Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments

Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo and James Diebel

The International Journal of Robotics Research 2010 29: 485 originally published online 25 January 2010
DOI: 10.1177/0278364909359210

Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments

Dmitri Dolgov
AI & Robotics Group,
Toyota Research Institute,
Ann Arbor, MI 48105,
USA
ddolgov@ai.stanford.edu

Sebastian Thrun
Michael Montemerlo
James Diebel
Stanford Artificial Intelligence Laboratory,
Stanford University,
Stanford CA 94305,
USA
{thrun, mmde}@ai.stanford.edu

Abstract

We describe a practical path-planning algorithm for an autonomous vehicle operating in an unknown semi-structured (or unstructured) environment, where obstacles are detected online by the robot's sensors. This work was motivated by and experimentally validated in the 2007 DARPA Urban Challenge, where robotic vehicles had to autonomously navigate parking lots. The core of our approach to path planning consists of two phases. The first phase uses a variant of A search (applied to the 3D kinematic state space of the vehicle) to obtain a kinematically feasible trajectory. The second phase then improves the quality of the solution via numeric non-linear optimization, leading to a local (and frequently global) optimum. Further, we extend our algorithm to use prior topological knowledge of the environment to guide path planning, leading to faster search and final trajectories better suited to the structure of the environment. We present experimental results from the DARPA Urban Challenge, where our robot demonstrated near-flawless performance in complex general path-planning tasks such as navigating parking lots and executing U-turns on blocked roads. We also present results on autonomous navigation of real parking lots. In those latter tasks, which are significantly more complex than the ones in the DARPA Urban Challenge, the time of a full replanning cycle of our planner is in the range of 50–300 ms.*

The International Journal of Robotics Research
Vol. 29, No. 5, April 2010, pp. 485–501
DOI: 10.1177/0278364909359210

Figures 1–15, 17 appear in color online: http://ijr.sagepub.com

KEY WORDS—path planning, autonomous driving

1. Introduction

The task of autonomous driving has received much attention from the robotics community, especially in recent years with events such as the DARPA Grand Challenges (Buehler et al. 2005) and the Urban Challenge (Buehler et al. 2008a,b) serving to catalyze research in the field.

In this paper, we focus on the problem of path planning for an autonomous vehicle operating in an unknown environment. We assume the robot has adequate sensing and localization capabilities and must replan online while incrementally building an obstacle map. This scenario was motivated, in part, by the DARPA Urban Challenge (DUC), where vehicles had to freely navigate parking lots. The path-planning algorithm described in this paper was used by the Stanford Racing Team's robot, Junior (Figure 1), in the DUC[1]. In the course of the DUC and the preceding National Qualification Event, Junior demonstrated near-flawless performance in complex free-space path-planning tasks (many involving driving in reverse) such as navigating parking lots, executing U-turns, and dealing with blocked intersections.

One of the main challenges in developing a practical path planner for free-space navigation zones, such as parking lots, arises from the fact that the space of all robot con-

1. See http://www.darpa.mil/grandchallenge/rules.asp.

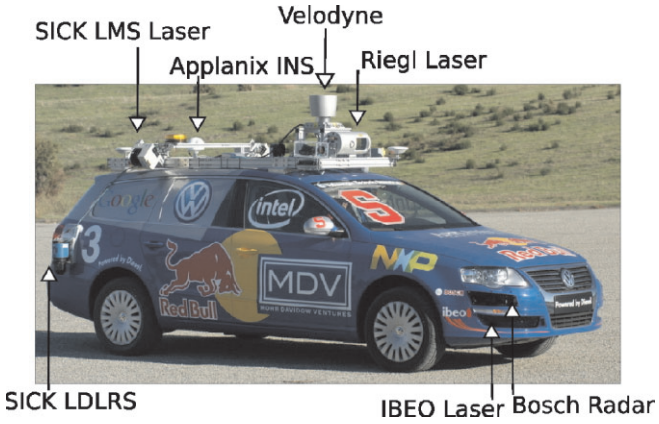


Fig. 1. Junior, our entry in the DUC. Junior is equipped with several LIDAR and RADAR units, and a high-accuracy GPS and inertial-measurement system.

trols (and hence trajectories) is continuous, leading to a complex, continuous-variable optimization problem. Much of the prior work on search algorithms for path planning (Ersson and Hu 2001; Koenig and Likhachev 2002; Ferguson and Stentz 2005; Nash et al. 2007) yields fast algorithms for discrete state spaces, but those algorithms tend to produce paths that are non-smooth and do not generally satisfy the non-holonomic constraints of the vehicle. An alternative approach that guarantees kinematic feasibility of the path is a forward search in continuous coordinates, e.g., using rapidly exploring random trees (Kavraki et al. 1996; LaValle 1998; Plaku et al. 2007). The key to making such continuous search algorithms practical for online implementations lies in an efficient guiding heuristic. Another approach is to directly formulate the path-planning problem as a non-linear optimization problem in the space of controls or in the space of parametrized curves (Cremean et al. 2006). However, in practice, guaranteeing fast convergence of such programs is difficult due to the complex optimization landscape with multiple local minima.

Our path-planning algorithm builds on the existing work discussed above and consists of two main phases. The first phase uses a heuristic search in continuous coordinates to produce kinematically feasible trajectories. While lacking theoretical optimality guarantees, in practice this first step typically produces a trajectory that lies in a neighborhood of the global optimum. The second phase uses numerical optimization in continuous coordinates to locally improve the quality of the solution, producing a path that is at least locally optimal, but in practice often attains the global optimum. This paper builds on earlier versions of our path-planing work (Dolgov et al. 2008).

This two-phase approach to path planning works well in unstructured environments (such as the "free-navigation zones" in the DUC). However, this free-space planner often produces trajectories that are not well suited to semi-structured environments, such as real parking lots. In situations where the environment has strong topological structure, it is important for the robot to observe that structure. For example, cutting across a parking lot is often considered impolite and can be unsafe if the robot shares the environment with human drivers.

In scenarios involving semi-structured environments, we assume that a lane graph representing the topological structure of the environment is known to the robot. The goal is then to bias the planner towards trajectories that observe the structure of the environment, while not restricting the robot to driving on the given lane graph. For example, in parking lots, we would like our robotic vehicle (for the most part) to stay on the given lane graph. However, just as human drivers do, the robot might need to significantly deviate from the graph while performing maneuvers such as turning around, pulling in and out of parking spaces, avoiding other cars, etc.

In situations where a topological lane graph is available to the robot, we extend the two phases of our path-planning algorithm to take that prior information into account. The resulting algorithm seamlessly integrates free-space and graph planning, leading to a faster search and producing final trajectories that are better suited for the environment.

We present results from the DUC as well as from performing navigation in real parking lots. All experiments were performed on a robotic vehicle, shown in Figure 1, equipped with a high-accuracy pose-estimation system (Applanix) and a number of laser range finders, of which the most important one is the 3D LIDAR (Velodyne).

2. Hybrid-state A* Search

The path-planning problem is as follows. The input is an obstacle map (in our implementation, a grid), an initial state of the robot $\mathbf{s}_0 = \langle x, y, \theta \rangle_0$, and a goal state $\mathbf{s}_g = \langle x, y, \theta \rangle_g$, where the $\langle x, y, \theta \rangle$ are the location and orientation of the vehicle, respectively. The desired output is a trajectory (sequence of vehicle states $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n = \mathbf{s}_g$ with a certain resolution δ_s ($\|\mathbf{s}_i - \mathbf{s}_{i-1}\| \leq \delta_s$)) that is safe, near-minimal in length, smooth, and satisfies the kinematic constraints (turning radius) of the car.

Notice that we do not model the vehicle speed in our planning problem. Instead, we assume movement at a constant speed v_0,[2] find a trajectory that is feasible at that speed, and then set the velocity profile of the final trajectory as a function of curvature and proximity to obstacles. As mentioned above, our algorithm proceeds in two phases.

