Optimization-Based Navigation for the DARPA Grand Challenge

Dmitriy Kogan and Richard M. Murray

***Abstract*— This research addresses the path planning problem with a nonlinear optimization method running in real time. An optimization problem is continually solved to find a time-optimal, dynamically feasible trajectory from the vehicle's position to some receding horizon ahead (20m-70m forward). The locally optimal numerical solver optimizes both the spatial and temporal components of the trajectory simultaneously, and feeds its output to a trajectory-following controller. The method has been implemented and tested on a modified Ford E350 van. Using one stereo pair and four LADAR units as terrain sensors, the vehicle was able to consistently traverse a 2 mile obstacle course at the DGC qualifying event. At the main DGC event, the vehicle drove 8 autonomous miles through the Nevada desert before experiencing non-planning issues. During this time, the planning system generated a plan 4.28 times per second on average. This execution speed, coupled with a feedback-based trajectory-following controller was shown to be adequate at providing smooth and reliable obstacle avoidance even on complicated terrain.**

***Index Terms*— kinodynamic planning, receding horizon control, optimization-based control, dynamic inversion, autonomous navigation**

I. Introduction

AUTONOMOUS navigation is one of the major open problems in robotics research. This problem is both interesting theoretically and has wide-spread applications. To promote the development of this and other fields of interest to autonomous robotics, DARPA, the research arm of the Department of Defense of the United States, sponsored the DARPA Grand Challenge (DGC), an off-road race for autonomous, land-based vehicles. The DGC entries had to traverse as quickly as possible over 130 miles of Mojave Desert terrain, specified by a GPS-defined corridor. While moving, the vehicles had to observe corridor boundaries and DARPA-imposed speed limits. In order to eliminate sophisticated preplanning as a viable navigation option, DARPA distributed the corridor data to the teams only two hours before the race.

This research was developed to provide a navigation system for Team Caltech's entry to the DGC, Alice (Figure 1). Figure 2 outlines the top level design of Alice's systems and the connecting data flow. The vehicle-mounted sensors produce an estimate of Alice's pose and, via the map server, of the surrounding terrain. These are input to the planning system, which continually computes a trajectory for the vehicle to follow. A control system computes the actuator inputs necessary to keep Alice on this trajectory. All these components run asynchronously and process the newest data available to them at the time.

The authors are with Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA (email: dkogan@cds.caltech.edu; murray@cds.caltech.edu).

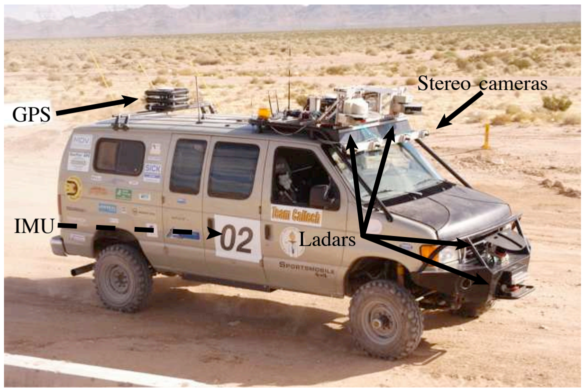


Fig. 1. Alice: Team Caltech's 2005 DGC entry. LADARs and cameras sense the terrain. The GPS and IMU units provide the vehicle state.

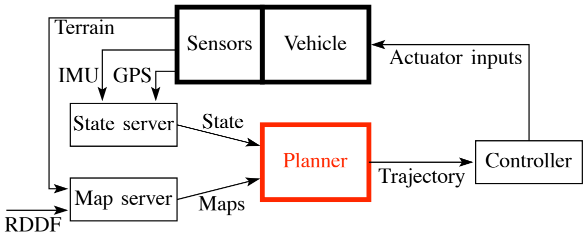


Fig. 2. A block diagram of the top level planner framework. The planner receives terrain data from the map server and state data from the IMU, GPS. These data are processed to produce trajectories, which are followed by the controller to send actuator inputs to the vehicle. The vehicle motion produces updated state and terrain data, which closes the loop.

