the lowest score is kept. Different scoring formulas yielding additional trajectory types for increased navigation performance are described in further sections.

5.2.3. Planning with Trajectories

The first step in planning is to determine the current speed state from the motor sensors, i.e., the speed of each wheel. This speed state indexes from the bank the set of currently feasible trajectories (based on what we have recorded). Next, the set of trajectories is tested against the current cost map as shown in Figure 6. All trajectories passing through non-traversable cells are ignored, and the remaining set of possible trajectories are assigned a $\text{cost}_{\text{traj}}$ based on the cost of the traversed cells in the map and the recorded traveling time of the trajectory $\text{time}_{\text{traj}}$. Cell costs represent traversability difficulty in seconds per meter; the minimum cost cost_{min} of perfectly traversable terrain is 0.77 s/m (vehicle's maximum speed is 1.3 m/s). Assuming trajectories are recorded on easy ground (e.g., asphalt), the minimum $\text{cost}_{\text{traj}}$

equals the sum of $\text{time}_{\text{traj}}$ and an "extra" cost given by the cells as follows:

$$\text{cost}_{\text{traj}} = \text{time}_{\text{traj}} + \sum_{i}^{n_{\text{cells}}} \left(\text{cost}_{\text{cell}_i} - \text{cost}_{\text{min}}\right). \quad (11)$$

This formula augments the real recorded time of the trajectory by adding extra time based on the estimated difficulty of traversing a cell whose terrain has been classified by the FarOD and FastOD map creation.

Trajectory costs are available on the 2.5-m radius around the robot. From those costs, costs of optimal paths to each cell on the 5-m radius are computed using the Dijkstra algorithm. The costs on the 5-m radius are added to the Layer-2 planning costs to form a global cost. The trajectory minimizing the global cost is selected and sent to the lower level planning execution process (Layer 0), thus protecting lower levels from latencies in more complex higher levels. The top planning level (Layer 2), running at 1 Hz using the Dijsktra algorithm, computes all optimal paths from the goal to each cell of the map and passes it on to Layer 1, which runs at 5–10 Hz. Computing paths to all cells of the map is necessary because the Layer-1 map moves relative to the Layer-2 map and thus uses different interfacing cells for every Layer-1 iteration. Once the best trajectory is selected, a list of wheel commands is sent to the Layer-0 process, which is responsible for sending wheel commands at 20 Hz. The resulting multilayer mapping and planning is represented in Figure 7.

5.2.4. Tail Whacking

To reduce computation, the width of the robot is taken into account by growing obstacles in the map by half the vehicle width. This simple method is fast but assumes that the length of the robot is null. Unaware of its length, the robot would often whack obstacles with its tail while turning. This issue is easily resolved by adding and keeping track of one or more points along the robot's length axis (Figure 16). A trajectory is discarded during planning if either the head or the tail trajectories encounter a nontraversable cell. With only one additional point near the tail of the LAGR robot, tail whacking completely disappeared during the numerous field tests.

5.2.5. Fail Safe

The vehicle may fall into a state that has no recorded trajectory if it reaches a rare state (very hard turns), if the bank is not full enough, or if the vehicle is completely boxed in by obstacles. A simple solution is to stop the robot if it is moving or turning to let it fall into another state where there will be a usable trajectory. If it reaches the null speeds state and still no trajectory is available then, a backup mode is triggered (scripted driving straight backward with a slight turn) to move the vehicle away from obstacles until feasible trajectories are available again. This simple solution has proved to be efficient as the vehicle is never stuck standing still or stuck in backup mode in any of the numerous field tests.

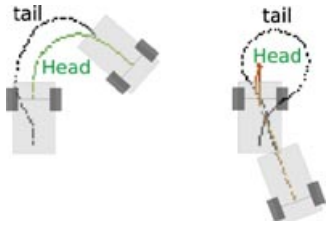


Figure 16. Tail whacking. To account for the vehicle's length and to avoid hitting obstacles with the tail, one additional "tail" trajectory is computed from each recorded "head" trajectory. (Width is handled by obstacle growing.) During planning, a trajectory is ignored if either the head or the tail of a trajectory encounters a lethal cell. Multiple tail trajectories can be added depending on the vehicle's length, but only one tail trajectory was necessary to get rid of the tail whacking issue on the LAGR vehicle.

5.2.6. Refinements

The simple recording solution described above required minimal engineering and tuning efforts while providing a computationally cheap and collision-free navigation platform. Following are a few additions to the system that brought further improvements. Other additions that increase the robot's self-adaptability to changing environments are described in the Future Work section.

When planning with obstacles, having only one trajectory per initial wheel state and angular candidate can be restrictive. A single obstacle close to the vehicle can wipe out all the trajectories in a general direction. To give more opportunities to the planner, it is desirable to store multiple trajectories, taking different paths to the same candidate. Storing multiple banks each with different criteria for selecting the trajectories, one can achieve this goal. For example, one might record trajectories that take as straight a path as possible to the candidate, as well as trajectories that steer to the left or right before taking the fastest path to the candidate. This allows the planner to pick a trajectory that goes around an obstacle on its way toward the optimal candidate for planning in the larger map.

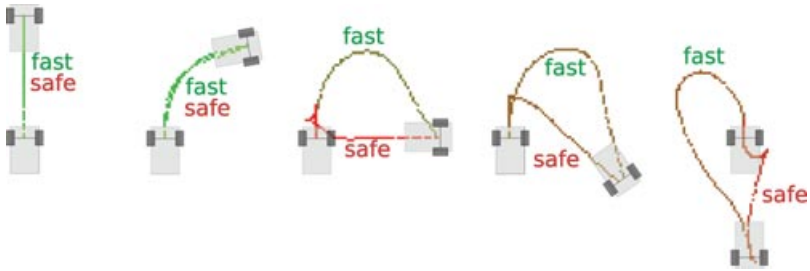


Figure 17. Fast and safe trajectories: A few examples of different trajectories obtained for a same end point using different scoring measures. The measure *fast*, which rewards fastest trajectories, selected smooth trajectories with large curves, maximizing the vehicle's speed. The measure *safe* rewards maximum initial alignment of the vehicle's heading and the candidate heading and thus selected trajectories that cause the vehicle to stop and turn in place before driving straight to the candidate. Having dissimilar trajectories for a same candidate helps the vehicle to properly handle different situations.

By sampling only those trajectories that drive fastest from where the vehicle is to the 160 angular candidates around it, we are selecting the behavior we want from the vehicle (to drive fast) but also greatly reducing the space of all possible vehicle movements (all sequences of wheel commands) and other desirable behaviors. By adding other selection criteria, we can more completely cover the space of possible movements and produce more complex behaviors. For example, in some situations, such as when the robot has made a wrong turn and gotten boxed in, it is desirable for the system to stop, turn in place, and go straight rather than keeping momentum by driving as fast as possible along smooth curves. These different types of behaviors can be obtained simply by using different scoring measures when selecting trajectories from the recorded samples for a bank.

Two types of trajectories were extensively tested; they are called *fast* and *safe* trajectories (Figure 17). The fast trajectories are selected by minimizing the time to reach the 2.5-m radius and maximizing the distance traveled in the first few samples as follows. For all trajectories filling a single slot in the bank $(\text{LSpeed}_0, \text{RSpeed}_0, \text{Angle}_{R_{\text{traj}}})$, retain the one that minimizes

$$\text{time}_{\text{traj}} - \sum_{k=0}^{n_{\text{cmds}}-1} |k, k-1| \times \text{decay}^k \times \gamma_{\text{dist}}, \qquad (12)$$

where n_{cmds} is the number of wheel commands of the trajectory and $|k, k-1|$ the Euclidean distance between two samples, measured by the vehicle's wheel encoders. γ_{dist} is the normalization term between the time measure and the distance measure. By using the exponentiation of *decay* (Figure 18), we ensure that only the distance at the beginning is maximized. Because the fast planner picks a new trajectory every 100–250 ms, in practice only the first few commands of a trajectory are actually driven; therefore it is important to have a term that forces the desired behavior to happen at the beginning of the trajectory; hence the exponential decay terms.

Safe trajectories, on the other hand, try to minimize the traveled distance and the angular difference between the vehicle's heading and the candidate's heading, thus forcing the vehicle to drive as closely as possible to the ray from the vehicle to the candidate. To achieve this, the scoring formula minimizes the traveled distance while maximizing the initial turning toward the target heading $\text{angle}_{\text{cand}}$. The resulting behavior is turn on a dime and go straight in the candidate's direction. For all trajectories reaching a single $(\text{LSpeed}_0, \text{RSpeed}_0, \text{Angle}_{R_{\text{traj}}})$, retain the one that minimizes

$$\sum_{k=0}^{n_{\text{cmds}}-1} |k, k-1| - |\text{angle}_{\text{cand}} - \text{angle}_k| \times \text{decay}^k \times \gamma_{\text{heading}}, \qquad (13)$$

where angle k is the heading of sample k. As in the fast trajectories, decay (Figure 18) helps maximize turning in the first few commands. γ_{heading} is the normalization term between the distance measure and the heading measure.