The first phase uses a variant of the well-known A* (Hart et al. 1972) algorithm applied to the 3D kinematic state space

2. In the DUC, we used a maximum velocity of 5 mph for parking lots and generated A* transitions based on the turning radius of our vehicle corresponding to 360° of the steering wheel (equivalent to a 25° turn of the front wheels).

of the vehicle, but with a modified state-update rule that captures continuous-state data in the discrete search nodes of A*. In our implementation, A* used a 4D search space $\langle x, y, \theta, r \rangle$, where the fourth dimension ($r = \{0, 1\}$) represents the current direction of motion (forward or reverse). We use the direction-of-motion bit (r) in our path-cost function to apply penalties for driving in reverse and for switching the direction of motion. The former is a multiplicative penalty that is applied to the length of segments driven in reverse, and the latter is an additive penalty that is applied every time the robot switches directions.

The hybrid-state A* works as follows. Just as in conventional A*, the search space is discretized and a graph is imposed on the grid with centers of cells acting as neighbors in the search graph. However, unlike traditional A*, our hybrid-state A* associates with each grid cell a continuous 3D state of the vehicle. The search then proceeds as follows. Initially, the current continuous state of the vehicle is associated with the initial search node. When a node is popped from the open list of A*, it is expanded by applying several steering actions (in our implementation there are three: max-left; no-turn; max-right) to the continuous state associated with the node, and new children states are generated using a kinematic model of the vehicle. For each of these continuous children states, we compute a grid cell that it falls into. Then, if a node with the same grid cell is already present on the A* open list, and the new cost of the node is lower than the current cost, the continuous state of the node is updated and the node is re-prioritized on the open list.

Clearly, our hybrid A* is similar to Field D* (Ferguson and Stentz 2005). Both address the limitation of classical A* where only the centers of grid cells can be visited. However, the difference between the approaches (as illustrated in Figure 2) is that Field D* is limited to piecewise-linear paths, whereas hybrid A* is not, and because it uses a continuous kinematic model when expanding the nodes, the paths produced by hybrid A* are guaranteed to be drivable.

However, our hybrid-state A* is *not* guaranteed to find the minimal-cost solution because of the discretization of controls and time, as well as the effective pruning of all but one of the continuous-state branches that enter a cell. Furthermore, we are also giving up theoretical guarantees of search completeness by changing the reachable state space, since the search violates the Markov property of the graph (because the transitions from a state are now a function of the continuous coordinates within that state). However, the resulting path is guaranteed to be kinematically feasible (rather than being piecewise-linear as in the case of conventional A*). In practice, hybrid-A* always finds a solution in realistic environments and the obtained solution typically lies in the neighborhood of the global optimum. This allows us to frequently arrive at the globally optimal solution via the second phase of our algorithm (which uses gradient descent to locally improve the path, as described below).

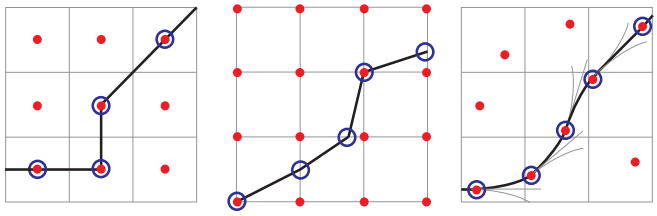


Fig. 2. Graphical comparison of search algorithms. Left: A* associates costs with centers of cells and only visits states that correspond to grid-cell centers. Center: Field D* (Ferguson and Stentz 2005) associates costs with cell corners and allows any linear path from cell to cell. Right: Hybrid A* associates a continuous state with each cell, and the score is the cost of its associated continuous state.

2.1. Heuristics

Our search is guided by two heuristics which are described below and illustrated in Figure 3. These heuristics do not rely on any properties of hybrid-state A* and are also applicable to other search methods (e.g. conventional discrete-state A*).

The first heuristic, which we call *non-holonomic-without-obstacles*, ignores obstacles but takes into account the non-holonomic nature of the car. To evaluate this heuristic, we compute the shortest path to the goal from every point in the 4D space $\langle x, y, \theta, r \rangle$ in some discretized neighborhood of the goal, assuming an obstacle-free environment[3]. Clearly, this cost is an admissible heuristic. The effect of this heuristic is that it assigns high costs to search branches that approach the goal with the wrong heading. Because this heuristic does not depend on run-time sensor information, it can be fully pre-computed offline (and then simply translated and rotated to match the current goal). In our experiments in real driving scenarios, this heuristic provided significant improvements in the number of nodes expanded over the naive 2D Euclidean-distance cost. For example, in the situation shown in the top row of Figure 3, a search with the Euclidean-distance heuristic expands 20,790 nodes, while the non-holonomic-without-obstacles heuristic expands 12,196 nodes. The heuristic is sensitive to the density of obstacles in the environment. In relatively densely populated environments, such as the ones in Figure 3, the benefits are fairly small (a factor of two in Figure 3(b)). In sparser parking lots (such as those in the Urban Challenge), we frequently obtained an order-of-magnitude improvement in the number of expanded nodes.

The second, *holonomic-with-obstacles* heuristic is a dual of the first in that it ignores the non-holonomic nature of the car, but uses the obstacle map to compute the shortest distance to the goal by performing dynamic programming in 2D. In

3. During the Urban Challenge, we used a 160 m × 160 m grid with 1 m × 1 m × 5° resolution.

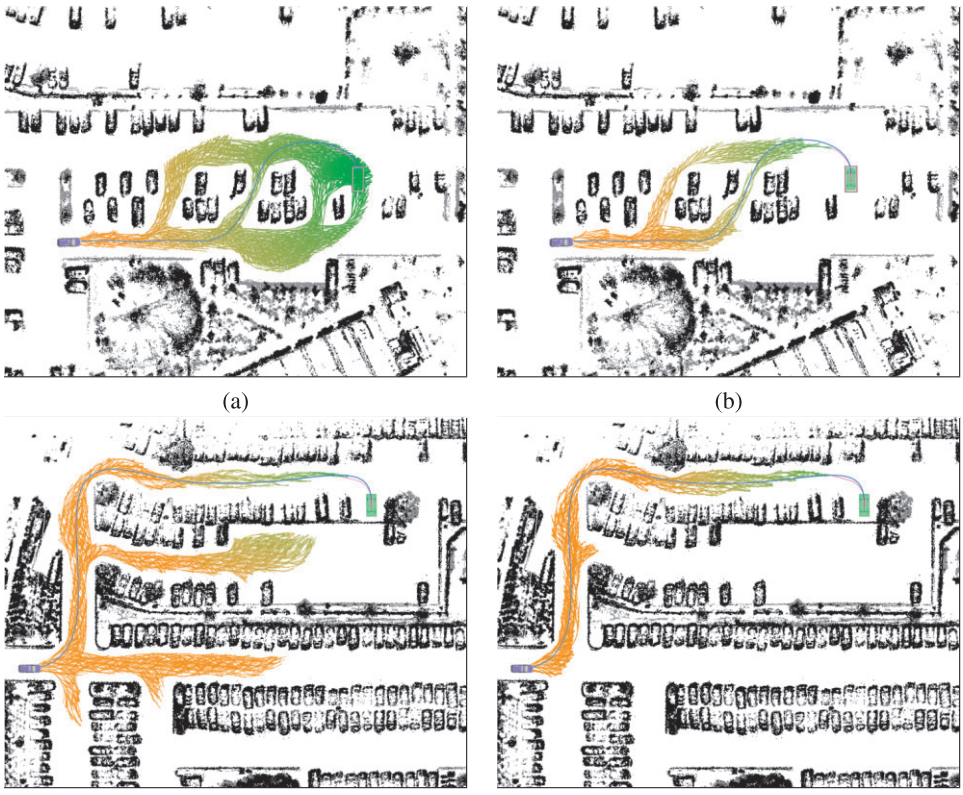


Fig. 3. Search heuristics. Euclidean 2D distance expands 20,790 nodes (a). The non-holonomic-without-obstacles heuristic is better: it expands 12,196 nodes (b), but can lead to wasteful exploration of dead-ends in more complex settings (c), where it expands 37,181 nodes. Combining the latter with the holonomic-with-obstacles heuristic leads to a search tree with 11,302 nodes (d).

essence, given some path-cost function (e.g. collision avoidance) imposed on the configuration space $f_3(x, y, \theta)$, we create a 2D version $f_2(x, y) = \min_\theta f_3(x, y, \theta)$. For example, a 2D state is assumed to be safe (collision cost is 0) if there exists at least one safe 3D state with the same 2D projection. This heuristic is clearly admissible. It captures the 2D geometry of the obstacle map; for example, it discovers all U-shaped obstacles and dead-ends in 2D and then guides the more expensive 4D search away from these areas.