DARPA specified the problem loosely, leaving several key decisions to the teams' discretion. The width of the course corridor was unlimited, not explicitly invalidating corridors much wider than the vehicle. Wider corridors could contain maze-like structures, necessitating use of a long-term planning algorithm, like D*. Conversely, if the corridor width was sufficiently limited, the trackline (corridor center) could serve as the long-term plan. Team Caltech decided to assume the latter, and to base its navigation off of the trackline.

Keeping these considerations in mind, we developed a planning system that

1) produces time-optimal plans to place well in the race.
2) is fast enough to run in real-time and has a low enough latency to produce good closed loop performance, even despite the added delays of the physical system.
3) is robust to various terrain/vehicle pose combinations.
4) produces dynamically feasible plans to ensure a low

error bound in performance of the trajectory-following control system.

Even though no ideal methods for tackling the planning problem currently exist, this problem is a well-researched one. Significant effort was directed into avoiding obstructions in a binary obstacle field. This is inadequate for our purpose, however, since we want to plan through both obstacles and impedances, where one could potentially travel, but with a reduced speed. Another well-researched approach to the planning problem is a graph search. This is a scheme where the terrain is represented as a graph with nodes corresponding to spatial locations, and the edges representing the cost of movement from one node to another. A* is a well-known algorithm of this type. It can plan an optimal path through a fully mapped environment, but since this information is not completely known *a priori* [1], this is insufficient. This particular drawback is addressed in D*, an extension of A* that dynamically and efficiently modifies the plan if a before-unknown obstacle is encountered [2]. Unfortunately, if only a spatial graph is searched, neither D* nor A* are enough, since the plans they produce include no dynamics. A car-like robot includes non-holonomic constraints, which would render unfollowable many spatial paths. Algorithms that employ A* to search an extended graph to produce feasible trajectories exist [3], [4]. Unfortunately, these are either too slow or simplify the problem too much by employing binary obstacles.

Other research efforts compute the optimal spatial path and then determine the optimal velocity profile that is feasible for a vehicle to traverse that path[5], [6]. This separation of spatial and temporal planning, however, is a potential source of non-optimality since the shortest path is not necessarily the fastest path.

To address these issues, work has been directed towards finding methods to compute dynamically feasible trajectories around obstacles in real-time. Some of those algorithms use a database of precomputed clothoid curves to speed up the online computations [7]. The discrete nature of this method may create difficulties in finding a feasible solution and can cause suboptimal trajectories to be produced.

Using an optimization method to generate plans would address those issues. An optimization problem can be set up to produce optimal plans that also avoid obstacles and satisfy dynamic feasibility constraints. An optimization-based planner that performed the spatial and temporal planning separately and based its spatial plans off D* was attempted in [8]. Unfortunately, since that research was done in 1991, the computational power available at the time was insufficient to solve the planning problem effectively, and the results from that research remain inconclusive.

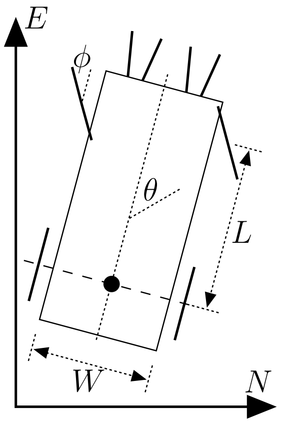
Real-time optimization-based control was successfully attempted in [9] to plan paths for the Caltech ducted fan, a testbed for planning and control algorithms [10]. In this research, we present a robot planning system that uses a convex numerical optimization method to explicitly compute, in real-time, optimal, dynamically feasible trajectories that satisfy all vehicle, terrain constraints, thus performing the spatial and temporal planning simultaneously. This method allows for fast driving and optimal obstacle avoidance even in complicated terrain, while utilizing a very simple control system. Further, this method is very flexible and could be extended to handle more complicated scenarios, such as dynamic obstacles.

II. Approach

Since the DGC race is over 130 miles long, it is both computationally prohibitive and unnecessary to plan the vehicle's trajectory all the way to the finish line at all times. Thus, we run the planner in a receding horizon framework, where the plans are computed not to the final goal, but to a point a set distance ahead on some rough estimate of the path towards the goal. The planning range thus continually moves forward with the vehicle.