5.3. Fast Rotational Visual Odometry

The ability of a robot to localize itself can be critical for successful autonomous operation. Whereas a globally consistent solution to the localization problem must necessarily also perform mapping (Thrun, 2003), many applications do not require or benefit from a globally consistent map. Locally consistent

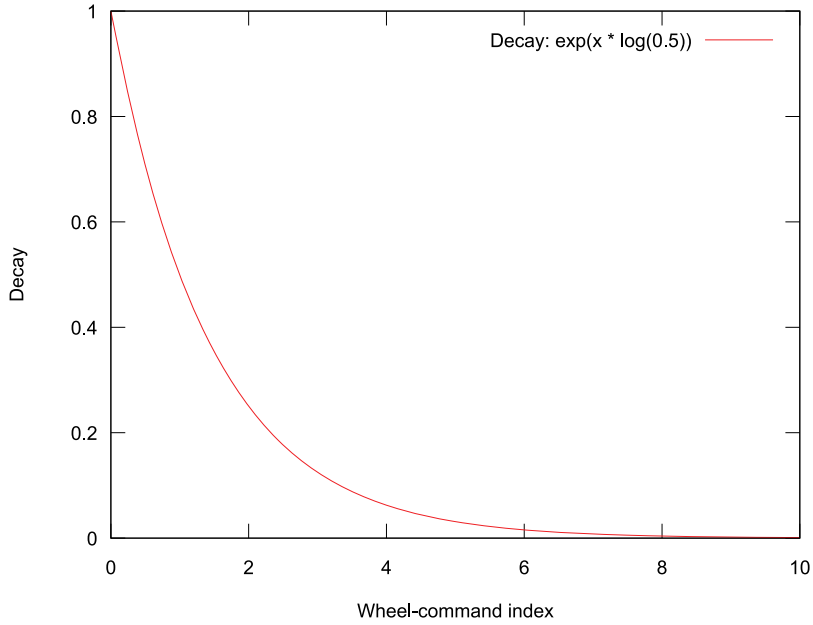


Figure 18. Trajectory type decays: A decay of 0.5 is exponentiated with the wheel-command index in order to give more importance to first samples and near no importance after five samples for the distance and heading measure of the fast and safe trajectories.

approaches such as fixed-time-window SLAM (Bibby & Reid, 2007) and visual odometry (Agrawal & Konolige, 2007; Nistér, Naroditsky, & Bergen, 2004) have shown great success in applications such as goal-directed navigation and localization in dynamic environments.

Wheel odometry has been a popular mainstay for robotic localization due to its low overhead and high sampling frequency. Its accuracy, however, is limited by wheel slip, a source of error that can be challenging to detect and correct without other sensors. Wheel slip can be particularly frequent and destructive in runs over outdoor terrain with loose or uneven ground. Full VO uses feature tracking to entirely replace ground odometry, but current systems (Agrawal & Konolige, 2007) require 100% of the processing time on a high-end CPU. For single-CPU autonomous robots, this cost can be prohibitive. The cost of VO can be a burden even for multi-CPU platforms, as it is often desirable for all camera–computer pairs to be able to perform additional tasks at high frame rates, such as short-range obstacle detection.

We have implemented a visual localization system that runs at a fraction of the cost of state-of-the-art VO systems while maintaining comparable accuracy. On our multiprocessor, multicamera system, this allows a single processor–camera pair to handle VO in parallel with other visual tasks, enabling tight coupling between localization, obstacle detection, and control. Our system can be of even more use to robots with limited computational power, such as small robots, consumer-oriented platforms, or extraterrestrial rovers.

We achieve this performance gain by specializing the visual odometer to the task of tracking only one degree of freedom (DOF): the robot's bearing. This bearing estimate is combined with a wheel odometer's translation estimate to yield three-DOF pose estimates with comparable accuracy to state-of-the-art six-DOF visual odometers. The efficiency of our system comes from the fact that tracking only the bearing allows us to operate at much lower resolutions than would be acceptable on a six-DOF odometer. This is because the uncertainty of an object's distance grows rapidly with distance at low resolutions, whereas the uncertainty of its robot-relative bearing is constant. For example, at our resolution of 160×120 pixels, an uncertainty of ± 0.25 pixels translates to ± 0.1 deg of yaw but ± 1.25 m of distance for an object 6 m away. Additional speedups are gained by using spherical image projection for more reliable feature tracking and limiting the feature tracking to windows bounded by wheel odometer output. We show that this hybrid odometry approach achieves much of the benefit of a full visual odometer at a greatly reduced computational cost.

5.3.1. Related Work

There has been much work in both wheel odometry and VO, but only limited attention has been paid to the intersection of the two approaches. Schäfer, Hahnfeld, and Berns (2007) use wheel odometry to check the output of a full VO system for errors arising from moving objects. Rather than run full visual and wheel odometry systems in parallel, we have focused on how to best exploit wheel odometry to lighten the computational burden of VO. A number of authors have used visual matching to correct IMU or GPS data on aerial platforms (Andersen and Taylor, 2007; Brown & Sullivan; 2002; Veth, Raquet, & Pachter, 2006). Our system is implemented on a ground rover, where many of the assumptions afforded in the air, such as nearly coplanar features and slow visual flow, do not apply. Veth and Raquet (2006) use IMU output to guide the feature search of a ground-based visual odometer, running on a manned sensor platform that takes pictures at 1 Hz. Many autonomous vehicles require a higher frame rate to ensure overlapping features between frames and to ensure that spurious poses from the IMU or wheel odometer are corrected before the robot controller reacts.

Most other work in relative pose tracking has focused either on using nonvisual sensors to detect wheel slip or on employing full VO as a complete replacement to wheel odometry.

5.3.2. Wheel Odometry Correction

Wheel slip can occur in many flavors, from sudden spurts of wheel speed to gently increasing drift. The latter in particular is difficult to detect by simple cross checking against motor current or inertial sensor output, requiring more nuanced approaches. Ojeda, Cruz, Reina, and Borenstein (2006) correct odometry using parameterized functions of motor current and soil cohesion. However, Maimone, Cheng, and Matthies (2007) report that such an approach fares poorly unless the soil consistency is nearly homogeneous. Ward and Iagnemma (2007a) train an SVM on hand-labeled odometric and inertial sensor outputs to detect immobilization. In another paper (Ward & Iagnemma, 2007b), the same authors employ a model-based approach, inferring robot velocity by fusing the output of a physical model with IMU, GPS, and wheel odometry output. A simpler approach to detecting slip would be to complement odometry with the absolute position measurements provided by GPS. Unfortunately, GPS input can be sporadic and inaccurate in wooded or urban environments (Hofmann-Wellenhof, Lichtenegger, & Collins, 2001; Sukkarieh, Nebot, & Durrant-Whyte, 1999) and even under optimal conditions refreshes only once per second. On our ground rover platform, we have found that 1 s is plenty of time for a robot to hallucinate a sharp turn due to wheel slip and react to it by sharply turning in the opposite direction. When driving alongside entangling vegetation or on narrow forest paths, such sudden "corrective" turns can prove catastrophic to a run. Furthermore, both GPS and odometry exhibit highly non-Gaussian error profiles, as neither GPS "jumps" nor wheel slip errors are well modeled as a random walk around a mean value. Sukkarieh et al. (1999) therefore use a chi-squared gating function to detect and discard blatantly spurious sensor outputs and feed only vetted sensor readings to a Gaussian model, in their case an extended Kalman filter (EKF) sensor fusion algorithm. However, this does not address the problem of GPS's low update frequency or a gating function's insensitivity to gradual odometry drift at low speeds. Rather than attempting to selectively detect and correct bad rotation estimates by the wheel odometer, we have opted to replace them entirely with the more reliable rotation estimates by our visual rotation tracker.

5.3.3. Full Visual Odometry

Full VO tracks visual features to make a differential estimate of the robot's full six-DOF pose. Problems that have been addressed in the past include include tracking and/or calibration from a small number of known landmarks (Triggs, 1999), performing triangulation with uncalibrated cameras (Stewenius, Nister, Kahl, & Schaffalitzky, 2005a), and tracking or calibrating with minimal information (Schaffalitzky, Zisserman, Hartley, & Torr, 2000; Stewenius, Schaffalitzky, & Nister, 2005b). Closer to our goals is the attempt to perform VO in real time in natural scenery. Nistér et al. (2004) tracked Harris corners (Harris & Stephens, 1988) in real time, discarding spurious feature associations between frames using RANSAC (Fischler & Bolles, 1981). Agrawal and Konolige (2007) improved upon this in their own real-time system by incorporating bundle adjustment to reduce drift. However, their system occupies all of the processor time on a high-end CPU, requiring additional computers to handle other aspects of autonomous operation such as mapping and planning. The NASA Mars Exploration Vehicle is hit particularly hard by the computational demands of full VO, which can take up to 3 min per frame on its 20-MHz processor, leading to an average movement of 10 m/h (Maimone et al., 2007). By contrast, our system is implemented on the same robotic platform used by Agrawal and Konolige (2007), where one of its two camera–computer pairs is dedicated to the task of real-time full VO. However, this leaves that camera-computer pair unable to perform other potentially critical tasks on its field of

view, such as obstacle detection. For this reason, we have chosen our more lightweight approach.

Not all existing approaches to visual pose estimation attempt to calculate the full six-DOF pose. Montiel and Davison (2006) demonstrated that EKF-based SLAM algorithms, although slow when overloaded with landmarks, serve well in the limited domain of "visual compassing." In this scenario, the camera rotates in place, estimating its own orientation and the bearing and elevation of landmarks, without need of estimating their position. On our nonstationary platform, tracking landmarks for more than a few frames would have required estimating their position. Moreover, maintaining estimates of visual landmarks over long periods of time (Clemente, Davison, Reid, Neira, & Tards, 2007) was of questionable utility in the LAGR contest, in which prior locations were seldom revisited. To avoid implementing such a full six-DOF mapping algorithm, we chose to discard landmarks after a few frames, adopting an approach closer to VO.