Since both heuristics are mathematically admissible in the A* sense, the maximum of the two can be used. In the scenario shown in the bottom row of Figure 3, a search using only the first (non-holonomic-without-obstacles) heuristic expands 37,181 nodes, while a search that uses both heuristics expands 11,302 nodes.

Heuristics that are essentially equivalent to the ones described above were also independently developed (for a slightly different search method) by the CMU team for the DUC (Likhachev and Ferguson 2008; Urmson et al. 2008). Our non-holonomic-without-obstacles heuristic also parallels the notion of a non-holonomic metric (Laumond et al. 1998; LaValle 2006).

2.2. Analytic Expansions

The forward search described above uses a discretized space of control actions (steering). This means that the search will never reach the exact continuous-coordinate goal state (the accuracy depends on the resolution of the grid in A*). To address this precision issue, and to further improve search speed, we augment the search with analytic expansions based on the Reed–Shepp model (Reeds and Shepp 1990). In the search described above, a node in the tree is expanded by simulating a kinematic model of the car (using a particular control action) for a small period of time (corresponding to the resolution of the grid).

In addition to children generated in such a way, for some nodes an additional child is generated by computing an optimal Reed–Shepp path from the current state to the goal (assuming

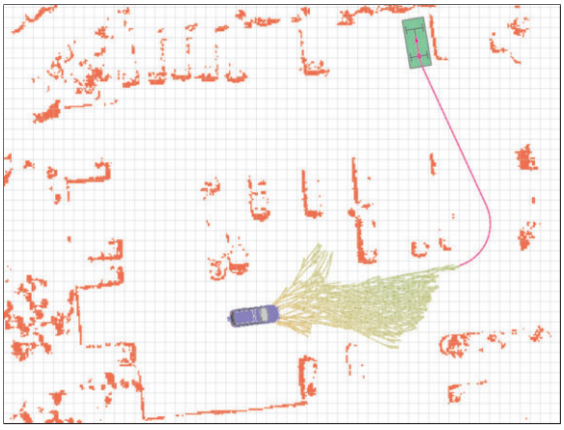


Fig. 4. Analytic Reed–Shepp expansion. The search-tree branches corresponding to short incremental expansions are shown in the yellow–green color range and the Reed–Shepp path is the purple segment leading towards the goal.

an obstacle-free environment). The Reed–Shepp path is then checked for collisions against the current obstacle map, and the child node is only added to the tree if the path is collision-free. Despite the fact that Reed–Shepp expansions are done analytically (i.e. in constant time), in our implementation they are slightly more expensive than the regular forward node expansions. Therefore, it might not be desirable to apply the Reed–Shepp expansion to every node (especially far from the goal, where most such paths are likely to go through obstacles). In our implementation we used a simple selection rule where the Reed–Shepp expansion is applied to one of every N nodes, where N decreases as a function of the cost-to-goal heuristic (leading to more frequent analytic expansions as we get closer to the goal).

A search tree with the Reed–Shepp expansion is shown in Figure 4. The search tree generated by the short incremental expansion of nodes is shown in the yellow-green color range, and the Reed–Shepp expansions is shown as the single purple line leading to the goal. We found that this analytic extension of the search tree leads to significant benefits in both accuracy and planning time.

The computational benefits of using the Reed–Shepp analytic expansions highly depend on the density of obstacles in the environment. On one side of the spectrum are large obstacle-free environments, where Reed–Shepp expansions will find a solution in one step, whereas a forward search will scale with the size of the environment. On the other side of the spectrum are environments with high density of obstacles, where most Reed–Shepp expansions will result in collisions and will have to be pruned. However, we found that in typical driving scenarios this technique leads to noticeable gains in replanning speed. We discuss empirical results of the benefits in Section 5.

2.3. Variable Resolution Search

In typical applications of a forward search it is desirable to use a resolution that is as fine as possible, given the computational limitations. The main reasons for this are: (i) completeness (in our domain, narrow passages might become untraversable if the resolution is too coarse); (ii) optimality (in our domain, coarse resolution often leads to trajectories that oscillate around the optimal solution); and (iii) computational efficiency.

Given our two-phase approach, where the second step locally optimizes the trajectory (as described in Section 3), the second issue (optimality) is not a problem. Indeed, as long as phase one produces a feasible solution in the neighborhood of the optimum, the second phase will find that optimum using a gradient descent.

However, the first issue (completeness) is a concern as is the third one (efficiency). Our implementation of forward search allows for a straightforward realization of variable-resolution search, where the length of the trajectory arcs generated during the node expansion is determined by the amount of obstacle-free space around the node. In our implementation, the grid resolution is held constant, which means that the arc size is always lower-bounded by the size of a grid cell. Because of this, we still cannot guarantee theoretical completeness, but it does help (in practice) in terms of both completeness and speed. It is also possible to combine variable-resolution steps with a variable-resolution grid, leading to stronger completeness guarantees. We have not done this in our implementation because of the overheads associated with maintaining a variable-resolution grid.

Given a point $\langle x, y\rangle$ in the 2D workspace, we compute the distance to the nearest obstacle $d_{\mathcal{O}}(x, y)$ and the distance to the nearest edge of the Generalized Voronoi Diagram (GVD) $d_{\mathcal{V}}(x, y)$. When expanding a node $\langle x, y, \theta, r\rangle$, we consider the size of the free-space region $d_{\mathcal{O}}(x, y) + d_{\mathcal{V}}(x, y)$ around the point $\langle x, y\rangle$ to determine the length of the emanating trajectory arcs (or, equivalently, the time duration for which we unroll the kinematic model). The larger the Voronoi region $d_{\mathcal{O}}(x, y) + d_{\mathcal{V}}(x, y)$, the longer are the arcs generated from node $\langle x, y, \theta, r\rangle$.

In essence, this technique allows us to take big steps forward in wide-open areas, while using a fine resolution in narrow passages. The use of the total "width" of the current Voronoi region $d_{\mathcal{O}}(x, y) + d_{\mathcal{V}}(x, y)$ instead of a more naive metric of proximity to obstacles $d_{\mathcal{O}}(x, y)$ ensures that search effort is not wasted near obstacles that have large free-space regions surrounding them.

3. Trajectory Optimization

As we have already mentioned, the paths produced by hybrid-state A* are often suboptimal and worthy of further improvement. Empirically, we find that such paths are drivable, but

can contain unnatural swerves that require excessive steering. We therefore post-process the hybrid-state A* solution by applying the following two-stage optimization procedure. In the first stage, we formulate a non-linear optimization program on the coordinates of the vertices of the path that improves the length and smoothness of the solution. We solve this optimization using conjugate-gradient (CG) descent, which is a fast numerical-optimization technique. This first optimization essentially moves around the vertices of the path to improve smoothness, but does not explicitly change the path's discretization.

Because the discretization of the resulting path is too coarse for comfortable control of a physical vehicle (≈0.5 m), we then execute a second stage that performs non-parametric interpolation on the output of the first stage using another iteration of a CG. The interpolated paths have higher-resolution discretization (around 5–10 cm) and are suitable for smooth control of the robot.