A. Vehicle model

As mentioned earlier, the planner is designed to generate trajectories that are dynamically feasible in respect to some model of the vehicle. The model chosen here is the kinematic bicycle model (Figure 3).



$$\begin{aligned}\dot{N} &= v\cos\theta \\ \dot{E} &= v\sin\theta \\ \dot{v} &= a \\ \dot{\theta} &= \frac{v}{L}\tan\phi\end{aligned} \qquad (1)$$

Fig. 3. Here N and E are the Northing and Easting coordinates of the middle of the rear axle, v is the scalar speed of the center of the rear axle, θ is the yaw of the vehicle, a is the scalar longitudinal acceleration of the vehicle, L is the wheelbase of the vehicle, and ϕ is the steering angle.

B. System flatness

To apply a numerical optimization scheme to the path planning problem, we have to represent the space of all potential trajectories as a vector space. The space of all these trajectories clearly has infinite order. To use a numerical solver, it is necessary to approximate this space with a finite-dimensional one, and to facilitate fast computation, this dimension needs to be kept as low as possible. To begin to meet these goals, we employ a flat representation of (1), using $\{\theta(s), v(s), S_f\}$ as the bases, where s is the normalized arc length of the trajectory, $\theta(s)$ is the vehicle yaw, $v(s)$ is the scalar speed of the vehicle, and S_f is the total length of the trajectory (Figure 4). This representation provides a clean separation between the spatial and temporal components of a trajectory:

$$\begin{array}{rcl} N(s) & = & N_0 + S_f \int_0^s \cos(\theta(s))ds \\ E(s) & = & E_0 + S_f \int_0^s \sin(\theta(s))ds \\ \frac{dN}{ds} & = & S_f \cos(\theta(s)) \\ \frac{dE}{ds} & = & S_f \sin(\theta(s)) \\ \tan\phi & = & \frac{L}{S_f}\frac{d\theta}{ds} \\ \frac{d\phi}{ds} & = & \frac{L}{S_f \sec^2(\phi)}\frac{d^2\theta}{ds^2} \\ v(s) & = & S_f \frac{ds}{dt} \\ a & = & \frac{v}{S_f}\frac{dv}{ds} \\ T & = & S_f \int_0^s \frac{1}{v(s)}ds \end{array} \tag{2}$$

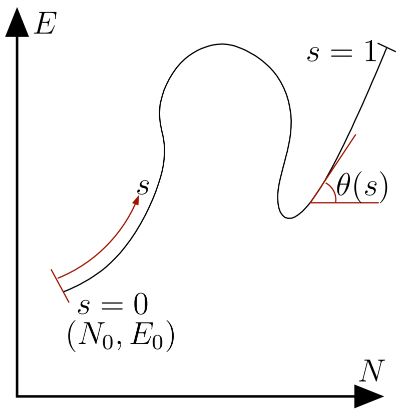


Fig. 4. The intrinsic coordinate system used to describe the solution space.

C. Problem formulation

We want to compute trajectories that are time optimal and satisfy various kinematic and terrain constraints. This can be stated within the context of a general nonlinear programming (NLP) optimization problem:

$$\text{Find } \vec{x} \in \mathbb{R}^n \text{ that minimizes } J(\vec{x}) \in \mathbb{R}$$
$$\text{subject to } \vec{L} \leq f(\vec{x}) \leq \vec{U} \in \mathbb{R}^m$$

To pose the planning problem as an NLP, let $\vec{x}$ contain some representation of a trajectory (both spatial and temporal). Dynamic feasibility of the trajectory is specified as constraints in $f(\vec{x})$. Terrain conditions are a cost and/or a constraint ($J(\vec{x})$ and/or $f(\vec{x})$). Traversal time enters into the cost function $J(\vec{x})$. Since we are using a numerical solver, constraints can not be explicitly satisfied at every point on the trajectory. Instead, we satisfy these constraints at discrete points throughout the trajectory, known as collocation points. The collocation point spacing is chosen to be tight enough to ensure that the constraints would be sufficiently satisfied globally, but sparse enough to facilitate rapid computation.