5.3.4. Algorithm

In the interest of keeping running costs down, our system tracks only six patches per frame, tracking a small number of wide-angle image patches at low resolution. The system samples patches from a region around the horizon, interpreting their horizontal motion from frame to frame as a rotation of the robot. There are two challenges to this approach: one is that a small number of features may be less robust to mismatches. The other is that a robot driving in a straight line will see features drift toward the sides of the image as it drives by ("drive-by drift"). Under a naïve implementation, such horizontal motion in the image would be incorrectly interpreted as a rotation. In this section we give a walk-through of our algorithm, paying particular attention to the solutions to the above problems. The overall algorithm is as follows:

1. Remap the image to remove feature size distortions due to planar projection.
2. Sample features from a small region of the previous frame selected using wheel odometry.
3. Search for the sampled features in the current frame, again limiting the search area using wheel odometry.
4. Cross validate the features' motions between frames and discard any outliers.
5. Localize the robot in the current frame by replacing the wheel odometry's rotation estimate with that of the visual odometer and rotating the translation estimate by the difference in rotations.

5.3.5. Remap Image to a Spherical Projection

Because we track a small number of image patches per frame, care must be taken to minimize the number of mismatched patches. We use wide patches subtending 8 deg of yaw, as larger patches tend to be more distinctive. However, such large features stretch when moved from the center of the image to the edges, where each pixel subtends a smaller solid angle. This distortion can cause mismatches when searching for features that have moved from the center of the image to near an edge, or vice-versa. To remove this distortion, we remap the camera image using a "spherical projection," where each pixel row and column subtends a fixed amount of vehicle-relative pitch and yaw $[\phi, \theta]$, respectively (Figure 19). We define the mapping from camera image coordinates $[i, j]$ to spherical image coordinates $[k, l]$ as

$$k(i, j) = [\phi(i, j) - \phi_o]/a, \tag{14}$$

$$l(i, j) = [\theta(i, j) - \theta_o]/a, \tag{15}$$

Here, a is a chosen ratio of radians per pixel and $[\phi_o, \theta_o]$ is the chosen pitch and yaw of the view ray corresponding to the upper-left pixel in the spherical image. We choose a as being the radians per pixel of the center pixel in the planar image. The center pixel subtends the largest angle of all the pixels, causing the spherically mapped image to be slightly smaller than the planar image. The spherical mapping is efficiently performed using a precalculated lookup table. To get the value of a pixel $[k, l]$ in the spherical image, one need only look up the corresponding camera pixel indices in the table and sample from that pixel in the camera image. We use the IPP (Stewart, 2004) function `Re-map`, which performs this mapping smoothly using bilinear interpolation. The $[i, j]$ corresponding to $[\phi, \theta]$ is calculated as

$$[i, j] = f(R_c R_\theta R_\phi \hat{\mathbf{z}}), \tag{16}$$

where $\hat{\mathbf{z}}$ is the unit vector in the forward direction in the vehicle's rotational coordinate frame, R_ϕ and R_θ are the rotation matrices corresponding to pitch and yaw, R_c is the rotation from the vehicle frame to the camera frame, and f is the projection function.

5.4. Sampling Features from Previous Frame

Even when the robot is moving straight forward, any feature except for those directly in front of the robot will drift toward the side of the image as the robot drives by. If we are to interpret the horizontal motion of features as robot rotation, we must limit our features to those whose "drive-by drift" between frames is less than half a pixel in the spherical image and

(a) Camera image (b) Spherical image

Figure 19. A rectified camera image and its spherical transform. The asymmetry of the spherical image is due to the fact that the camera is pointed off to the left and down, with some axial roll. The robot is pointed straight at the area highlighted by four yellow dots, placed at pitch and yaw values [0.03, 0.1], [0.03, −0.1], [−0.03, −0.1], and [−0.03, 0.1], in radians. After the transformation, these points form an axis-aligned rectangle around the frontal direction. In practice, we transform only the portion of the camera image roughly located around the horizon, highlighted by the green rectangle above.

therefore undetectable. As shown in Figure 20, this constraint defines a relation between the feature's distance r, bearing θ, the maximum possible travel of the camera between frames d, and the yaw subtended by a pixel in the spherical image θ_p:

$$r > \frac{d \tan(\theta_p/2 + \theta)}{\cos\theta \tan(\theta_p/2 + \theta) - \sin\theta}. \quad (17)$$

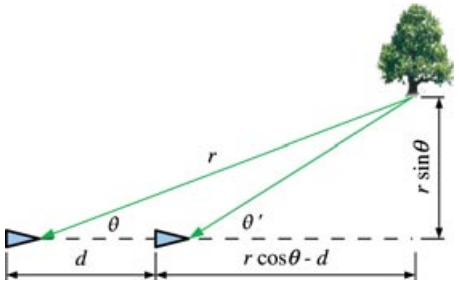


Figure 20. Parallax from pure forward motion. We wish to limit our attention to objects that will not shift under parallax from frame to frame. In a spherical image where each pixel subtends θ_p radians of yaw, this requires that the bearing θ of the object must not change more than $\theta_p/2$. Given upper limits on θ and the frame-to-frame translation d, we may solve for a minimum distance r. We avoid parallax by detecting distance using stereo data calculated earlier for obstacle detection and ignoring all features closer than r.

With a sufficiently reliable stereo vision system with which to estimate r, one could apply the above criteria to all features in the image, sampling image patches only around those features satisfying relation (17). However, stereo can be unreliable for large r, particularly at low image resolutions. Instead, we opt to limit our features to a small range of yaw $-\theta_{\max} < \theta < \theta_{\max}$ centered on the frontal direction $\theta = 0$. We then plug $\theta_{\max}$ into relation (17) to derive a corresponding minimum distance $r_{\min}$. Any feature closer than $r_{\min}$ is deemed unfit for use. As discussed in Section 5.4.2, when an insufficient number of viable landmarks are found in a particular frame, the pose for that frame is estimated using wheel odometry. In practice, this happens rarely outdoors. Even in dense forests such as the one in Figure 21(c), hybrid VO "punted" on only 92.1% of the frames. We estimate distance using the dense stereo image already calculated by another component for the purposes of obstacle detection. If no such calculation is already being done, calculating stereo disparities over the entire image is an inefficient means of estimating feature distances. In this case, the approach of Agrawal and Konolige (2007) may be used, where stereo disparity is calculated for each patch rather than for each pixel.

The choice of $\theta_{\max}$ can be made based on the robot's expected speed, environment, and camera geometry. On our platform, the cameras point off to the sides, thereby making the frontal direction close to one side of the spherical image [see Figure 19(b)].

