Planning and Control in Unstructured Terrain

Brian P. Gerkey
Willow Garage
gerkey@willowgarage.com

Kurt Konolige
SRI Artificial Intelligence Center
konolige@ai.sri.com

***Abstract*— We consider the problem of autonomous navigation in an unstructured outdoor environment. We describe the planning and control aspects of an implemented system that drives a robot at modest speeds ($\sim$1 m/s) over a variety of outdoor terrain. In real time, we use a gradient technique to plan globally optimal paths on a cost map, then employ a predictive dynamic controller to compute local velocity commands. Our planner and controller are considered the "best in class" among 10 teams competing in the DARPA Learning Applied to Ground Robotics (LAGR) program.**

I. Introduction

We consider the problem of autonomous navigation in an unstructured outdoor environment. The goal is for a small outdoor robot (Fig. 1) to come into a new area, learn about and map its environment, and move to a given goal at modest speeds ($\sim$1 m/s). In this short paper, we describe the planning and control aspects of an implemented system performing this task. The planner and controller are considered the "best in class" among 10 teams competing in the DARPA Learning Applied to Ground Robotics (LAGR) project [1]. In experiments conducted by an independent testing team in terrain that we do not see beforehand, our system safely controls the LAGR robot at its maximum speed of 1.3 m/s, with the robot flawlessly negotiating maze-like obstacle courses at an average speed of 1.1 m/s.

For planning, we use a 2D grid map, in which each map cell contains multidimensional criteria relevant for motion, such as the presence of an obstacle, or membership in a recognized path. These criteria are merged into a scalar cost map, from which an optimal global plan is computed at a 10 Hz rate using a gradient method [2]. The gradient method is unusual in computing the Euclidean distance (in a uniform cost field), even though the input is sampled on a grid. Another unique aspect of the planning method is its use of intermediate *line goals*, in which all points on a line goal are considered equally good. Line goals help to keep the planner from always pushing directly towards the goal, when a promising path is found that leads in the general direction of the goal.

The planner is optimistic, in that it considers the robot to be a holonomic cylinder, with diameter equal to the vehicle's width, and infinite acceleration. The robot is in fact non-holonomic (differentially-driven), with a rectangular footprint (longer than wide), and fairly low maximum acceleration. Thus the planner can generate unrealistic paths. To ensure safe, feasible motion, we employ a local predictive controller that encodes the robot's kinodynamic constraints,

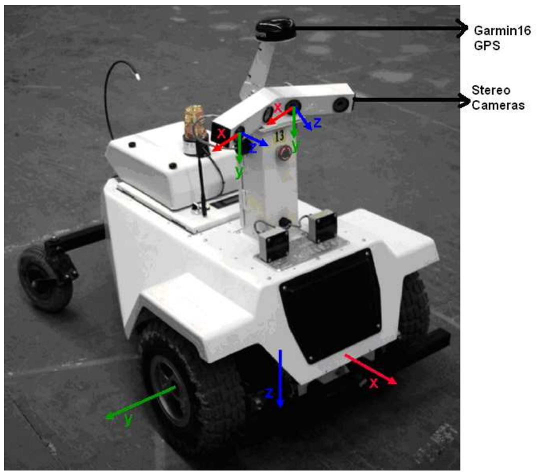


Fig. 1. *The LAGR robot is a differentially driven outdoor vehicle, equipped with GPS, two pairs of stereo cameras, and a tactile bumper.*

including finite acceleration and non-zero turning radius. This controller samples the vehicle's control space, predicting possible trajectories with discrete-time simulation over a receding horizon. The predicted trajectories are evaluated with a multi-objective criterion that encodes several components of the planner's cost structure, with the best trajectory being chosen at each iteration. We also monitor and actively compensate for wheel slip, which can be significant in outdoor terrain. This efficient controller easily runs at the same rate as the planner.