So far we have talked about the vector $\vec{x} \in \mathbb{R}^n$ being some representation of a trajectory, with the $\mathbb{R}^n$ space mapping into a suitable approximation to the space of all trajectories. We need to represent $\theta(s)$ and $v(s)$ (Figure 4) in a finite-dimensional vector space, while assuring that $\theta(s)$ is C^1 and $v(s)$ is C^0 (continuous speed and steering angle profiles). Thus, we represent $\theta(s)$ as a quadratic spline and $v(s)$ as a linear spline (piecewise linear). Since constraints are only satisfied at the collocation points, we only need to evaluate $\theta(s)$ and $v(s)$ at those points. To do this, we have derived matrices that satisfy

$$\begin{array}{rcl} \theta\big|_{s\in\{s_{\text{collocation}}\}} & = & V_{\text{val}}\vec{x_\theta} + \theta_0 \\ \frac{d\theta}{ds}\Big|_{s\in\{s_{\text{collocation}}\}} & = & V_{\text{d1}}\vec{x_\theta} \\ \frac{d^2\theta}{ds^2}\Big|_{s\in\{s_{\text{collocation}}\}} & = & V_{\text{d2}}\vec{x_\theta} \\ v\big|_{s\in\{s_{\text{collocation}}\}} & = & W_{\text{val}}\vec{x_v} + v_0 \\ \frac{dv}{ds}\Big|_{s\in\{s_{\text{collocation}}\}} & = & W_{\text{d1}}\vec{x_v} \end{array} \tag{3}$$

Since these V and W matrices can all be computed off-line, this produces a very fast method for computing the values of our flat variables at the collocation points.

All the various costs and constraints in the planning problem are expressed in the $f(\vec{x})$ and $J(\vec{x})$ expressions in (3). Figure 5 illustrates how the user-specified costs and constraints (expressed in system variables) map into those specified as a function of $\vec{x}$, used by the solver.

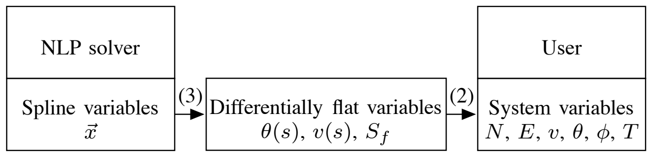

NLP solver
Spline variables $\vec{x}$
(3)
Differentially flat variables $\theta(s)$, $v(s)$, S_f
(2)
User
System variables N, E, v, θ, ϕ, T

Fig. 5. An illustration of how costs and constraints are evaluated by the solver. The user expresses everything in terms of the system variables, which are obtained from the flat variables (2). These, in turn, are obtained from the spline-defining coefficients $\vec{x}$ (3).

The main contributor to the cost function $J(\vec{x})$ is the time-optimality term that describes traversal time:

$$T = S_f \int_0^1 \frac{1}{v(s)}ds \tag{4}$$

This is linearly combined with steering and acceleration "effort" terms:

$$\begin{array}{rcl} E_{\dot\phi} & \equiv & \int_0^1 \dot\phi(s)^2 ds \\ E_a & \equiv & \int_0^1 a(s)^2 ds \end{array} \tag{5}$$

These effort terms are necessary to promote smoother driving. Even though it may be feasible and optimal to decelerate hard with the brake pedal depressed all the way to avoid an obstacle, system lags and model inaccuracies make this dangerous. Thus, we penalize abrupt changes in speed and steering angle. The weights on each contributing term to the cost function were chosen empirically in the field.

The constraints are broken into conditions that are to be satisfied at the start of the trajectory (initial constraints), at the end of the trajectory (final constraints) and at the collocation points (trajectory constraints). The initial constraints ensure that the current vehicle pose matches the pose at the start of the plan. The final constraints ensure that the end of the plan lies in a tight neighborhood around the target point. Finally, the trajectory constraints ensure dynamic feasibility and obstacle avoidance. The full list of trajectory constraints is

$$\begin{array}{lccccc} \textbf{Speed} & & & v & < & v_{\text{limit}} \\ \textbf{Acceleration} & a_{\min} & < & a & < & a_{\max} \\ \textbf{Steering} & -\phi_{\max} & < & \phi & < & \phi_{\max} \\ \textbf{Steering speed} & -\dot{\phi}_{\max} & < & \dot{\phi} & < & \dot{\phi}_{\max} \\ \textbf{Rollover} & -\frac{gW}{2h_{\text{cg}}} & < & k\frac{v^2\tan\phi}{L} & < & \frac{gW}{2h_{\text{cg}}} \end{array} \tag{6}$$