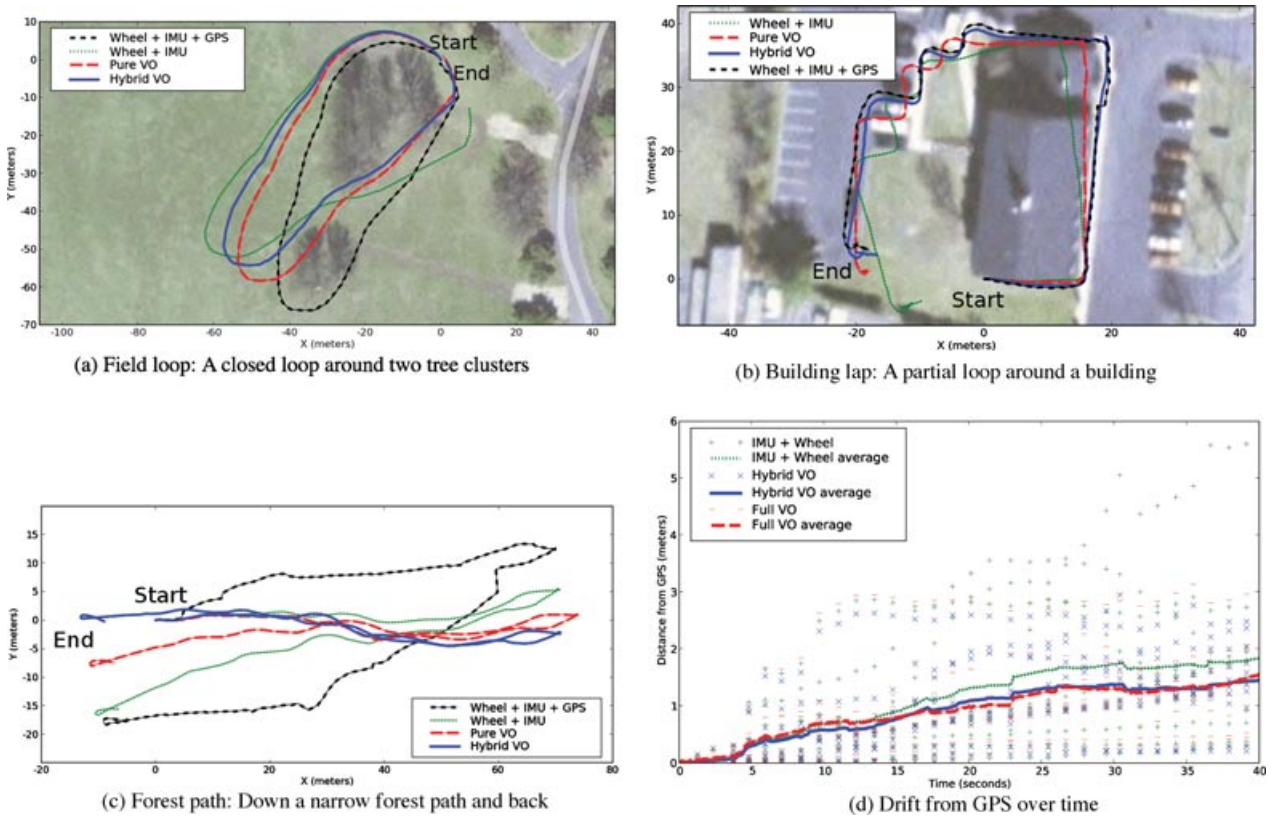


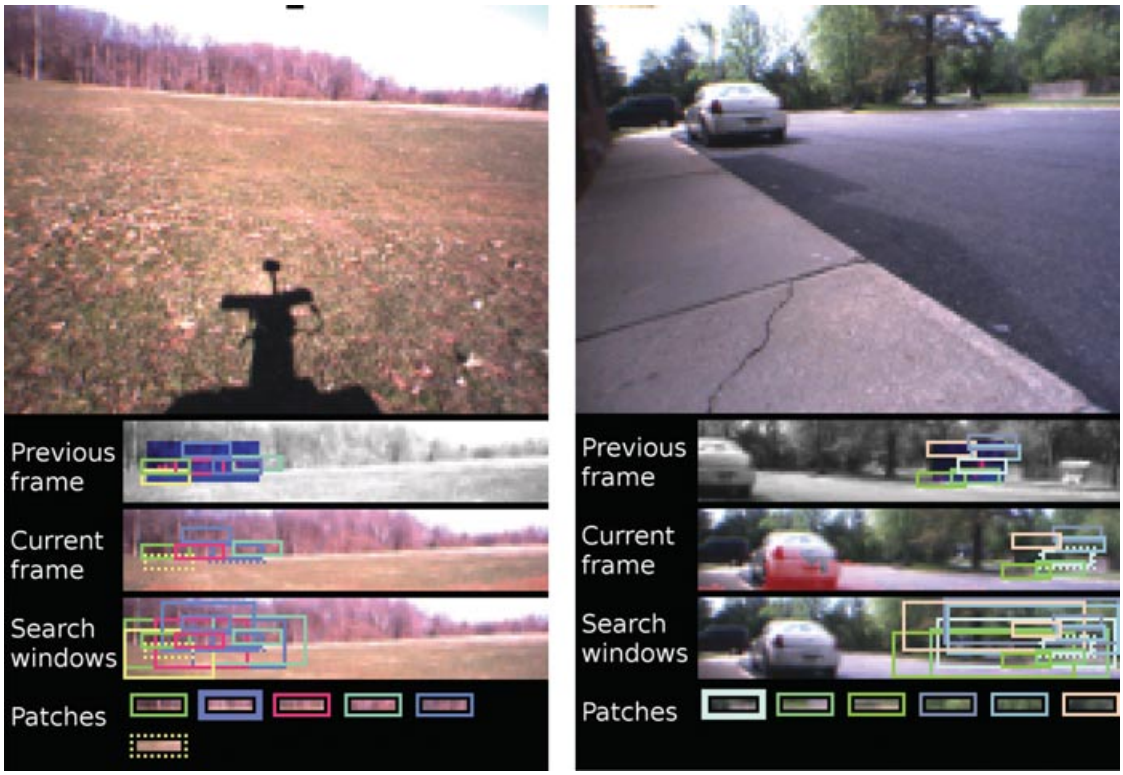
Figure 21. Vehicle trajectories, as measured using wheel odometry + IMU, GPS + wheel odometry + IMU, full VO (Agrawal & Konolige, 2007), and vision-corrected wheel odometry. In panel (a), the robot's initial pose and final pose are identical. The size of the opening in the loop can therefore be used as an indication of the correctness of the trajectories' shape. The non-GPS trajectories are globally oriented using the initial GPS orientation estimate, which can be noisy. In panel (b) the robot follows the sidewalk between the robot's first turn and sixth turn. The sidewalk's shape serves as ground truth during this segment. A typical GPS jump can be observed at the right side of the figure. In panel (c), the robot traverses a narrow forest path and then backtracks down the same path. GPS is inaccurate and disruptive in wooded areas such as these. Panel (d) shows the drift from GPS over time. The dots show the distances of estimated trajectories from the GPS position for 10 randomly chosen runs. The lines show the corresponding averages over all runs. The average distance traveled (as measured by GPS) was 30.3 m.

We therefore chose $\theta_{\max}$ to be the absolute value of the yaw of that side, namely ± 0.09 or ± 0.14 radians, depending on the eye. This left 15%–23% of the image available for sampling. Using these $\theta_{\max}$ values, along with a d of 0.13 m (1.3 m/s/10 Hz), relation (17) yields minimum distances $r_{\min}$ of 2.42 and 3.69 m. Note that d is the maximum expected travel of the camera, which is not necessarily the maximum expected travel of the robot. On our robot the geometry and kinematics make the maximum vehicle travel a reasonable approximation to the maximum camera travel. Other platforms may require a different estimate of d.

The region defined by $-\theta_{\max} < \theta < \theta_{\max}$ is further shrunk on all sides by half the patch dimensions before applying a Harris corner detector to one of its channels. These corners are used to find suitable points to sample RGB patches from. The horizon images labeled "previous frame" in Figure 22 show this region tinted in dark blue. As each image patch is selected, the Harris corners under that patch are set to zero as a form of nonmax suppression. We sample six patches measuring 13×3 pixels, or 7.6 by 1.75 deg of solid angle or 11% by 3% of the field of view.

5.4.1. Searching for Patches

In outdoor environments, it is a common occurrence for the robot to move less than reported by wheel



(a) Open field (b) Office parking lot

Figure 22. The robot's view, while running two of the courses in Figure 21. In panel (a), at top is the current camera image, at the low resolution of 160 × 120. Below this is the spherical horizon image of the previous frame. Harris corners are detected in a small window shown tinted in dark blue, with red pixels indicating Harris corner strength. This window is chosen either as the frontal direction (a) or the region that will be the frontal direction in the current frame, according to wheel odometry (b). Image patches (shown as small rectangles) are sampled around the strongest corners. The next image below shows the horizon of the current frame, with the image patches from the previous frame found using normalized cross correlation (NCC). Each patch's NCC search is run in a window that spans the patch's position in the previous frame and the patch's predicted position in the current frame, as predicted by wheel odometry. These search windows are shown in the next image below, as rectangles surrounding each image patch. In (b) these search windows are wider because the robot is in the middle of a sharp turn. At the bottom are the isolated image patches. Panel (a) shows five good patches and one outlier patch (dotted outline) that was rejected for disagreeing with the other patches about how much the robot rotated. Comparing the yellow rectangle in the previous frame and the current frame, we can see that it has shifted by one pixel relative to the other patches.

odometry ("wheel slip"). The opposite case of the robot moving significantly more than the wheels ("wheel skid") is far less common under vehicle speeds typically operated under by autonomous vehicles (Ward & Iagnemma, 2007b). We therefore trust our odometry to set an upper limit on the expected patch motion from frame to frame. When searching for image patches, we limit our search window to a region that encompasses the patch's position in the previous frame and the patch's position in the current frame, as predicted by wheel odometry. This window is inflated on all sides by half the patch dimensions for an added measure of safety. We apply a normalized cross correlation of the image patch with this search window and choose the maximum as the best-matching location.

5.4.2. Cross-Validate Matches

The change in yaw of an image patch from the previous frame to the current frame is that patch's estimate of the robot's rotation. To detect outliers, we cross validate by having each patch "vote" for every other patch whose estimate differs by less than θ_p.

Patches whose vote tally is more than half the number of patches are deemed reliable, and others are rejected as outliers. The rotation estimate shared by all inliers is taken as the robot's rotation between the previous and current frames. When using a small number of patches, a pair of frames may occasionally present no patches with a sufficient number of votes. On such frames we let the wheel odometer supply the change in pose.

5.4.3. Localizing the Robot

The robot's current pose is estimated as

$$p' = p + R_{\theta_v/2} R_{-\theta_w} \Delta p_w, \tag{18}$$

where p' and p are the current and previous pose estimates, Δp_w is the change in pose as reported by wheel odometry, θ_v and θ_w are the change in yaw as reported by VO and wheel odometry, and R_{angle} is a rotation matrix representing a yaw rotation by angle. The concatenation $R_{\theta_v/2} R_{-\theta_w}$ expresses the fact that we undo the odometry-reported rotation θ_w and replace it with $\theta_v/2$. We use the midpoint method to numerically integrate the pose forward; hence the use of $\theta_v/2$ rather than θ_v.

5.4.4. Implementation

The visual odometer is implemented in Lush compiled to C. We captured the camera images at a low resolution of 160×120 pixels and used six image patches of 13×3 pixels. The Intel Performance Primitives (IPP) library (Stewart, 2004) was used for the spherical image transform, Harris corner detection, and normalized cross correlation. The hybrid VO system runs within the same thread as the short-range stereo-based obstacle detector (Sermanet, Ben, Naz, Beat, & Lecun, 2007), running at 5–10 Hz as shown in Figure 2. The processor time is also shared by the long-range vision module running in a separate thread at 1 Hz.

6. RESULTS

6.1. Architecture Results

The mapping and planning algorithms used in this section are not the final ones described further, but the results described here are similar in the final system. Those intermediate algorithms were described in previous papers (Hadsell et al., 2007).