The first optimization is described below (Section 3.1 defines the core of the approach, and Sections 3.2 and 3.3 present some technical refinements). The interpolation step is discussed in Section 3.4.

3.1. Trajectory Smoothing via Conjugate Gradient

Given a sequence of vertices $\mathbf{x}_i = \langle x_i, y_i \rangle$, $i \in [1, N]$, we define several quantities: (i) $\mathbf{o}_i$, the location of the obstacle nearest to the vertex; (ii) $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$, the displacement vector at the vertex; (iii) $\Delta\phi_i = |\tan^{-1}\frac{\Delta y_{i+1}}{\Delta x_{i+1}} - \tan^{-1}\frac{\Delta y_i}{\Delta x_i}|$, the change in the tangential angle at the vertex. The objective function is defined as:

$$w_o \sum_{i=1}^{N} \sigma_o\left(|\mathbf{x}_i - \mathbf{o}_i| - d_{\max}\right) + w_\kappa \sum_{i=1}^{N-1} \sigma_\kappa\left(\frac{\Delta\phi_i}{|\Delta x_i|} - \kappa_{\max}\right) + w_s \sum_{i=1}^{N-1} (\Delta\mathbf{x}_{i+1} - \Delta\mathbf{x}_i)^2, \quad (1)$$

where $\kappa_{\max}$ is the maximum allowable curvature of the path (defined by the turning radius of the car), and σ_o and σ_κ are penalty functions (empirically, we found quadratic penalties to work well); w_o, w_κ, w_s are weights.

The first term penalizes collisions with obstacles. The second term imposes an upper bound on the instantaneous curvature of the trajectory at every node and enforces the non-holonomic constraints of the vehicle. The third term is a measure of the smoothness of the path.

For a fast implementation of a CG, it is imperative that the cost function has a well-behaved gradient that can be efficiently computed. In our case, the function is differentiable with respect to the coordinates of the vertices $\langle x_i, y_i \rangle$, and an analytical gradient can be computed directly, as described below.

For the collision penalty with a quadratic $\sigma_o = (|\mathbf{x}_i - \mathbf{o}_i| - d_{\max})^2$, we have when $|\mathbf{x}_i - \mathbf{o}_i| \le d_{\max}$:

$$\frac{\partial\sigma_o}{\partial\mathbf{x}_i} = 2(|\mathbf{x}_i - \mathbf{o}_i| - d_{\max})\frac{\mathbf{x}_i - \mathbf{o}_i}{|\mathbf{x}_i - \mathbf{o}_i|}.$$

For the maximum-curvature term at vertex i, we have to take the derivatives with respect to the three points that affect the curvature at point i: $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, and $\mathbf{x}_{i+1}$. For this computation, the change in the tangential angle at node i is best expressed as

$$\Delta\phi_i = \cos^{-1}\frac{\Delta\mathbf{x}_i^T\,\Delta\mathbf{x}_{i+1}}{|\Delta\mathbf{x}_i||\Delta\mathbf{x}_{i+1}|}, \quad (2)$$

and the derivatives of curvature $\kappa_i = \Delta\phi_i/|\Delta\mathbf{x}_i|$ with respect to the three nodes are then:

$$\frac{\partial\kappa_i}{\partial\mathbf{x}_i} = -\frac{1}{|\Delta\mathbf{x}_i|}\frac{\partial\Delta\phi_i}{\partial\cos(\Delta\phi_i)}\frac{\partial\cos(\Delta\phi_i)}{\partial\mathbf{x}_i} - \frac{\Delta\phi_i}{(\Delta\mathbf{x}_i)^2}\frac{\partial\Delta\mathbf{x}_i}{\partial\mathbf{x}_i},$$

$$\frac{\partial\kappa_i}{\partial\mathbf{x}_{i-1}} = -\frac{1}{|\Delta\mathbf{x}_i|}\frac{\partial\Delta\phi_i}{\partial\cos(\Delta\phi_i)}\frac{\partial\cos(\Delta\phi_i)}{\partial\mathbf{x}_{i-1}} - \frac{\Delta\phi_i}{(\Delta\mathbf{x}_i)^2}\frac{\partial\Delta\mathbf{x}_i}{\partial\mathbf{x}_{i-1}},$$

$$\frac{\partial\kappa_i}{\partial\mathbf{x}_{i+1}} = -\frac{1}{|\Delta\mathbf{x}_i|}\frac{\partial\Delta\phi_i}{\partial\cos(\Delta\phi_i)}\frac{\partial\cos(\Delta\phi_i)}{\partial\mathbf{x}_{i+1}},$$

where

$$\frac{\partial\Delta\phi_i}{\partial\cos(\Delta\phi_i)} = \frac{\partial\cos^{-1}(\cos(\Delta\phi_i))}{\partial\cos(\Delta\phi_i)} = \frac{-1}{(1-\cos^2(\Delta\phi_i))^{1/2}}.$$

The derivative of $\cos(\Delta\phi_i)$ is easiest expressed in terms of orthogonal complements:

$$\mathbf{a} \perp \mathbf{b} = \mathbf{a} - \frac{\mathbf{a}^{\mathrm{T}}\mathbf{b}}{|\mathbf{b}|}\frac{\mathbf{b}}{|\mathbf{b}|}. \quad (3)$$

Introducing the following normalized orthogonal complements:

$$\mathbf{p}_1 = \frac{\mathbf{x}_i \perp (-\mathbf{x}_{i+1})}{|\mathbf{x}_i||\mathbf{x}_{i+1}|}; \quad \mathbf{p}_2 = \frac{(-\mathbf{x}_{i+1}) \perp \mathbf{x}_i}{|\mathbf{x}_i||\mathbf{x}_{i+1}|}, \quad (4)$$

we can then express the derivatives as:

$$\frac{\partial\cos(\Delta\phi_i)}{\partial\mathbf{x}_i} = -\mathbf{p}_1 - \mathbf{p}_2; \quad \frac{\partial\cos(\Delta\phi_i)}{\partial\mathbf{x}_{i-1}} = \mathbf{p}_2;$$

$$\frac{\partial\cos(\Delta\phi_i)}{\partial\mathbf{x}_{i+1}} = \mathbf{p}_1. \quad (5)$$

The result of performing the smoothing step described above is shown in Figure 5; the wavy red line is the A* solution and the smooth blue line is the path obtained by CG.

3.2. Guaranteeing Smoother Safety

The smoothing problem formulated above has a potential term responsible for collision avoidance, however, the potential

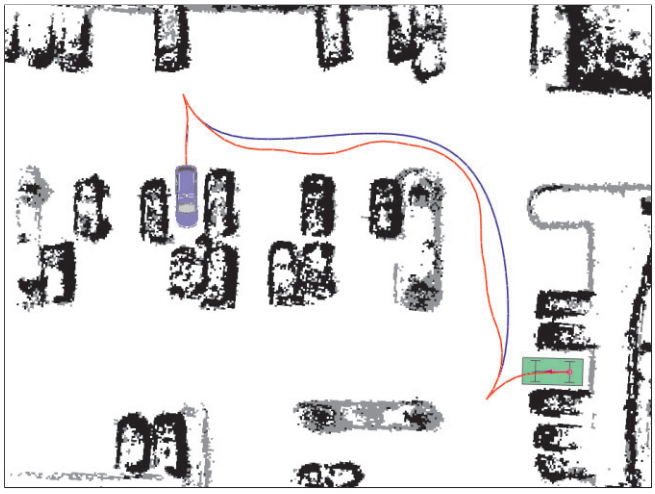


Fig. 5. Hybrid-A* and CG-smoothed paths for a complicated maneuver, involving backing out of and into parking spots. The Hybrid-state A* path is the wavy red line and the CG solution is the smooth blue line.