II. Related work

Our planning approach is an enhanced reimplementation of the gradient technique [2], which computes a global navigation function over the cost map. A similar approach is used in wavefront planning [3], although wavefront planners usually minimize Manhattan or diagonal distance, whereas we minimize Euclidean distance. Level sets [4] offer an equivalent method for computing paths that minimize Euclidean distance. The underlying computation for all such planners is a variation on dynamic programming [5]. For reasons of efficiency, our planner treats the robot as a holonomic cylinder with no kinodynamic constraints. These constraints could be incorporated into the planner by use of sampling-based algorithms such as rapidly-exploring random trees (RRTs) [6].

We enforce kinodynamic constraints in our local con-

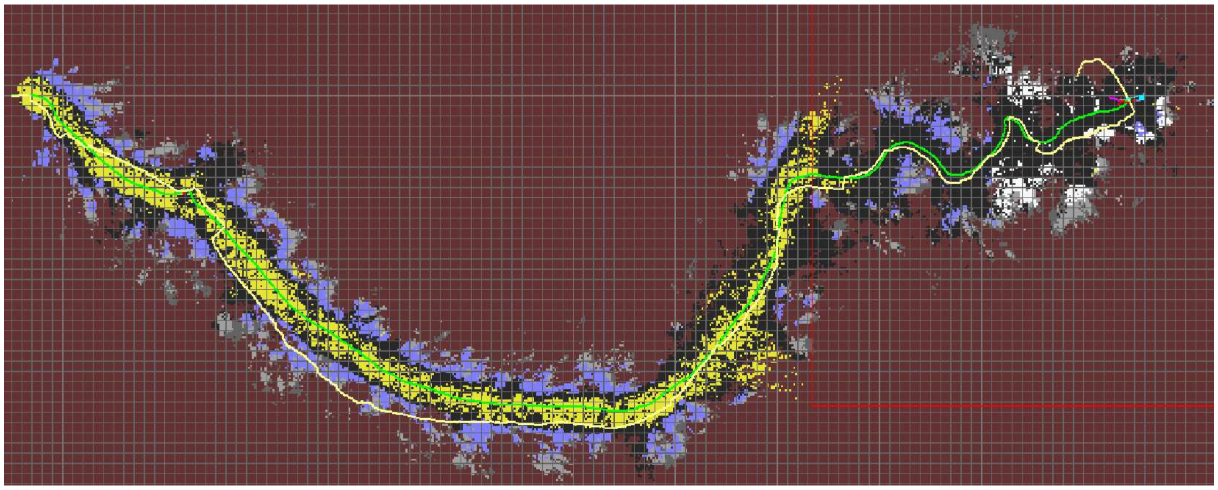


Fig. 2. *Our system constructs coherent cost maps over long (in this case, 130m) autonomous runs. Grid color code: yellow is recognized path; black is free space; purple, gray, and white are obstacles; brown is unknown. The green and yellow lines indicate the path followed by the robot, using different pose estimation techniques.*

troller. Control algorithms such as DWA [7] compute local controls by first determining a target trajectory in position or velocity space (usually a circular arc or other simple curve), then inverting the robot's dynamics to find the desired velocity commands that will produce that trajectory. We instead explore the control space directly, and simulate and evaluate the resulting trajectories, in a manner reminiscent of the controller used in the RANGER system [8], with the key differences being the definition of the state space and the trajectory evaluation function. The Stanley controller [9] also rolls out and evaluates possible trajectories, but divides them into two categories ("nudges" and "swerves"), based on their expected lateral acceleration. Howard et al. [10] present a more general approach to constraining the search for controls by first sampling directly in the vehicle's state space.

III. Map representation

For indoor work, a standard map representation is a 2D *occupancy grid* [11], which gives the probability of each cell in the map being occupied by an obstacle. Alternatives for outdoor environments include 2.5D elevation maps and full 3D voxel maps [12]. These representations can be used to determine allowable kinematic and dynamic paths for an outdoor robot in rough terrain. We choose to keep the simpler 2D occupancy grid, foregoing any complex calculation of the robot's interaction with the terrain. Instead, we abstract the geometrical characteristics of terrain into a set of categories, and fuse information from these categories to create a scalar *cost* of movement.