In the roll-over constraint expression, W is the track of the vehicle (distance between left and right wheels), h_{cg} is the height of the center of gravity of the vehicle above ground, and g is the acceleration due to gravity. This expression can be derived either by assuming flat ground and considering roll-over due purely to a centripetal force, or by evaluating the lateral acceleration. When evaluating this constraint as roll-over, one has to take into account that on many surfaces side-slip will occur much before roll-over. This necessitates an adjustment factor, k, to compensate [15].

Obstacle avoidance enters into the problem as the speed limit constraint, with the corridor and terrain data processed to produce a combined, discrete map that represents a spatially dependent speed limit. What this means is that the areas outside of the corridor and those that lie inside obstacles have a very low speed limit, and thus any trajectory that goes through those areas is either infeasible or suboptimal. This representation allows the obstacle avoidance and quick traversal conditions to be cleanly combined.

In (6) all of the bounds except for v_{limit} are known offline, and thus are entered into the NLP problem in the solver set-up, before the computation begins. This is not as simple in the case of the speed limit constraint, though, since v_{limit} depends on (N, E). This constraint is thus entered into the problem as

$$0 \quad < \quad v_{\text{limit}} - v \quad < \quad \infty \tag{7}$$

D. Map sampling

The $\vec{x}$-dependent computation of v_{limit} is a non-trivial matter. There are several issues to consider:

1) To ensure the convergence of the solver, the map surface must possess a gradient, rather than being flat (the speed limit constraint has to be convex).
2) The NLP solver expects all functions to have some degree of smoothness, thus the discrete map data have to be interpolated.
3) The vehicle is not a point; it has non-negligible width and length. Thus, obstacle avoidance of *the entire vehicle*, rather than just the center of its rear axle, has to be assured.

1) Ensuring convexity: We are using a convex optimization method which will not work well, or at all, if it is input non-convex functions. One such scenario is a flat speed limit profile (Figure 6).

If at any point in the solution cycle, the current plan runs over the top of any obstacle that does not possess a gradient, the gradient descent-based solver will not be able to determine the proper optimization direction.

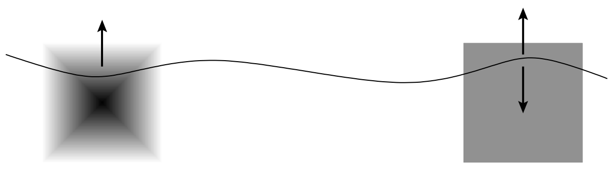


Fig. 6. An illustration of flat obstacles and the numerical non-convexity they cause. The solver will move the curve up, off of the sloped obstacle, but will be unable to determine this adjustment direction in the flat case.

2) Smoothness of the map surface: The numerical optimizer works best when all function supplied to it are at least C^1. In particular, the assumed C^1 properties of the input are utilized to determine when an optimal solution is reached. A discrete map, if used directly, produces a speed limit surface that is not even C^0. Thus an interpolation scheme is *essential* to ensure proper convergence of the planning problem.

3) Growing of obstacles: The vehicle is not a point. Thus when considering the planning problem, the width and length of the vehicle must be taken into account. One way to do this is to enter several obstacle avoidance constraints, at different parts of the vehicle, into the optimization problem. Unfortunately, the extra constraints have an adverse effect on the performance of the computation. Another way to handle this issue is to grow the obstacles, either isotropically or anisotropically, taking into account the orientation of the vehicle (Figure 7).

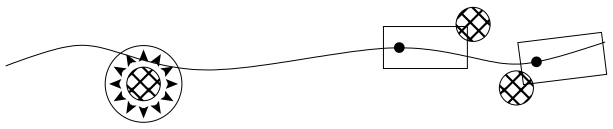


Fig. 7. Different methods of obstacle growing. Isotropic growing, where obstacles are grown equally in all directions is illustrated on the left; anisotropic, where the profile of the vehicle is taken into account when evaluating obstacles is on the right.