6.1.1. Timing Results

Figure 23 plots the latencies and frequencies of an actual run of the robot for different layers of the system. In the top plot, the bottom curves indicate a camera and API latency of 190 ms, i.e., the age of the data when they are available to our software on the eye machines. On top is the actuation latency of 250 ms approximately, i.e., the age of the same data when they reach the motors. The goal of Fast-OD or Layer 1 is to minimize this processing time between the sensors and the actuators in order to maximize reactivity. Similarly, an actuation latency of 700 ms was recorded for the Far-OD (Layer 2). Finally, Fast-OD exhibited a frequency of 10 Hz against 2–3 Hz for Far-OD. To conclude, those timing measures demonstrate Fast-OD's success to guarantee a low reactivity latency and a high frequency while giving Far-OD the possibility to run at a slower pace. The corresponding field performance is tested in the following experiment. Note again that those timing results were gathered during the middle of the LAGR program; as Layers 1 and 2 increased in complexity, frequencies dropped to 5–10 Hz for Layer 1 and 1 Hz for Layer 2 but remained effective.

6.1.2. Field Results

To show the benefits of our multilayer architecture, the following test comparing different combinations of Fast-OD and Far-OD was designed and is depicted in Figure 24. Note, however, that a 320×240 stereo module was used as Far-OD in lieu of the neural network module because the latter was not mature enough at the time. First, to test reactivity, the robot starts a short distance from an obstacle and facing 90 deg away from it. The goal is also 90 deg away, so that the robot will turn right into the obstacle and avoid it or not, depending on its reactivity. Second, to test long-range vision, a cul-de-sac of hay bales deeper than the Fast-OD radius is placed after the first obstacle and on the straight line to the goal. This way, a robot with no long-range or low-frequency long-range vision will enter the cul-de-sac. To reach the goal the fastest in this test, a system has to combine fast reactivity and long-range vision.

Three different modes are tested: Fast-OD only, i.e., fast reactivity but no long-range vision; then Fast-OD and Far-OD in parallel, i.e., moderate reactivity and moderate long-range reactivity; and finally Fast-OD and Far-OD in series, i.e., poor reactivity but good long-range reactivity. Each configuration was run five times, and an average was taken for each metric and configuration.

Results are reported in Table I. Configuration (1) had the highest frequency and lowest latency and never crashed but consistently ended up in the

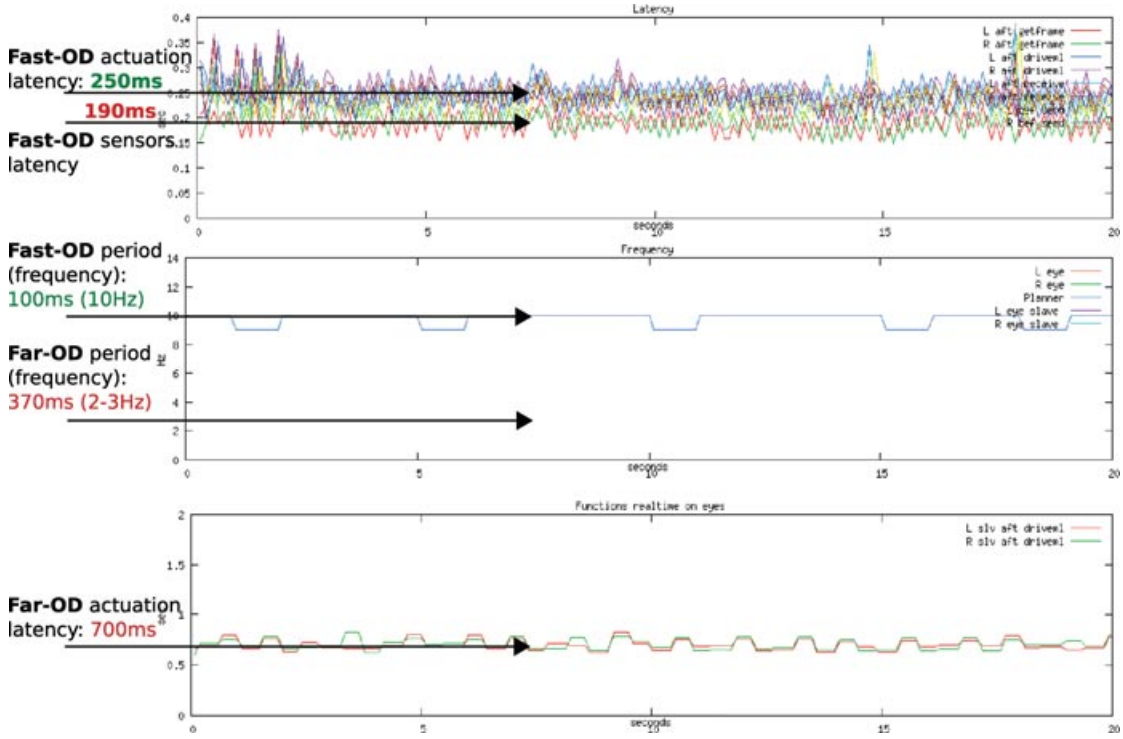


Figure 23. Internal timing plots. The top plot shows the age of the data at different processing stages. The data are already 190 ms old when received by the API, but only 250 ms old when delivered to the actuators by Fast-OD, whereas it is 700 ms old when delivered by Far-OD to the actuators (bottom plot). The middle plot exhibits the Fast-OD and Far-OD frequencies of 10 and 2–3 Hz. (The 2–3-Hz Far-OD frequency plot is not available in this particular figure but was observed in similar plots at the same period.)

cul-de-sac. Configuration (2) had low frequencies and good latency and crashed once into an obstacle. This crash probably happened because Fast-OD was running at a lower frequency compared to configuration (1). It entered the cul-de-sac once, most likely because of Far-OD's lower frequency. This shows the limit of the low frequencies of Fast-OD and Far-OD and that a trade-off must be found. Configuration (2) reached the goal the fastest by having fewer crashes and going less frequently into the cul-de-sac. Configuration (3) had a moderate frequency and low latency, and many crashes were observed. However, it consistently avoided the cul-de-sac. Note that latency is determined by sensor-to-actuator time and is not necessarily a factor of frequency; for example, on the LAGR platform, the eye machines can run at 4 Hz but frame latency can be more than 250 ms because it includes network time between eye and planner machines as well as planning time. Thus using Far-OD only (about 4 Hz) would yield good frequency but poor latency, resulting in poor performance; however, Fast-OD at 4 Hz does maintain good latency: latency and frequency have to be maintained together.

Table I indeed shows that Fast-OD alone performed very well in reactivity but not in long-range vision and Fast-OD+Far-OD in series performed well

Table I. Average run results. The parallel system performed the fastest, with a nearly null number of collisions and entrances into the cul-de-sac.

Test mode	Runs	Frequency	Latency (ms)	Total time (s)	Crashes/ run	Into cul-de-sac/run
(1) Fast-OD only	5	Fast-OD: 10 Hz	280	26.2	0	1
(2) Fast-OD+Far-OD in parallel	5	Fast-OD: 4–5 Hz, Far-OD: 2Hz	300	19	0.2	0.2
(3) Fast-OD+Far-OD in series	5	Fast-OD+Far-OD: 3 Hz	650	32.1	1.4	0

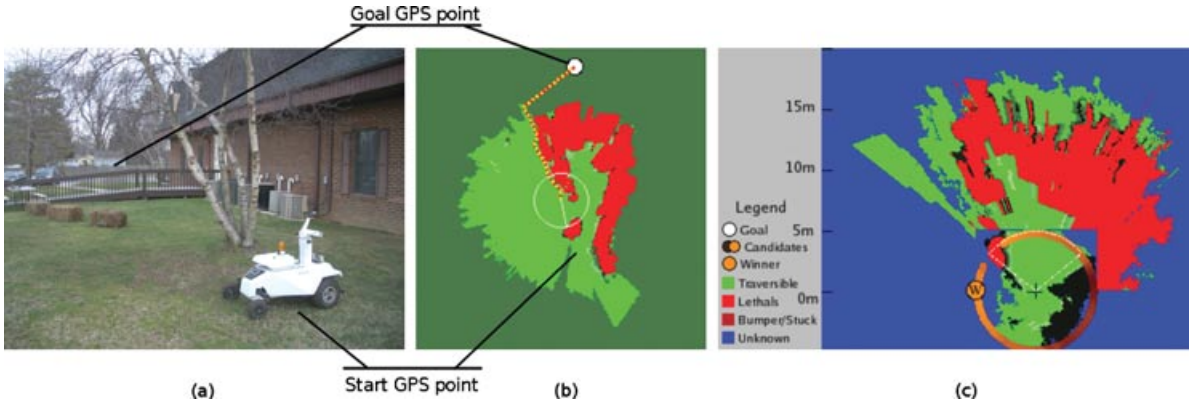


Figure 24. Test description. (a) Site of the field test. (b) Global map of the test produced by the robot. (c) Vehicle map produced by the robot at the entrance of the cul-de-sac. The vehicle map combines Fast-OD and Far-OD maps; thus we can see that thanks to the Far-OD, the winning candidate "W" of the Fast-OD map is chosen to avoid the cul-de-sac.

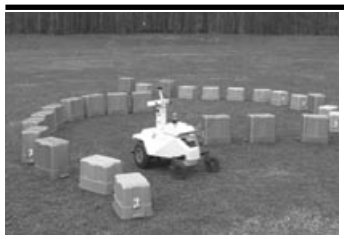
in long range but not in reactivity. The Fast-OD+ Far-OD in parallel combination did not individually perform as well on each task but overall provided the best compromise between them.