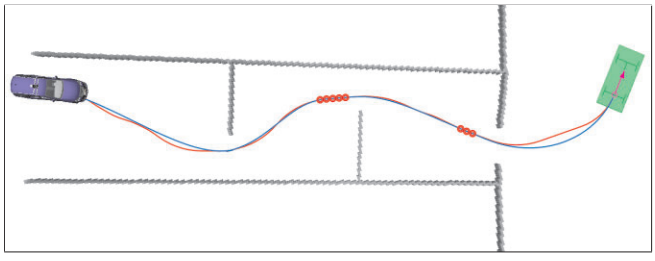


Fig. 6. Anchoring waypoints to guarantee smoother safety.

does *not* guarantee that the solution is collision free. The reason is that the potential attempts (within its effective range) to maximize the distance between every vertex of the trajectory and the nearby obstacles. However, this is not always the right solution. For example, when approaching a narrow gap between obstacles at an angle (as illustrated in Figure 6), the trajectory for the center of the rear axle of the vehicle stays closer to one side of the gap, allowing the car to safely turn into the gap. Unfortunately, because the collision-potential used in the smoother does not model the shape of the vehicle, it is unable to do such precise collision detection and will center the trajectory within the gap, resulting in an unsafe maneuver.

While it is possible to model precise collision detection within the smoother, it is computationally prohibitive. In particular, computing the derivative of the collision cost with respect to the coordinates of the path of the rear axle is a computationally intensive task which has to be performed within the inner loop of CG optimization.

Therefore, we opt for a computationally simpler (albeit less elegant) solution that guarantees that smoother output is collision-free. We use an iterative approach that works, as follows. We run the CG smoother and check its output for collisions. If we find any unsafe states, we anchor them to the A* solution (the smoother is not allowed to modify the coordinates of anchored states) and re-run the smoother. This process is repeated until the smoother output is collision-free. It is guaranteed to converge because A* path is guaranteed to be safe. In the worst case, the smoother will return the same solution as A*, which will only happen under extreme circumstances.

Figure 6 illustrates the process. As before, the wavy red curve shows the path produced by A*, while the straight blue line is the output of the smoother. The circles designate states that ended up being anchored to the A* path (i.e., not allowed to move). Notice that in this rather constrained problem, only a few states are locked down, while the rest of the trajectory is successfully smoothed.

A video corresponding to the situation in Figure 6 is available from the following URL: http://ai.stanford.edu/%7Eddolgov/gpp/anchors.avi.

3.3. Navigation Potential Using the Voronoi Field

One issue that we have omitted from our discussion of path planning so far is the trade-off between proximity to obstacles and trajectory length. A weakness of the path-planning algorithm as described in the previous sections is that it tends to "hug walls", i.e., it will choose the minimal-length trajectory that is collision free, often causing the robot to drive at the minimal collision-free distance to obstacles.

A common way of penalizing proximity to obstacles is to use a potential field (see, e.g., Andrews and Hogan (1983(@), Pavlov and Voronin (1984), Miyazaki and Arimoto (1985) and Khatib (1986)). However, conventional potential fields have a couple of important drawbacks. First, as has been observed by many researchers (see, e.g., Tilove (1990) and Koren and Borenstein (1991)), conventional potential fields create high-potential areas in narrow passages, which can make the cost of traversing these passages prohibitively high. Second, which plays an even more important role in our approach, is computational efficiency. A straightforward potential around an obstacle is typically defined as a function of the distance to the obstacle. This means that in order to compute the value of such a potential field at a given $\langle x, y \rangle$ point, we need to compute the contributions of the potentials from all obstacles that contain $\langle x, y \rangle$ within their effective radius. This can be computationally expensive. A common technique to avoid this issue is to approximate the potential by using only the contribution of the potential from the nearest obstacle, which can be computed much more effectively. However, this introduces another problem. Since we will use the potential within the CG smoother, we need the potential to be smooth and have a well-defined

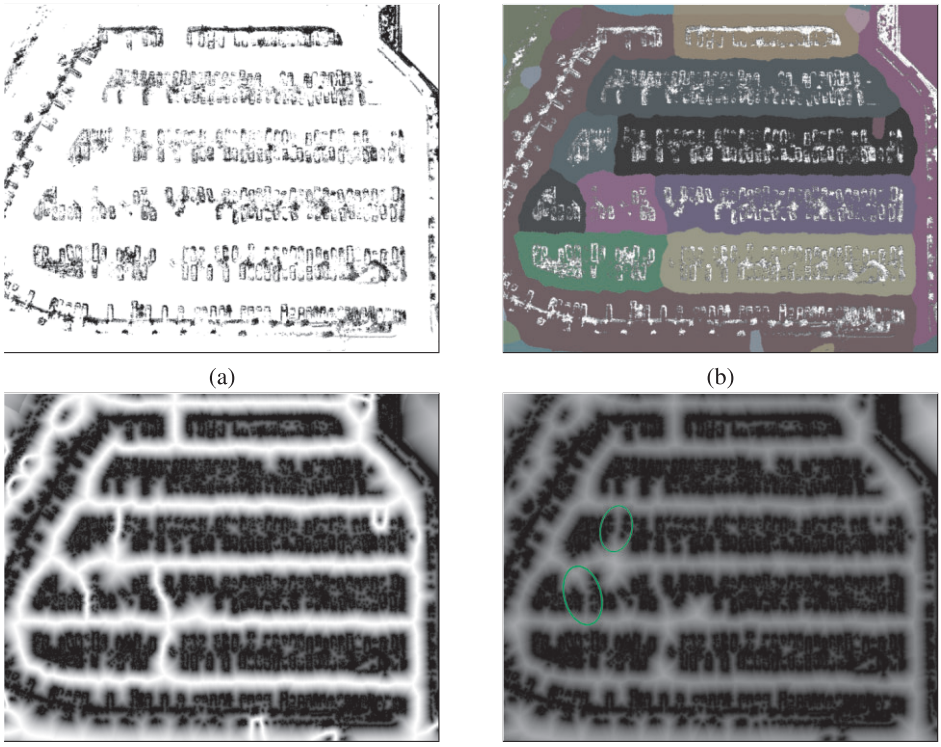


Fig. 7. Best viewed in color. The Voronoi field in a typical parking lot. The obstacle map is shown in (a), the corresponding generalized Voronoi diagram is shown in (b). Compare the resulting Voronoi field shown in (c) to a standard potential field in (d). Notice the high-potential regions in narrow passages in (d).

derivative with respect to $\langle x, y \rangle$. The difficulty stems from the fact that a conventional potential approximated via the distance to the nearest obstacle (as described above) has an ill-defined gradient on the edges of the Voronoi diagram of the obstacle map.

To address these issues, we introduce a potential, termed the *Voronoi field*, that rescales the field based on the geometry of the workspace. This potential allows precise navigation in narrow passages while also effectively repelling the robot from obstacles in wider areas. Furthermore, the potential can be efficiently computed and has a well-defined, continuous derivative everywhere in $\langle x, y \rangle$.

Given a point $\langle x, y \rangle$ in the 2D workspace, we compute the distance to the nearest obstacle ($d_{\mathcal{O}}$) and the distance to the nearest edge of the GVD ($d_{\mathcal{V}}$). The potential is then defined in terms of these distances as:

$$\rho_V(x, y) = \left(\frac{\alpha}{\alpha + d_{\mathcal{O}}(x, y)}\right)\left(\frac{d_{\mathcal{V}}(x, y)}{d_{\mathcal{O}}(x, y) + d_{\mathcal{V}}(x, y)}\right) \times \frac{(d_{\mathcal{O}} - d_{\mathcal{O}}^{\max})^2}{(d_{\mathcal{O}}^{\max})^2}, \quad (6)$$

where $\alpha > 0$, $d_{\mathcal{O}}^{\max} > 0$ are constants that control the falloff rate and the maximum effective range of the field, respectively. The expression in (6) is for $d_{\mathcal{O}} \leq d_{\mathcal{O}}^{\max}$; otherwise, we let $\rho_V(x, y) = 0$.