4) Map accessor: We developed a map access method that anisotropically samples the map on an approximation to the vehicle profile (the "kernel") to generate a surface that is as close to being C^1 as is desirable. The fundamental approach to this map accessor implementation is a weighted average of the map data. Even though the map data is defined on a discrete-domain, the continuous-domain weighting function allows for accessor continuity:

$$f(x,y) \equiv \frac{\sum_{i,j} w(x - x_i, y - y_j) g[i,j]}{\sum_{i,j} w(x - x_i, y - y_j)} \tag{8}$$

where the discrete map g is being accessed to produce a continuous map f. Note that the weighting function w is defined in reference to the interpolation location (x, y). This endows the same degree of continuity onto f as that of w, assuming that the averaging domain $\{i, j\}$ remains constant. In practice, it is important for $f(x, y)$ to be a function of the map data only in some finite neighborhood of (x, y). This can create potential discontinuities of $f(x, y)$ at the regions where that neighborhood of map data shifts (this happens discretely since the map data is defined on a discrete domain). This issue can be addressed by designing the weighting function w to drop off to negligible values far enough away from (x, y).

Ideally, the vehicle speed would be limited by the lowest speed allowed by the map in any part of the physical vehicle body. Thus we need to design w such that $f(x, y)$ represents a *minimum* of the data in the vehicle profile around (x, y). The minimization and the localization can be accomplished with

$$w(x, y) \equiv w_v(x, y) g[i, j]^{-n} \tag{9}$$

where n is some positive value that controls the minimization strength and $w_v(x, y)$ is a smooth function that is ≈ 0 away from the vehicle profile and ≈ 1 in the vehicle profile. If a rectangular vehicle profile is assumed, $w_v(x, y)$ can be constructed from two one-dimensional functions:

$$w_v(x, y) \equiv w_{\text{long}}(u) w_{\text{lat}}(v) \tag{10}$$

where (u, v) are aligned with the vehicle and $w_{\text{long}}(u)$, $w_{\text{lat}}(v)$ are designed to match the vehicle dimensions. The exact definitions of n, $w_{\text{long}}(u)$ and $w_{\text{lat}}(v)$ can be tuned to achieve the best system performance.

Since $f(x, y)$ is computed very many times during each planning cycle, a slow $f(x, y)$ function very strongly adversely affects the computational performance of the planner. It was found that on Alice's Opteron CPUs, mathematical expressions consisting purely of addition, subtraction, multiplication and division evaluate orders of magnitude faster than those that contain more sophisticated functions, like exponentials and trigonometric or hyperbolic functions. Thus n was chosen to be an integer (2.0), and $w_{\text{long}}(u)$ and $w_{\text{lat}}(v)$ were fit to a rational function.

Another computation time-saving measure was splitting $w_{\text{long}}(u)$ into two non-contiguous sections, one at the front of the vehicle and one at the rear, with each one smaller than half the original. This reduced the number of mathematical operations that needed to take place during each map access cycle. Since this method effectively puts a hole into the middle of the vehicle, it was necessary to make sure that this gap is small enough and the collocation point spacing tight enough to render the missing piece insignificant.

III. Results and Discussion

The previously described planning system was implemented and tested both at the DGC qualifying event and at the DGC race. Of those, the qualifying runs were much more interesting as a testing ground for the planning system, since these contained varied terrain and some obstacles to be avoided. Thus to evaluate the performance of the planning system, we analyze one specific qualifying run. Some snapshots from this run appear in Figure 9.

Throughout the planner development, the convergence speed of the optimizer was a constant concern, but on Alice's 2.2 GHz Opteron CPU, an average rate of 4.8 plans/second was achieved during the qualifying run, 5713 successful planning computations in 1187 seconds. A histogram detailing the spread of these computation times appears in Figure 8.