Note that a reasonable balance between each module must be found to ensure good results. One cannot, for example, let the Far-OD control-loop latency and period be too high to keep good long-range reactivity. Depending on the speed of the vehicle, processing time, and maximum distance of each vision module, a trade-off must be found. One could also use more than two layers of perception and planning modules. For example, a midrange vision module with a slow stereo system could be placed between the Far-OD and the Fast-OD, thus allowing even more computation time for Far-OD.

6.2. Trajectory Results

We compare our system of learned vehicle dynamics to our previous system, which uses a function of the steering angle to determine which wheel commands to apply. The vision, localization, mapping, and planning of the two systems are identical; only the low-level driving commands (Layer 1) were changed. Our previous system performed adequately and was rated the first best performer in independent government tests at the end of Phase I of the LAGR project. But, the lack of precise vehicle dynamics caused collisions, which are completely avoided with the new system. Both systems were run one to three times over five different 30-m courses (Table II), each consisting of a different arrangement of buckets as obstacles, with gradually increasing difficulty. Running times and the number of bumper hits were recorded and averaged in Table II. As the difficulty increases, using the old system (called "steering"), the running time and number of hits increase dramatically. The most difficult course could not be tested because the system would systematically destroy the bucket arrangement. The new system with learned trajectories, on the contrary, keeps a low running time over all

Table II. Comparison of old steering-angle system (red) versus new learned-trajectories system (green). On the left is a picture of the "slalom" course. On the right, running times and number of collisions are reported. The new system with learned trajectories maintains a low running time over all courses and a total of 0 collisions. The trajectory bank used in this experiment was only 15% full and was extracted from 2 h of human recorded runs.



	Running time		Obstacle hits		Number of runs	
Course	Steering	Trajectories	Steering	Trajectories	Steering	Trajectories
1 wall (easy)	19 s	16 s	0	0	1	1
2 walls (easy)	20 s	15 s	0	0	1	1
3 walls (moderate)	36.5 s	18 s	3	0	2	1
Slalom (hard)	56 s	26.33 s	11	0	1	3
Scattered (hardest)	X	24	X	0	0	1

courses and a total of 0 collisions. The driving capabilities of the system can be seen in a video (Sermanet et al., 2008).

6.3. Visual Odometry Results

The hybrid VO system has been tested on various outdoor terrains, including the area around an office building, an open field, and a narrow path through a forest. For the results presented here, we recorded logs in these settings and ran both our hybrid VO and the full six-DOF VO of Agrawal and Konolige (2007) on them, as a benchmark for accuracy and processor time. Figure 21 shows the pose trajectories of three runs. Predictably, wheel odometry fused with IMU consistently fares the worst, accumulating small rotational errors that affect the entire trajectory that follows. The EKF fusion of wheel odometry, IMU, and GPS does better, except in the forest, where the GPS signal can be both sporadic and inaccurate. Even with clear GPS reception as in Figure 21(b), the GPS-aided trajectory features a discontinuity at the top of the image due to satellites coming into and out of reception.

In Figure 21(b), the robot was driven around a sidewalk. At the end of the run, wheel spin was deliberately induced by attempting to drive up an intraversable curb. Naturally, all trajectories other than full VO end with a spurious short straight segment that represents the resulting hallucinated forward motion. In practice, we found such sharp discontinuities rare. Furthermore, devoting a CPU to full VO on our platform would have prevented one of the two stereo camera pairs from performing obstacle detection on its field of view. The plot in Figure 24(c) shows the robot going down a narrow path through a forest. Accurate, high-frequency (i.e., inexpensive) estimates of the robot bearing are particularly important in such settings, where false rotations due to wheel slip are frequent and can cause a robot to counter-steer into entangling obstacles on each side. The hybrid VO retraces the path accurately after a quick 180-deg turn.

The processor times per frame for each of these runs are reported in Table III. The run times shown are those of a Pentium 4M laptop at 2 GHz, running off of image and sensor data logged by the robot. The robot's CPUs are approximately 2.5 times faster. Although our system does not track translations, it does use range information to rule out features that are too close to the robot. As discussed in Section 5.4, we get this information "for free" by appropriating the stereo image already calculated by our obstacle classification system. If no such stereo image data are available, we can also adopt the approach of Agrawal and Konolige (2007), where a patch is searched for in the other stereo camera to estimate distance on a per-patch rather than per-pixel basis. In Table III, the run times for running just the hybrid VO and for running the hybrid VO with per-patch stereo matching are shown separately. The full VO run time is best compared to the latter.

Table III. CPU time per frame for full VO (Agrawal & Konolige, 2007) and hybrid VO on the three courses shown in Figure 21. "Hybrid VO + range" shows the total time for calculating both stereo information and the pose estimate. "Hybrid VO" shows the time for the pose estimate only. Patch-based stereo, as performed by the full VO implementation, would result in a time between these two.

Log file	Full VO (ms)	Hybrid VO + range (ms)	Hybrid VO (ms)
Field loop	266	11.5	8.0
Building lap	238	11.0	8.2
Forest path	153	14.0	9.3

6.4. Complete System Results

We present experimental results obtained by running the robot on outdoor courses with the long-range vision turned on and off and using the Baseline system. The goal is to show the benefits of each system against the other in end-to-end runs. The three navigation systems compared the following:

- Long range: our complete system with long-range vision enabled (100 + m).
- Short range: same system as long range but long-range vision is disabled, relying on the 5-m range reactive module.
- Baseline: the LAGR native system used as the 2004 state-of-the-art reference by the Government team, with approximately 15 m of vision range.

It is unfortunately not trivial to set up systematic field tests to gather statistically significant evidence because of the debugging-oriented character of most of the test runs during the development phase and limited time in a tight development and testing schedule. Also in general many little things make systematic testing harder, such as limited robot and laptop battery power (recharging takes hours), hardware issues such as booting failures, logging disks failures, and software logging failures, and the human factor during winter cold and summer heat. Therefore the complete runs results we present count only handfuls of runs but are, however, representative of our experience with development experiments toward the end of the development phase. Results are reported for two outdoor locations in New Jersey.

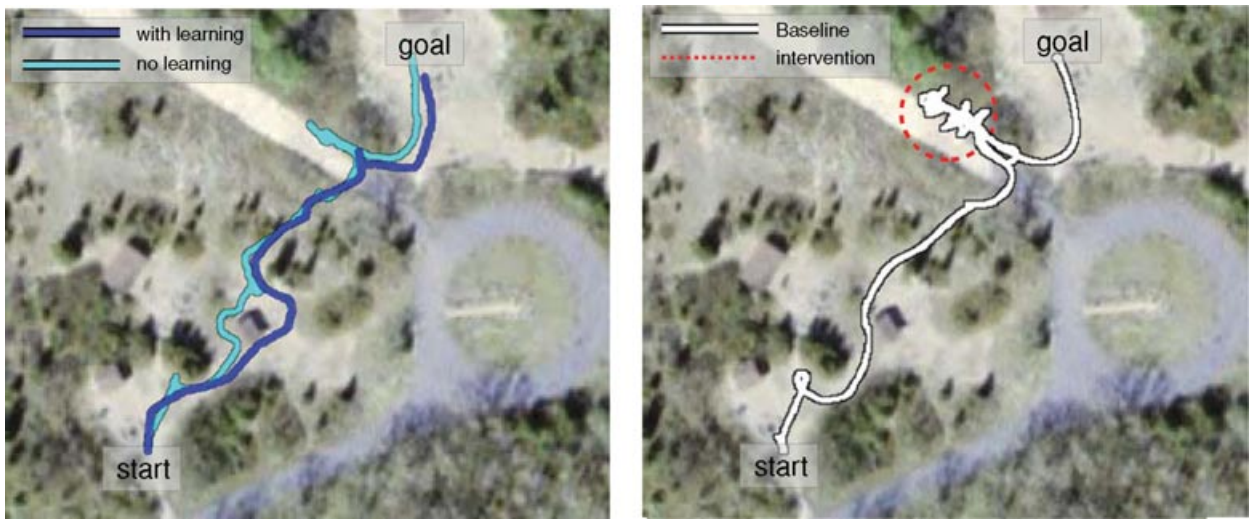


Figure 25. Aerial plots of comparison runs of long-range, short-range, and Baseline systems. On the left are plotted the paths followed by our system with long-range vision enabled (dark blue) or disabled (short range, light blue). The short-range vision explores a bit more to the left toward the end of the run as it lacks long-range vision. Those runs required no human intervention. On the right is the Baseline system, which systematically got into trouble toward the end of the run and required several human interventions to finish the run.

6.4.1. Field Tests Location 1: Sandy Hook, New Jersey

The course (Figure 25) consists of trees, bushes, and a few man-made obstacles. Results are reported in Table IV with average distances to goal, average times to goal, number of human interventions, and number of runs. The long-range vision performed the fastest and faster than the Baseline system by a factor of 3.2. The short-range system, even though much more shortsighted than the Baseline, performed significantly better and almost as well as the long-range system. It is important to note that the traveled distance of the long-range system was almost identical on average to that of the short-range system but the former still performed faster on average. In fact, the distances of the two long-range system runs were 113.79 and 130.52 m, averaging 122.15 m, and the times were 108.70 and 119.03 s, averaging 113.86 s. This can be explained by the fact that even though the long-range system might have chosen longer paths than the short-range system, it made smooth decisions and maintained a higher cruising speed, whereas the shortsighted robot made poor decisions and had to stop or make stronger turns more often, thus increasing the total run time.

Table IV. Performance comparison of long-range, short-range, and Baseline systems. The long-range system performed the fastest, closely followed by the same system with the long-range vision turned off, and the Baseline took significantly more time to reach the goal and required two human interventions after getting stuck in bushes.