We use a grid of 20cm $\times$ 20cm cells to represent the global map. Each cell has a probability of belonging to one of four categories, derived from visual analysis: obstacle, ground plane free space, sight line free space [1], and path free space. The sight line free space is an interesting new category based on unobstructed visual sight lines: we know that there is free space out to the furthest obstacle, even though we can't tell exactly how far away that obstacle is.

Note that these categories are not mutually exclusive, since, for example, a cell under an overhanging branch could have both path and obstacle properties. We are interested in converting these probabilities into a cost of traversing the cell. If the probabilities were mutually exclusive, we would simply form the cost function as a weighted sum. With non-exclusive categories, we chose a simple prioritization schedule to determine the cost. Obstacles have the highest priority, followed by ground plane, sight lines, and paths. Each category has its own threshold for significance. For example, if the probability of an obstacle is low enough, it will be ignored in favor of one of the other categories. The combination of priorities and thresholds yields a very flexible method for determining costs. Fig. 2 shows a color-coded representation of computed costs for a typical run.

IV. Planning

The LAGR robot was provided with a "baseline" system that used implementations of D* [13] for global planning and Dynamic Window Approach (DWA) [7] for local control. Using this system, we (as well as other teams) had frequent crashes and undesirable motion. The main causes were the slowness of the planner and the failure of the controller to sufficiently account for the robot's dynamics. The D* planner is optimized for very large-scale environments. It uses dynamic programming to compute the minimum-cost potential to the goal at each cell; it needs significant resources to maintain the indices necessary to unravel the minimum-cost computations incrementally. In our environments (100m x 200m, 20 cm^2 cells) it would take many seconds to compute a plan, even when only a small portion of the map was filled. For large-scale maps this may be acceptable, but we need much faster response to tactical maneuvers over smaller scales (e.g., cul-de-sacs).

Instead, we re-implemented a gradient planner [2], [14] that computes optimal paths from the goal to the robot, given a cost map. The gradient planner is a wavefront planner that computes the cost of getting to a goal or goals at every cell in the workspace. It works by using a local neighborhood to update the cost of a cell. If the cell's cost is higher than the cost of a neighbor cell plus the local transit cost, then it is updated with the new cost. The overall algorithm starts by initializing the goal with a zero cost, and everything else with a very large cost. All goal cells are put onto an "open" list. The algorithm runs by popping a cell of the open list, and updating each of the cell's neighbors. Any neighbor that has a lowered cost is put back onto the open list. The algorithm finishes when the open list is empty.

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

- Unlike other implementations, it uses a true Euclidean metric, rather than a Manhattan or diagonal metric, in performing the update step [4]. The update can be performed on the four nearest neighbors of a cell. Generally speaking, the two lowest-cost neighbors can be used to determine the direction of propagation of the cost potential, and the cell updated with an appropriate distance based on this direction.
- The algorithm computes the configuration space for a circular robot, and includes safety distances to obstacles. This is one of the interesting parts of the gradient method. Since there is already a method for computing the distance transform from a set of points, the configuration space can be computed efficiently. The obstacle points are entered as goal points, and the update algorithm is run over each of these points, generating a new open list. Each open list is processed fully, leading to a sequence of open lists. At the end of n cycles, the distance to obstacles has been determined up to $n * c$, where c is the cell size. Usually this is done to a distance of 3 or 4 times the robot radius, enough to establish a safety cushion to the obstacle. Finally, a cost is associated with the distance: an infinite cost within the robot radius to an obstacle, and a decreasing cost moving away from this.
- The queue handling is extremely efficient, using threshold-based queues, rather than a best-first update, which has high overhead for sorting. Instead, we use a 2-priority-queue method. A threshold shuttles new cells to one queue or the other, depending on whether their cost is greater or less than the threshold. The low-cost queue is always processed first. When no more cells remain in it, the threshold is increased, the second queue becomes the low-cost queue, and a new high-cost queue is initialized. This queue strategy is the key to the good performance of the algorithm: each update step happens very rapidly. Although the complexity of the algorithm is the order of the area to be covered, and there is no "best first" search from the goal to the robot position, still the extreme rapidity of each step makes it possible to cover reasonable areas (e.g., 80m x 80m) in several tens of milliseconds.
- Rapid switching of global paths is avoided by including hysteresis - lowering the cost along the path. There is a tradeoff between sticking to the current path, and exploring some new path if current readings indicate it might be better. We lower the cost enough so that it takes a significant amount of new information to turn the path aside.