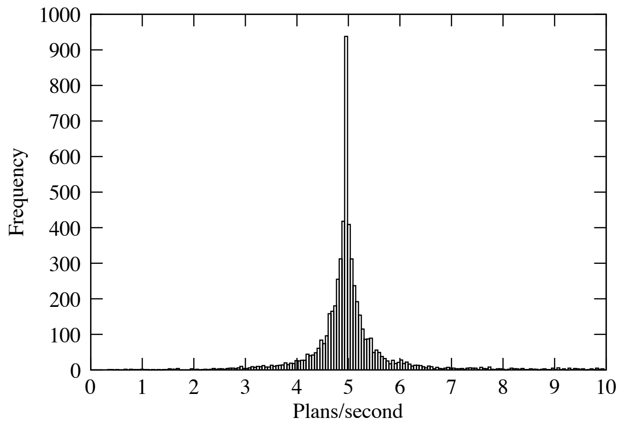


Fig. 8. A histogram representing the distribution of planning computation speed from one of the DGC qualifying runs. Most cycles converged at an average speed of 5 plans/second.

There were a few outliers at the slow end of the distribution, with 11 plans needing more than 1 second to compute. All of these outliers *and* all of the 47 planning attempts that failed to converge (99.18% rate of convergence) occurred at a very few episodes during the qualifying run. This indicates that the errors were not random, but rather were caused by specific events throughout the run. These events were manifestations of the imperfections in the state and terrain sensing systems that feed input to the planner. These imperfections caused aphysical motion of the vehicle and/or of obstacles (due to sensor miscalibration, say), and created infeasible (or nearly so) NLP problems. An example of how this infeasibility might occur is an obstacle that appears suddenly, close and in front of the vehicle. It may not be possible to avoid such an obstacle without violating the deceleration constraint (braking too hard) or a steering constraint (steering too hard, too fast or rolling over). A particularly damaging aspect of these cases is the wasted computation time that is used up by the optimizer before it terminates without producing a valid solution. This can be aggravated further by several cycles in a row failing. In fact, the longest period between valid plans during this qualifying run, 3.05 seconds, occurred when 7 planning cycles in a row failed to converge. This is the main limitation of the planning system, as currently implemented.

Even though the current implementation works well most of the time, and was sufficient for the DGC, a method is needed to quickly recognize an infeasibility and react to it before a lot of computation time is used. This would add the necessary extra element of robustness to the planning algorithm and allow it to be utilized more widely.

References

[1] N. J. Nilsson, *Principles of Artificial Intelligence*. Tioga, 1980.