System	Average time (s)	Average distance (m)	Human interventions	Number of runs
Long range	113.9	122.2	0	2
Short range only	123.7	122.9	0	1
Baseline	364.6	200.4	2	2

6.4.2. Field Tests Location 2: Cross Farm Park, Holmdel, New Jersey

This run demonstrates the performance of the long-range system over the Baseline system, over a long distance. (The width of the aerial view in Figure 26 is approximately 300 m.) A wide tall-grass area separates the start and the goal points; the long-range system clearly identifies the far edge of this area from the beginning, whereas the Baseline steers away only after reaching the grass. The Baseline also required human intervention after getting stuck in the grass. The long-range system, however, stayed away from trouble and even identified the road as easily traversable, going out of its way to drive on it for a bit. From Table V, the long-range system ran 198% faster and traveled only 65% of the distance of the Baseline run.

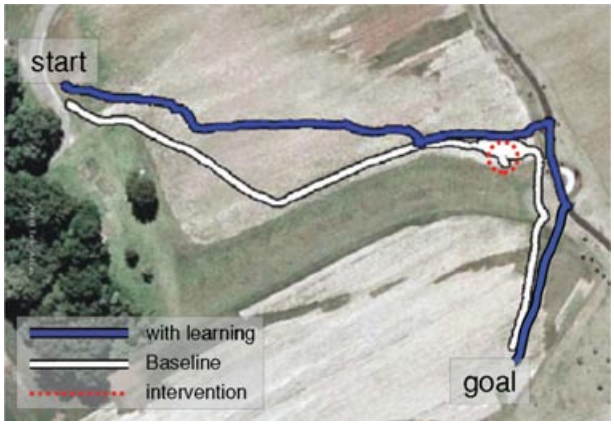


Figure 26. Aerial plots of comparison runs of long-range and Baseline systems. This aerial view is approximately 300 m wide. The long-range system in blue clearly identifies the big tall-grass area from the beginning and stays out of trouble, whereas the Baseline system got close to the grass and got into trouble.

Table V. Performance comparison of long-range and Baseline systems. The long-range system performed almost twice as fast as the Baseline and did not require any human intervention.

System	Time (s)	Distance (m)	Human interventions	Number of runs
Long range	254.2	314.0	0	1
Baseline	503.8	479.2	1	1

7. FUTURE WORK

7.1. Layer 3: Mapping and Planning

The hyperbolic polar map provides an efficient solution to vision-based and vehicle-centered mapping. Nevertheless, for courses longer than in LAGR tests, it needs to be coupled with an abstract global mapping as suggested by Layer 3 in Figure 2. This Layer 3 has not been studied for this project; however, a landmark recognition module for perception with a topological mapping and planning module seem to offer an elegant solution. Of course, the traditional Cartesian mapping remains easy to implement but its limitations (inefficient memory and, computationally wise, susceptibility to pose errors) should ultimately lead research to more abstract representations.

7.2. Layer 2: Mapping

The following current limitations of the h-polar mapping may be improved:

- **Points coverage (merging and splitting):** It is assumed that new data appear in front of the vehicle when moving forward, then merging with previous data. Points moving outside of the field of view into cells covering smaller areas scatter into tiny dots when they are used to represent a bigger area. Merging and splitting heuristics may be improved to cope with those issues but it is not clear whether they can solve it and for what price. However, this is in practice rarely an issue thanks to subsampling and smoothing and because points appear in front of the vehicle. Thus most points outside of the field of view compress together when moving away from the center.
- **Ground plane assumption:** Although 3D coordinates are kept internally, planning is executed in a flat 2D world. Z costs could be incorporated in the planning to reflect terrain relief.
- **Hysteresis:** An additional and fairly simple planning feature missing in the current system is planning stabilization through hysteresis. A lot of the instabilities sometimes observed during field testing are due to the inability to stick to a high-level planning decision, switching back and forth between different paths.
- **Registration:** h-polar provides an efficient space for registration of image patches. Registration techniques to cope with pose imprecisions should be investigated to improve the mapping precision.

7.3. Layer 1: Trajectory Recording and Learning

Besides exploring more difference measures to cover more of the trajectory search space as discussed above, another interesting way of research is to increase the autonomy of trajectory recording with online and offline methods, in order to facilitate even more the recording process without any human intervention and to constantly optimize trajectories during navigation.

- **Offline trajectories search with global search heuristics:** Using a minimal set of basic recorded trajectories, an offline algorithm could reconstruct new nonrecorded trajectories from pieces of those basic trajectories. This way, all slots of the bank could get filled and optimized more easily. To speed up the search for optimized trajectories, global search heuristics such as genetic algorithms

could be used to take advantage of the best trajectory pieces combinations.

- **Online trajectories recording in production mode:** When the vehicle runs in production mode, it is very cheap to record and add new trajectories on the fly when more optimized ones occur. With minimal effort, the robot can constantly improve its driving skills through experience.
- **Online trajectories recording and search in self-supervised mode:** Given a large obstacle-free training area, the robot can try to fill its trajectory bank itself by knowing which areas need trajectories or optimization. During this self-supervised learning mode, the trajectory space could be searched using simple wheel commands heuristics and gradient descent to approach the desired target points.

7.4. Layer 1: Visual Odometry

We are currently working on using bearing-only VO to bootstrap an additional "translation-only VO" step that operates on very nearby patches at the bottom of the image. Such patches have usually been difficult to track, as small features on the ground are often indistinct, whereas larger areas of texture may require a rotation-invariant representation. Using the rotation estimate from bearing-only VO, large ground patches may be rotated in the local ground plane to their expected orientations in the current frame. The distance to such nearby patches may be estimated with stereo vision at a reasonable accuracy even at low resolutions. We expect such a three-DOF VO system to be far cheaper than existing VO methods, which estimate translation and rotation simultaneously, requiring higher resolutions.

8. CONCLUSION

A complete and robust software navigation system was developed providing collision-free and long-range navigation capabilities. The multilayer perception, mapping, and planning architecture handles the latency and frequency issues induced by sophisticated processing. It also conveniently separates the necessary different types of mapping and planning for different ranges, a Cartesian map with trajectory planning in the short range and a hyperbolic polar map with grid-based planning in the long range. The cheap but robust fast perception, planning, and VO layer provide the low-level stability required to support the sophisticated long-range vision. The maneuver dictionary and VO contributed to the robust real-time navigation thanks to their simplicity and efficiency. Similarly to this paper, the video (Sermanet et al., 2008) explains and demonstrates this entire system.

Results show individual performance of key modules and overall performance as well. The complete system shows a systematic performance improvement over the reference Baseline system and over itself when long-range vision is turned off. Further research would mainly address coupling the current navigation layers with a global abstract layer for quasi-infinite global navigation.

ACKNOWLEDGMENTS

This work was supported in part by the DARPA LAGR program under contract HR001105C0038.

REFERENCES

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

Albus, J., Bostelman, R., Chang, T., Hong, T., Shackleford, W., & Shneier, M. (2006). Learning in a hierarchical control system: 4D/RCS in the DARPA LAGR program. Journal of Field Robotics, 23(11–12), 975–1003.

Andersen, E., & Taylor, C. (2007). Improving MAV pose estimation using visual information. In Proceedings of International Conference on Intelligent Robots and Systems (IROS), San Diego, CA (pp. 3745–3750). IEEE.

Angelova, A., Matthies, L., Helmick, D., & Perona, P. (2006). Slip prediction using visual information. Robotics: Science and Systems II. Cambridge, MA: The MIT Press.

Angelova, A., Matthies, L., Helmick, D., & Perona, P. (2007). Fast terrain classification using variable-length representation for autonomous navigation. In Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR), Minneapolis, MN.

Bajracharya, M., Tang, B., Howard, A., Turmon, M., & Matthies, L. (2008). Learning long-range terrain classification for autonomous navigation. In Proceedings of International Conference on Robotics and Automation (ICRA).

Bayouth, M., Nourbakhsh, I., & Thorpe, C. (1997). A hybrid human-computer autonomous vehicle architecture. Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Control.

Beetz, M. (2001). Structured reactive controllers. Autonomous Agents and Multi-Agent Systems, 4(1–2), 25–55.

Behnke, S. (2004). Local multiresolution path planning. In Robocup 2003: Robot Soccer World Cup VII.

Bibby, C., & Reid, I. (2007). Simultaneous localisation and mapping in dynamic environments (SLAMIDE) with reversible data association. In Proceedings of Robotics: Science and Systems (RSS).

Blas, R., Agrawal, M., Sundaresan, A., & Konolige, K. (2008). Fast color/texture segmentation for outdoor robots. In IEEE International Conference on Intelligent Robots and Systems, Nice, France.

Brooks, R. (1986). A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, 2(1), 14–23.

Brown, A., & Sullivan, D. (2002). Inertial navigation electro-optical aiding during GPS dropouts. In Proceedings of the Joint Navigation Conference.

Bryson, A. E., & Ho, Y. C. (1975). Applied optimal control. New York: Hemisphere Publishing.

Clemente, L., Davison, A., Reid, I., Neira, J., & Tards, J. (2007). Mapping large loops with a single hand-held camera. In Proceedings of Robotics: Science and Systems (RSS).

Coombs, D., Murphy, K., Lacaze, A., & Legowik, S. (2000). Driving autonomously offroad up to 35 km/h. In Intelligent Vehicles Symposium.