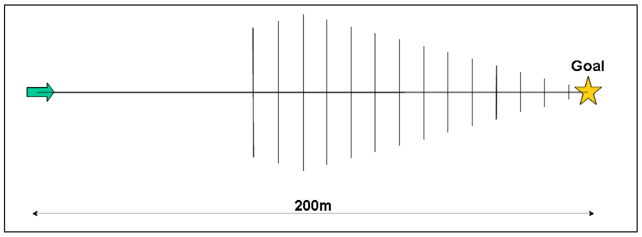


Fig. 3. *Line goals for a robot in a 200m environment. The line goal is placed 60m ahead of the robot, and its extent varies with the distance to the goal.*

Typically we run the global planner within a subregion of the whole map, since the robot is continuously moving towards the goal and encountering new areas. On longer runs, up to 200m, we use an 80m x 80m area; the global planner runs in about 30 ms in this region. Unless there is a large cul-de-sac, longer than 80m, this area is sufficient to maneuver the robot tactically around obstacles. For more global planning, which occurs when starting a run with a previously-made map, we run the planner over the whole area, which can take up to 100 ms for a large 100m x 200m map.

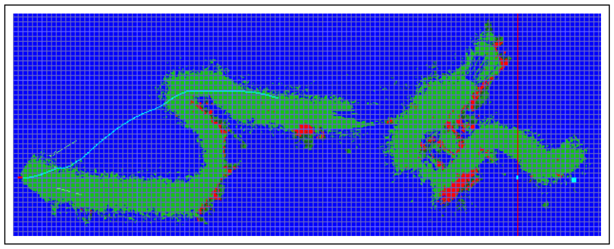
The global planner is optimistic in assuming the robot to be circular, with a diameter equal to the width of the robot. Also, it does not take into account the nonholonomic nature of the robot's motion. Instead, we rely on a local controller to produce feasible driving motions (Section V).

A. Line goals

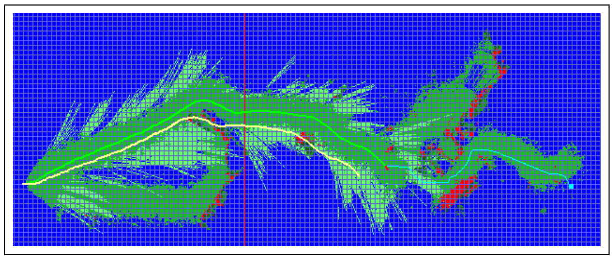
One of the problems encountered in directing the robot towards a point goal is that the plans tend to constantly urge the robot towards the center of the map. This is not necessarily an efficient strategy, because, for example, the robot will prefer to run near vegetation on the side of a path that does not point directly towards the goal. Instead, when the robot is far from the goal, we posit a relaxed *virtual goal line* that allows the robot to pursue more indirect paths to the goal (Fig. 3). In experiments, the robot is able to navigate more than 50m off the center line to the goal, and consequently find easily traversed paths that would have been difficult to find if it had headed directly to the goal (Fig. 2).

B. Map reuse

After creating a map in navigating to a goal, the map can be re-used to be more efficient at the same task. For example, the LAGR tests often involve maneuvering around some obstacles between the start and goal. Once the map is



(a)



(b)

Fig. 4. *Second run of the LAGR Test 26B course. The initial run produced the map (a) – the trajectory is the initial planned path for the second run. At the end of the run (b) the robot has navigated around several cul-de-sacs found in the first run. The green line is the actual robot trajectory found by VO, the yellow line is from (noisy) GPS.*

available, a better global plan can be computed for the same task. Fig. 4 shows one example of this capability. The initial map (a) has several cul-de-sacs that the robot entered, before eventually finding its way to the goal. The planner (blue trajectory) takes shortcuts on the second run. The parameters of the cost map need careful tuning, because in this case the robot is traversing unknown territory in performing the cutoff.