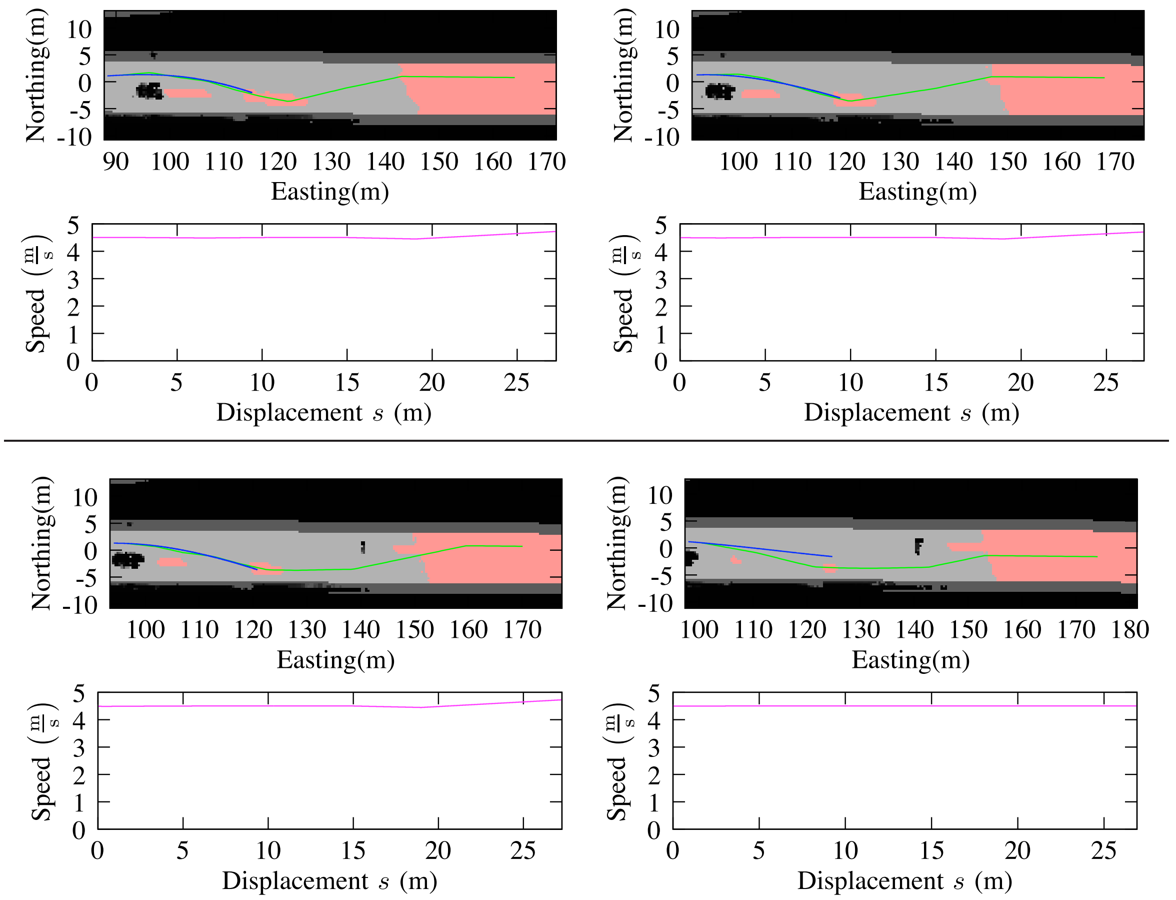


Fig. 9. Non-consecutive planner iterations showing Alice avoid a car at the NQE. The vehicle is at the West (left) edge of each snapshot, and is traveling East (to the right). The planner seed is green and the final solution blue. Grayscale colors are the speed limits in the map, with red representing no-data. Cars south and straight ahead of the vehicle are visible, along with the sensor shadow of no-data for the vehicle straight ahead. We can see distant no-data being treated favorably. We can also see a change of plans to avoid the newly detected second car.

[2] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1994, pp. 3310–3317. [Online]. Available: citeseer.ist.psu.edu/stentz94optimal.html

[3] J. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006. [Online]. Available: citeseer.ifi.unizh.ch/lavalle04planning.html

[5] M. Prado, A. Simon, E. Carabias, A. Perez, and F. Ezquerro, "Optimal velocity planning of wheeled mobile robots on specific paths in static and dynamic environments," *Journal of Robotic Systems*, vol. 20, no. 12, pp. 737–754, Dec 2003.

[6] W. G. Wu, H. T. Chen, and P. Y. Woo, "Time optimal path planning for a wheeled mobile robot," *Journal of Robotic Systems*, vol. 17, no. 11, pp. 585–591, Nov 2000.

[7] Coombs, Murphy, Lacaze, and Legowik, "Driving Autonomously Offroad up to 35 km/h," in *Procs. IEEE Intelligent Vehicles Symposium 2000*, Detroit, USA, Oct. 2000, pp. 186–191.

[8] Z. Shiller and G. R.Y., "Dynamic motion planning of autonomous vehicles," *IEEE Journal of Robotics and Automation*, vol. 7, no. 2, pp. 241–249, Apr 1991.

[9] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems," Ph.D. dissertation, California Institute of Technology, 2003.

[10] M. Milam and R. M. Murray, "A testbed for nonlinear flight control techniques: The caltech ducted fan," in *Proceedings of the 1999 IEEE Conference on Control Applications*, 1999, pp. 345–351.

[11] M. Fliess, J. Lvine, P. Martin, and P. Rouchon, "On differentially flat nonlinear systems," in *In Proc. IFAC-Symposium NOLCOS'92*, 1992, pp. 408–412.

[12] P. E. Gill, W. Murray, and M. A. Saunders, "User's guide for SNOPT 5.3: A fortran package for large-scale nonlinear programming."

[13] B. A. Murtagh and M. A. Saunders, "MINOS 5.5 user's guide."

[14] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "User's guide for NPSOL 5.0: A fortran package for nonlinear programming."

[15] H. Huang, "Lateral slip prevention, detection, and recovery for high-speed autonomous off-road driving," B.S., California Institute of Technology, 2005.