This potential has the following properties: (i) it is zero when $d_{\mathcal{O}} \geq d_{\mathcal{O}}^{\max}$; (ii) $\rho_V(x, y) \in [0, 1]$ and is continuous on (x, y), since we cannot simultaneously have $d_{\mathcal{O}} = d_{\mathcal{V}} = 0$; (iii) it reaches its maximum only within obstacles; and (iv) it reaches its minimum only on the edges of the GVD.

The key advantages of the Voronoi field are as follows. Firstly, the field is scaled in proportion to the total available clearance for navigation. As a result, even narrow openings remain navigable with low costs, which is not always the case for standard potential fields. Secondly, the field at any point $\langle x, y \rangle$ only depends on the distance to the nearest obstacle and the nearest point on the Voronoi edge, which means that the value can be computed efficiently using one of known efficient algorithms for computing distance fields on grids[4]. Finally, the Voronoi field has a well-defined derivative, which is a requirement for the CG smoother.

Figure 7 illustrates the Voronoi field in a parking lot. Figure 7(a) shows the obstacle map, Figure 7(b) shows the corre-

4. If an additional increase in speed is desired, instead of using Euclidean distance one might use a quasi-Euclidean metric (e.g. Rosenfeld and Pfaltz 1966) as an approximation.

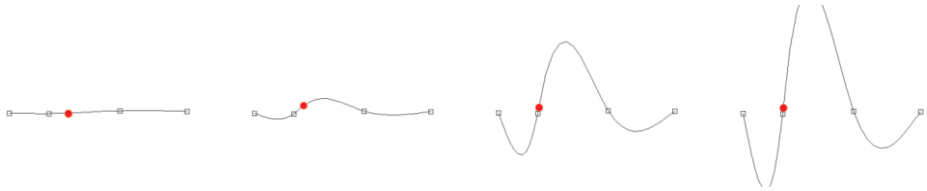


Fig. 8. A piecewise-cubic fit between five points is shown. Parametric interpolation using cubic splines is sensitive to noise in input; as one of the points moves closer to its neighbor, the parametric curve exhibits a large oscillation.

sponding GVD, and Figure 7(c) gives the 2D projection of the resulting Voronoi field. Notice that narrow passages between obstacles that are close to each other are not blocked off by the potential, and there is always a continuous $\rho_V = 0$ path between them. Compare this to an example of a conventional potential field $\rho(x, y) = \alpha(\alpha + d_{\mathcal{O}}(x, y))^{-1}$ shown in Figure 7(d), which has high-potential regions in narrow passages between obstacles.

To make use of the Voronoi field in the smoother, we simply add the following term to the objective function of the smoother (1):

$$w_\rho \sum_{i=1}^{N} \rho_V(x_i, y_i),$$

where w_ρ is the weight associated with the Voronoi field.

Finally, we have to compute the derivative of the Voronoi field with respect to the coordinates $\mathbf{x}_i$ of the trajectory. We have when $d_{\mathcal{O}} \leq d_{\mathcal{O}}^{\max}$:

$$\frac{\partial \rho_V}{\partial \mathbf{x}_i} = \frac{\partial \rho_V}{\partial d_{\mathcal{O}}} \frac{\partial d_{\mathcal{O}}}{\partial \mathbf{x}_i} + \frac{\partial \rho_V}{\partial d_{\mathcal{V}}} \frac{\partial d_{\mathcal{V}}}{\partial \mathbf{x}_i},$$

$$\frac{\partial d_{\mathcal{O}}}{\partial \mathbf{x}_i} = \frac{\mathbf{x}_i - \mathbf{o}_i}{|\mathbf{x}_i - \mathbf{o}_i|}, \quad \frac{\partial d_{\mathcal{V}}}{\partial \mathbf{x}_i} = \frac{\mathbf{x}_i - \mathbf{v}_i}{|\mathbf{x}_i - \mathbf{v}_i|},$$

$$\frac{\partial \rho_V}{\partial d_{\mathcal{V}}} = \frac{\alpha}{\alpha + d_{\mathcal{O}}} \frac{(d_{\mathcal{O}} - d_{\mathcal{O}}^{\max})^2}{(d_{\mathcal{O}}^{\max})^2} \frac{d_{\mathcal{O}}}{(d_{\mathcal{O}} + d_{\mathcal{V}})^2},$$

$$\frac{\partial \rho_V}{\partial d_{\mathcal{O}}} = \frac{\alpha}{\alpha + d_{\mathcal{O}}} \frac{d_{\mathcal{V}}}{d_{\mathcal{O}} + d_{\mathcal{V}}} \frac{(d_{\mathcal{O}} - d_{\mathcal{O}}^{\max})}{(d_{\mathcal{O}}^{\max})^2} \times \left(\frac{-(d_{\mathcal{O}} - d_{\mathcal{O}}^{\max})}{\alpha + d_{\mathcal{O}}} - \frac{d_{\mathcal{O}} - d_{\mathcal{O}}^{\max}}{d_{\mathcal{O}} + d_{\mathcal{V}}} + 2 \right),$$

where $\mathbf{v}_i$ is a 2D vector of the coordinates of the point on the edge of the GVD that is closest to the vertex i. We compute the nearest obstacle $\mathbf{o}_i$ and the nearest GVD-edge point $\mathbf{v}_i$ by maintaining a kd-tree of all obstacle points and GVD-edge points and updating the mapping of vertices to their nearest neighbors, when necessary, within the inner loop of a CG.

We should note that the use of Voronoi diagrams and potential fields has long been proposed in the context of robot motion planning. For example, Voronoi diagrams can be used to derive skeletonizations of the free space as proposed by Choset and Burdick (2000). However, navigating along the Voronoi graph is not possible for a non-holonomic car.

Navigation functions (Koditschek 1987; Rimon and Koditschek 1992) as well as Laplace potentials (Connolly et al. 1990) are also similar to our Voronoi field in that they construct potential functions free of local minima for global navigation. We do not use the Voronoi field for global navigation. However, we observe that for workspaces with convex obstacles, the Voronoi field can be augmented with a global attractive potential, yielding a field that has no local minima and is therefore suitable for global navigation.

3.4. Trajectory Interpolation

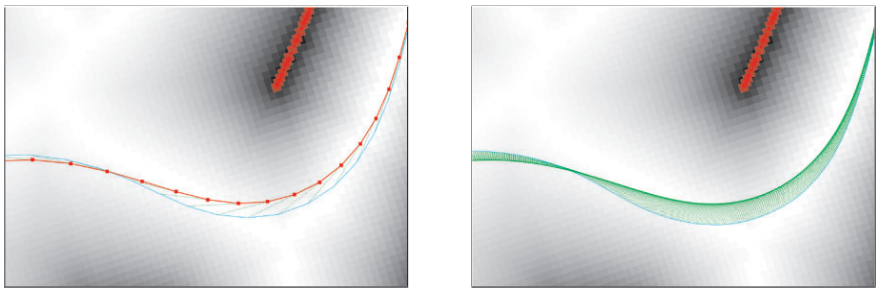
Using the CG smoothing described above, we obtain a path that is much smoother than the A* solution, but it is still piecewise linear, with a significant distance between vertices (around 0.5–1 m in our implementation). This can lead to abrupt steering when used on a physical vehicle. To address this, we further smooth the path by interpolating between the vertices of the CG solution.

Many parametric interpolation techniques are sensitive to noise in the input. For example, cubic splines exacerbate any such noise and can lead to arbitrarily large oscillations in the output as vertices in the input are moved closer to each other. This effect is illustrated in Figure 8, where a cubic spline between five points is shown. As one of the input points (shown as the large red dot) is moved closer to its neighbor, the parametric curve exhibits very large deviations from the input.

Therefore, we use non-parametric interpolation, where we super-sample the path, and then use CG to minimize the curvature of the path, while holding the original vertices fixed. The result of interpolating the path in Figure 9(a) is shown in Figure 9(b) (the paths for both the front and the rear axles are shown).