Currently we run in open-loop mode when using a map, that is, we register the robot's initial position in the map, based on images taken during the previous run; but then we rely solely on VO to keep the robot globally registered. Obviously there will be drift over longer runs, and the correct solution would be to continuously register against the map, using visual matching.

V. Control

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

A. Trajectory generation

We take an approach that is opposite to techniques such as DWA. Instead of searching the space of feasible *trajectories*, we search the space of feasible *controls*. As is the case with most differentially-driven platforms, the LAGR robot is commanded by a pair $(\dot{x}, \dot{\theta})$ of desired translational and rotational velocities.[1] Thus we have a 2D space of possible commands to consider.

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

This space is bounded in each dimension by velocity limits that reflect the vehicle's capabilities. Because we are seeking *good*, as opposed to *optimal*, control, we sample, rather than exhaustively search, this rectangular region of allowed velocities. We take a regular sampling ($\sim$25 in each dimension, $\sim$625 total), and for each sample simulate the effect of applying those controls to the robot over a short time horizon ($\sim$2s). The simulation predicts the robot's trajectory as a sequence of 5-dimensional $(x, y, \theta, \dot{x}, \dot{\theta})$ states with a discrete-time approximation of the vehicle's dynamics.

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

The generated trajectories, projected into the (x, y) plane, are smooth, continuous 2-dimensional curves that, depending on the acceleration limits, may not be easily parameterizable. For the LAGR robot, the trajectories are generally not circular arcs (Fig. 5).

B. Trajectory evaluation

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

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

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

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

C. Supervisory control

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

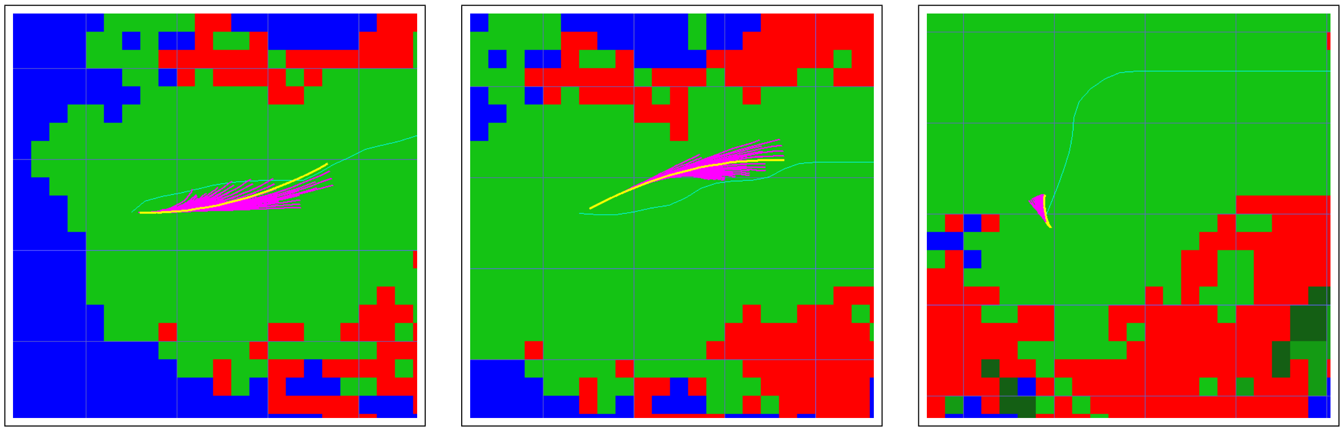

(a) (b) (c)

Fig. 5. *The controller generates trajectories by sampling feasible velocities and simulating their application over a short time horizon. Generated trajectories are purple, the chosen trajectory is yellow, the desired global path is cyan, and obstacles are red. As shown in (a) and (b), the trajectories are smooth but not easily parameterizable as they depend on the vehicle's current velocity and its acceleration limits. When forward motion is not possible, backward trajectories are considered (c).*

logic to direct the order in which candidate velocities are simulated and evaluated.