Elfes, A. (1991). Occupancy grids: A stochastic spatial representation for active robot perception. In S. S. Iyengar and A. Elfes (Eds.), Autonomous mobile robots: perception, mapping, and navigation (p. 6071). IEEE Computer Society Press.

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM, 24(6), 381–395.

Frazzoli, E., Dahleh, M. A., & Feron, E. (2005). Maneuver-based motion planning for nonlinear systems with symmetries. IEEE Transactions on Robotics, 21(6), 1077–1091.

Goldberg, S. B., Maimone, M. W., & Matthies, L. (2002). Stereo vision and robot navigation software for planetary exploration. In IEEE Aerospace Conference Proceedings, Big Sky, MT (pp. 2025–2036).

Grudic, G., Mulligan, J., Otte, M., & Bates, A. (2007). Online learning of multiple perceptual models for navigation in unknown terrain. In 6th International Conference on Field and Service Robotics.

Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Han, J., Flepp, B., Muller, U., & LeCun, Y. (2007). Online learning for offroad robots: Using spatial label propagation to learn long-range traversability. In Proceedings of Robotics: Science and Systems (RSS).

Hadsell, R., Sermanet, P., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U., & LeCun, Y. (2009). Learning long-range vision for autonomous off-road driving. Journal of Field Robotics. In Press.

Harris, C., & Stephens, M. (1988). A combined corner and edge detector. In Proceedings of the 4th Alvey Vision Conference (pp. 147–151).

Hofmann-Wellenhof, B., Lichtenegger, H., & Collins, J. (2001). Global positioning system: Theory and practice, 5th ed. Vienna: Springer-Verlag.

Howard, T., & Kelly, A. (2005). Terrain-adaptive generation of optimal continuous trajectories for mobile robots. In International Symposium on Artificial Intelligence.

Kanayama, Y., & Hartman, B. (1988). Smooth local path planning for autonomous vehicles (Tech. Rep.). Santa Barbara: Department of Computer Science, University of California.

Kanayama, Y., & Miyake, N. (1985). Trajectory generation for mobile robots. In Robotics research. Cambridge, MA: MIT Press.

Kelly, A., & Nagy, B. (2003). Reactive nonholonomic trajectory generation via parametric optimal control. International Journal of Robotics Research, 22(7–8), 583–601.

Kelly, A., & Stentz, A. (1998). Stereo vision enhancements for low-cost outdoor autonomous vehicles. ICRA Workshop WS-7.

Kelly, A., Stentz, A., Amidi, O., Bode, M., Bradley, D., Diaz-Calderon, A., Happold, M., Herman, H., Mandelbaum, R., Pilarski, T., Rander, P., Thayer, S., Vallidis, N., & Warner, R. (2006). Toward reliable off road autonomous vehicles operating in challenging environments. International Journal of Robotics Research, 25, (5–6), 449–483.

Komoriya, K., Tachi, S., & Tanie, K. (1984). A method for autonomous locomotion of mobile robots. Journal of the Robotics Society of Japan, 2, 222–231.

Kriegman, D. J., Triendl, E., & Binford, T. O. (1989). Stereo vision and navigation in buildings for mobile robots. IEEE Transactions on Robotics and Automation, 5(6), 792–803.

Longega, L., Panzieri, S., Pascucci, F., & Ulivi, G. (2003). Indoor robot navigation using log-polar local maps. In International IFAC Symposium on Robot Control.

Lookingbill, A., Lieb, D., & Thrun, S. (2007). Optical flow approaches for self-supervised learning in autonomous mobile robot navigation (pp. 29–44).

Low, K. H., Leow, W. K., & Ang, M. H. Jr. (2002). Integrated planning and control of mobile robot with self-organizing neural network. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'02) (vol. 4, pp. 3870–3875).

Maimone, M., Cheng, Y., & Matthies, L. (2007). Two years of visual odometry on the Mars exploration rovers: Field reports. Journal of Field Robotics, 24(3), 169–186.

Manduchi, R., Castano, A., Talukder, A., & Matthies, L. (2003). Obstacle detection and terrain classification for autonomous off-road navigation. Autonomous Robots, 18, 81–102.

Montiel, J. M. M., & Davison, A. (2006). A visual compass based on SLAM. In IEEE International Conference on Robotics and Automation (ICRA).

Nistér, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR) (vol. 1, pp. 652–659).

Ojeda, L., Cruz, D., Reina, G., & Borenstein, J. (2006). Current-based slippage detection and odometry correction for mobile robots and planetary rovers. IEEE Transactions on Robotics and Automation, 22(2), 366–378.

Ollis, M., Huang, W., & Happold, M. (2007). A Bayesian approach to imitation learning for robot navigation. In Proceedings of International Conference on Intelligent Robots and Systems (IROS).

PiedMonte, M., & Feron, E. (1999). Aggressive maneuvering of autonomous aerial vehicles: A human-centered approach. In International Symposium on Robotics Research.

Saxena, A., Schulte, J., & Ng, A. (2007). Depth estimation using monocular and stereo cues. In 20th International Joint Conference on Artificial Intelligence.

Schäfer, H., Hahnfeld, P., & Berns, K. (2007). Real-time visual self-localisation in dynamic environments. In Autonome Mobile Systeme.

Schaffalitzky, F., Zisserman, A., Hartley, R. I., & Torr, P. H. S. (2000). A six point solution for structure and motion. In Proceedings of the European Conference on Computer Vision (pp. 632–648).

Sermanet, P., Ben, R. H. J., Naz, A., Beat, E., & Lecun, M. Y. (2007). Speed-range dilammas for vision-based navigation in unstructured terrain. In IFAC Symposium on Intelligent Autonomous Vehicles (IFAC).

Sermanet, P., Hadsell, R., Scoffier, M., Grimes, M., Ben, J., Erkan, A., Crudele, C., Muller, U., & LeCun, Y. (2008). DARPA LAGR program: Learning applied to long-range vision using a collision-free navigation platform. http://www.youtube.com/watch?v=lowcgokiRG8. In AAAI Video Competition.

Sofman, B., Lin, E., Bagnell, J., Vandapel, N., & Stentz, A. (2006). Improving robot navigation through self-supervised online learning. In Proceedings of Robotics: Science and Systems (RSS).

Spenko, M., Kuroda, Y., Dubowsky, S., & Iagnemma, K. (2006). Hazard avoidance for high-speed mobile robots in rough terrain. Journal of Field Robotics, 23(5), 311–331.

Stavens, D., & Thrun, S. (2006). A self-supervised terrain roughness estimator for off-road autonomous driving. In Proceedings of Conference on Uncertainty in AI (UAI).

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In Robotics and Automation (RA).

Stewart, E. (2004). Intel integrated performance primitives: How to optimize software applications using Intel IPP. Intel Press.

Stewenius, H., Nister, D., Kahl, F., & Schaffalitzky, F. (2005a). A minimal solution for relative pose with unknown focal length. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005 (CVPR 2005) (vol. 2, pp. 789–794).

Stewenius, H., Schaffalitzky, F., & Nister, D. (2005b). How hard is 3-view triangulation really? Proceedings of the Tenth IEEE International Conference on Computer vision (vol. 1, pp. 686–693). Washington, DC: IEEE Computer Society.

Sukkarieh, S., Nebot, E., & Durrant-Whyte, H. (1999). A high integrity IMU/GPS navigation loop for autonomous land vehicle applications. IEEE Transactions on Robotics and Automation, 15, 572–578.

Thrun, S. (2003). Robotic mapping: A survey. In Exploring artificial intelligence in the new millennium (pp. 1–35). San Francisco, CA: Morgan Kaufmann Publishers Inc.

Triggs, B. (1999). Camera pose and calibration from 4 or 5 known 3D points. In Proceedings 7th International. Conference on Computer Vision (pp. 278–284). IEEE Computer Society Press.

Tsumura, T., Fujiwara, N., Shirakawa, T., & Hashimoto, M. (1981). An experimental system for automatic guidance of a robotic vehicle following a route stored in memory. In Proceedings of the 11th International Symposium on Industrial Robots (pp. 187–193).

Ulivi, G., & Vendittelli, M. (1997). Fuzzy maps: A new tool for mobile robot perception and planning. Journal of Robotic Systems, 14(3), 179–197.

Vernaza, P., Taskar, B., & Lee, D. D. (2008). Online, self-supervised terrain classification via discriminatively trained submodular Markov random fields. In Proceedings of the IEEE International Conference on Robotics and Automation.

Veth, M., & Raquet, J. (2006). Two-dimensional stochastic projections for tight integration of optical and inertial sensors for navigation. In National Technical Meeting Proceedings of the Institute of Navigation (pp. 587–596).

Veth, M., Raquet, J., & Pachter, M. (2006). Stochastic constraints for efficient image correspondence search. IEEE Transactions on Aerospace and Electronic Systems, 42(3), 973–982.

Wagner, M., Baird, J. C., & Barbaresi, W. (1980). The locus of environmental attention. Journal of Environmental Psychology, 1, 195–206.

Ward, C. C., & Iagnemma, K. (2007a). Classification-based wheel slip detection and detector fusion for outdoor mobile robots. In Proceedings of International Conference on Robotics and Automation (ICRA) (pp. 2730–2735). IEEE.

Ward, C. C., & Iagnemma, K. (2007b). Model-based wheel slip detection for outdoor mobile robots. In Proceedings of International Conference on Robotics and Automation (ICRA) (pp. 2724–2729). IEEE.

Zhang, H., & Ostrowski, J. (2002). Visual motion planning for mobile robots. IEEE Transactions on Robotics and Automation, 18(22), 199–208.