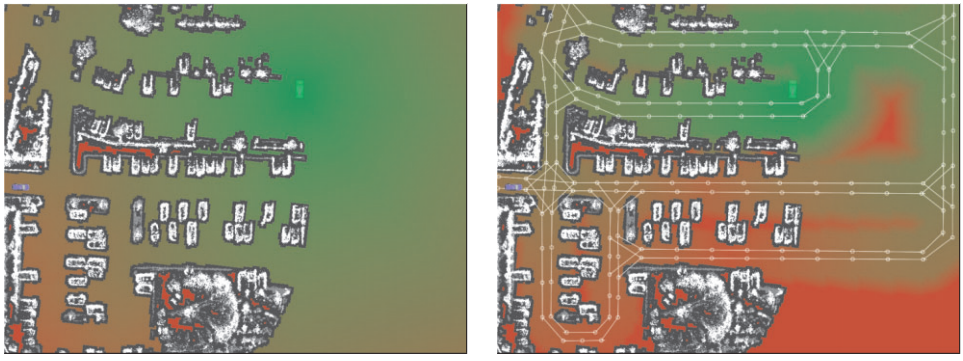
4. Graph-guided Path Planning in Semi-structured Environments

The free-space path planner described above is well-suited for unstructured (or poorly structured) environments, such as the



(a) (b)

Fig. 9. Non-parametric interpolation. The input path is shown in (a), the result of the interpolation is shown in (b). The planned paths of both the front and the rear axles are shown.



(a) (b)

Fig. 10. Best viewed in color. Comparison of the 2D heuristic for the same environment with (b) and without (a) using a lane-network graph. The green (light) color component decreases and the red (dark) component increases with distance to the goal. Notice the difference in the costs between (a) and (b) in the obstacle-free region on the right. The lane network is shown in white for the case that uses it (b).

free-navigation zones in the DUC. However, in better structured environments, it becomes important to realize advanced driving behaviors, such as observing the topological structure of parking lots (not cutting across), staying in the correct lane, passing oncoming traffic on the correct side of the road, etc.

To realize these more advanced driving behaviors, we assume that the robot has prior knowledge of a lane-network graph that captures the topological structure of the environment. In this work we ignore the problem of the creation of such graphs. From the perspective of the path planner, it is not important where that information comes from; the graph might be constructed manually, or it might be automatically estimated from sensor data using a variety of techniques (see, e.g., Seo et al. (2009) and Dolgov and Thrun (2009)). An example of a lane-network graph is shown in Figure 10(b).

The role of the lane network in the path-planning process is twofold. First, it modifies the cost function over trajectories, in that paths deviating from the lane network incur a higher cost. Second, since the lane network captures the topological structure of the environment, the graph provides a set of macro actions that are particularly well-tuned to the current environment.

Below we describe the modifications to the two phases (A* and smoothing) of our planning algorithm that allow us to use the lane-network graph to guide the planning process.

4.1. A Search in Semi-structured Environments*

Given a lane network represented as a directed graph $\mathcal{G} = \langle V, E \rangle$ with α_E denoting the angle of edge E, define a distance from a vehicle state $\mathbf{s} = \langle x, y \rangle$ to the graph:

$$\mathcal{D}(\mathcal{G}, \langle \mathbf{x}, \theta \rangle) = \min \left\{ E : |\alpha_E - \theta| < \alpha_{\min} \right\} \mathcal{D}(E, \mathbf{x}), \quad (7)$$

where $\mathcal{D}(E, \mathbf{x})$ is the Euclidean distance between a line segment E and a point $\mathbf{x}$. In words, we define the distance between a state of the vehicle and the lane-network graph as the

Euclidean distance between the $\langle x, y\rangle$ position of the car and the nearest edge, whose orientation is close (within some constant $\alpha_{\min}$) to the orientation of the vehicle.

The traversal cost of A* that takes into account the lane-network graph $\mathcal{G}$ is defined as:

$$C\big(\langle \mathbf{x}, \theta, r\rangle, \langle \mathbf{x} + \Delta\mathbf{x}, \theta + d\theta, r\prime\rangle\big) = dl\big(1 + \delta(1 - r)C_{\text{rev}}\big)$$

$$+ \delta(1 - |r - r\prime|)C_{\text{sw}} + dl\ \delta\big(\mathcal{D}(\langle \mathbf{x}, \theta\rangle, \mathcal{G}) > \mathcal{D}_{\min}\big)C_{\mathcal{G}},$$

where $\delta(i)$ is the Kronecker delta[5]; C_{rev} and C_{sw} are costs for driving in reverse and switching the direction of motion, respectively (this is the same as in the free-space planner); $C_{\mathcal{G}}$ is the multiplicative penalty for deviating from the lane network by a distance of more than some constant $\mathcal{D}_{\min}$ (this is the new cost for the planner using the graph $\mathcal{G}$).

Of the two heuristics described in Section 2.1, the first one (non-holonomic-without-obstacles) remains useful and admissible and does not require any changes. The second heuristic (holonomic-with-obstacles) remains admissible in the case of semi-structured planning (deviating from the lane graph only increases costs), but it provides poor guidance for the search. The heuristic can be easily modified to take into account the graph-modified traversal costs defined above. We retain the admissibility of the heuristic in the 2D (x, y) case by taking the distance to the graph to be the distance from the closest 3D (x, y, θ) state: $\mathcal{D}(\mathbf{x}, \mathcal{G}) = \min_\theta \mathcal{D}(\langle \mathbf{x}, \theta\rangle, \mathcal{G})$.

Figure 10 illustrates the modifications to the holonomic-with-obstacles heuristic. Figure 10(a) shows the heuristic for the free-space planning problem, while Figure 10(b) shows the version using the modified traversal costs that take the lane network into account. Notice the difference in costs in the obstacle-free region on the right. The graph-informed planner correctly estimates the high-cost region (due to topological structure) before it detects obstacles in that region.

The second use of the lane-network graph in A* is that it provides a good set of macro actions, tuned to the topology of the driving environment. We utilize this in the node-expansion step of A* search. As in the case of the free-space planner, when expanding a search node, we generate a fixed number of children nodes by applying a predefined set of control actions to the parent. In addition to that, we also apply a set of macro-actions that use the Reed–Shepp model to analytically compute a path to nearby nodes of the graph $\mathcal{G}$. If the resulting Reed–Shepp curve is collision-free, the corresponding child is added to the search tree.

Figure 11 illustrates the process. The Reed–Shepp curves corresponding to macro-actions to and between nodes of the lane-network graph $\mathcal{G}$ are shown in purple, while free-space expansions are shown in yellow. Notice the area where the blocked lane prevents transitions between graph nodes, resulting in free-space forward expansions around the obstacle, until a collision-free Reed–Shepp solution can be generated to the next graph node.

5. Kronecker delta $\delta(i)$ is defined as $\delta(0) = 1; \delta(i \neq 0) = 0$.

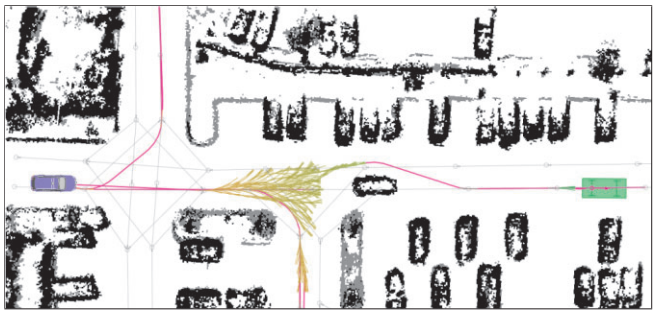


Fig. 11. Best viewed in color. Combining the topological graph search and the free-space path planning. The Reed–Shepp transitions to and between nodes of the lane-network graph $\mathcal{G}$ are shown in purple. The forward free-space expansions are shown in yellow.