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

D. *Slip handling*

Because the robot may have to traverse rough, steep terrain, it is necessary to detect and react to conditions in which the wheels slip or become stuck. We employ two mechanisms to handle these situations. In both cases, we are comparing the motion reported by the wheels to the motion estimated by visual odometry (VO), which is sufficiently accurate to be treated as ground truth [15].

First, the controller continuously compensates for the slip in each wheel by reducing its maximum speed. Our approach is similar to automotive traction control. For each wheel, we monitor the slip ratio s, defined as [16]:

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

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

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

VI. Performance

As part of the LAGR program, the independent evaluation group ran monthly blind tests of the perception, planning, and control software developed by all the teams and compared their performance to a baseline system. The last three tests of Phase I (Tests 11, 12, and 13) were considered the definitive performance measures for the first 18 months of the program. The SRI Team was first in the last two tests (12 and 13), after placing last in Test 11. Most of the problems in Test 11 were caused by using the baseline planner and controller, and we switched to the one described in the paper for Tests 12 and 13. We achieved the short run times in Tests 12 and 13 through a combination of precise map building, high-speed path planning, accurate predictive control, and careful selection of robot control parameters. Our average speed was over 1.1 m/s, while the robot top speed was limited to 1.3 m/s. Map building relied on VO to provide good localization, ground-plane analysis to help detect obstacles, and sight lines to identify distant regions that are likely to be navigable. We again placed first in the most recent (and penultimate) test, Test 26 (Fig. 4). In this case, our performance was greatly improved by our ability to correctly register and reuse learned maps, in order to identify better paths when running through previously explored areas.

VII. Summary

In this paper we have presented an integrated planning and control system for a robot navigating in unknown, outdoor terrain. We use a gradient technique to plan globally optimal paths on a cost map, then employ a predictive dynamic controller to compute local velocity commands. This system runs in real time on a robot moving at $\sim$1 m/s. Our planner and controller are considered the "best in class" among 10 teams competing in the DARPA Learning Applied to Ground Robotics (LAGR) program.

VIII. Acknowledgments

This work is supported in part by the DARPA LAGR program.

References

[1] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. P. Gerkey, "Outdoor mapping and navigation using stereo vision," in *Proc. of the Intl. Symp. on Experimental Robotics (ISER)*, Rio de Janeiro, Brazil, July 2006.

[2] K. Konolige, "A gradient method for realtime robot control," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2000.

[3] J.-C. Latombe, *Robot Motion Planning*. Norwell, Massachusetts: Kluwer Academic Publishers, 1991.

[4] R. Kimmel and J. A. Sethian, "Computing Geodesic Paths on Manifolds," *Proc. of the National Academy of Science*, vol. 95, pp. 8431–8435, July 1998.

[5] R. Bellman, *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.

[6] S. LaValle, *Planning Algorithms*. New York, New York: Cambridge University Press, 2006.

[7] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[8] A. Kelly, "A feedforward control approach to the local navigation problem for autonomous vehicles," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-94-17, May 1994.

[9] S. Thrun, M. Montemerlo, *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *J. of Field Robotics*, vol. 23, no. 9, pp. 661–692, Sept. 2006.

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

[11] H. Moravec and A. Elfes, "High resolution maps for wide angles sonar," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 1985.

[12] K. Iagnemma, F. Genot, and S. Dubowsky, "Rapid physics-based rough-terrain rover planning with sensor and control uncertainty," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 1999.

[13] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 4, 1994, pp. 3310–3317.

[14] R. Philippsen and R. Siegwart, "An interpolated dynamic navigation function," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2005.

[15] M. Agrawal and K. Konolige, "Rough terrain visual odometry," *Proc. of the Intl. Conf. on Advanced Robotics (ICAR)*, Aug. 2007.

[16] A. Angelova, L. Matthies, D. Helmick, G. Sibley, and P. Perona, "Learning to predict slip for ground robots," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2006, pp. 3324–3331.