4.2. Trajectory Smoothing in Semi-structured Environments

Finally, we describe modifications to the CG smoother for biasing the solution towards curves that lie close the graph $\mathcal{G}$. Given a trajectory $\mathbf{s} = \{\mathbf{x}_i, \theta_i\}$, we minimize the potential:

$$\rho(\mathbf{s}) = \rho_{\text{freespace}}(\mathbf{s}) + w_{\mathcal{G}} \sum_i \mathcal{D}(\langle \mathbf{x}_i, \theta_i\rangle, \mathcal{G}), \tag{8}$$

where $\rho_0(\mathbf{s})$ includes the potential terms used in the free-space planner (Section 3), while the second term provides a penalty for deviating from the graph ($w_{\mathcal{G}}$ is the associated weight).

As before, we need to compute the derivative of the second term in (8) with respect to coordinates of the trajectory $\mathbf{x}_i$.

For a given state $\langle \mathbf{x}_i, \theta_i\rangle$, we find the closest edge $E_{\min}$, whose orientation satisfies the condition $|\alpha_E - \theta_i| < \alpha_{\min}$ and compute the derivative of the distance between $\mathbf{x}_i$ and $E_{\min}$. Let the edge $E_{\min}$ have endpoints $\mathbf{p}_0$ and $\mathbf{p}_1$), and define $\mathbf{p} = \mathbf{p}_0 - \mathbf{p}_1$. For notational convenience, let us also refer to the current trajectory vertex $\mathbf{x}_i$ as $\mathbf{d}$. Our goal then is to compute the derivative of the distance between point $\mathbf{d}$ and edge $\mathbf{p}$.

Let $\mathbf{e}$ be the projection of $\mathbf{d}$ onto $\mathbf{p}$.

$$\mathbf{e} = \mathbf{p}_1 + \beta\mathbf{p}, \quad \beta = \frac{(\mathbf{d} - \mathbf{p}_1)(\mathbf{p})}{\|\mathbf{p}\|^2}. \tag{9}$$

Consider the case where $0 \leq \beta \leq 1$, i.e. $\mathbf{d}$ is closer to the interior of the segment $(\mathbf{p}_0, \mathbf{p}_1)$ than to either of the endpoints (otherwise the problem reduces to the derivative of distance between two points, which was discussed in Section 3).

We seek the derivative of the length of $\mathbf{r} = \mathbf{d} - \mathbf{e}$ (vector from data point $\mathbf{d}$ to the projection point $\mathbf{e}$):

$$\frac{\partial\|\mathbf{r}\|}{\partial\mathbf{d}} = \frac{\mathbf{r}^{\mathrm{T}}}{\|\mathbf{r}\|}\frac{\partial\mathbf{r}}{\partial\mathbf{d}} = \frac{\mathbf{r}^{\mathrm{T}}}{\|\mathbf{r}\|}\frac{\partial(\mathbf{d} - \mathbf{e})}{\partial\mathbf{d}} = \frac{\mathbf{r}^{\mathrm{T}}}{\|\mathbf{r}\|}\left(I - \frac{\partial\beta}{\partial\mathbf{d}}^{\mathrm{T}}\mathbf{p}\right).$$

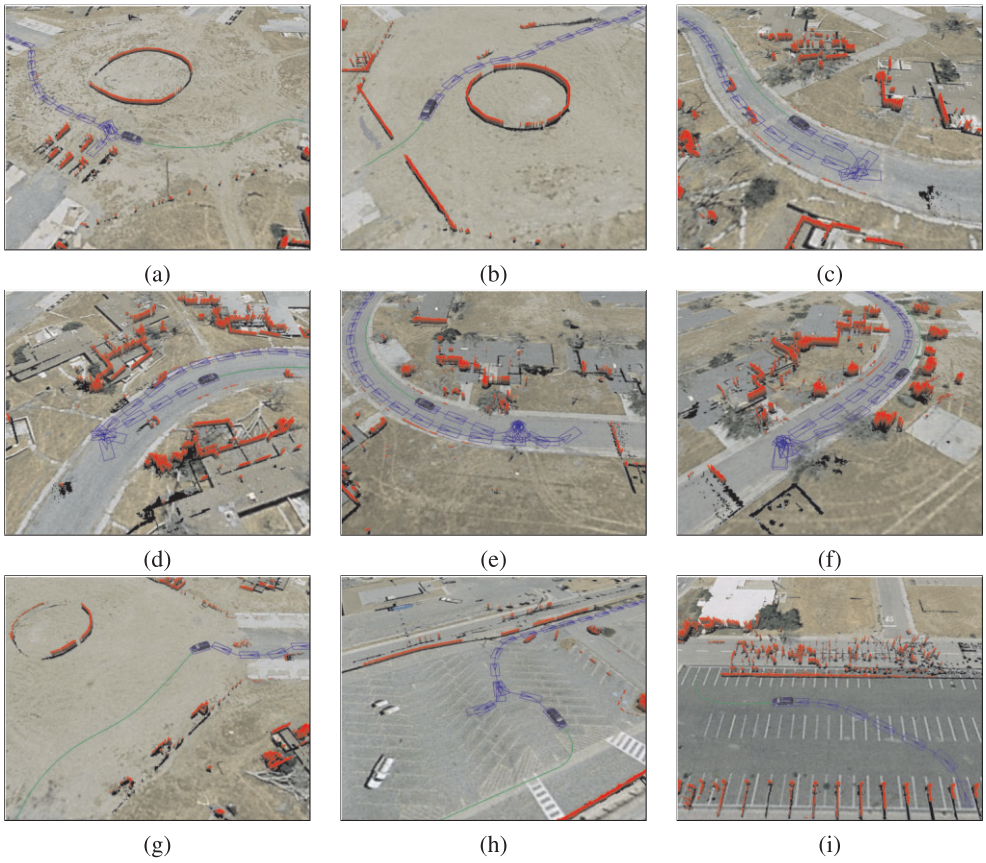


Fig. 12. Best viewed in color. Examples of trajectories generated by our planner and driven during the DUC. Trajectories shown in (a)–(f) were driven as part of the National Qualification Event, while (g)–(i) were executed during the race itself. Paths driven in parking lots and several U-turns on blocked roads are shown.

Expanding the last term

$$\frac{\partial \beta}{\partial \mathbf{d}} = \frac{1}{\|\mathbf{p}\|^2} \frac{\partial \mathbf{d} - \mathbf{p}}{\partial \mathbf{d}} = \frac{1}{\|\mathbf{p}\|^2}(I\,\mathbf{p}) = \frac{\mathbf{p}}{\|\mathbf{p}\|^2},$$

we finally obtain

$$\frac{\partial \|\mathbf{r}\|}{\partial \mathbf{d}} = \frac{1}{\|\mathbf{r}\|}\left(\mathbf{r}^{\mathrm{T}} - \frac{\mathbf{p}^{\mathrm{T}}}{\|\mathbf{p}\|^2}(\mathbf{r}\mathbf{p})\right).$$

5. Results

We present empirical results regarding the performance of our path planner below. Section 5.1 presents results from the DUC (using the free-space planner described in Sections 2 and 3). Section 5.2 presents results from real parking lots, using the graph-guided planner described in Section 4.

We used the following parameters for our planner: the obstacle map was of size 160 m × 160 m with 0.15 cm resolution; A* used a grid of size 160 m × 160 m × 360° with 1 m x–y resolution and 5° resolution for the heading θ. In real-life parking lots, typical running times for a full replanning cycle involving the hybrid A* search, CG smoothing, and interpolation are around 50–300 ms. The time used by the planner in the DUC was significantly lower due to low density of obstacles and relatively small environments.

Several videos illustrating the results discussed in this section are available from http://ai.stanford.edu/%7Eddolgov/gpp/.

5.1. Free-space Planning in the DARPA Urban Challenge

Figure 12 depicts several trajectories driven by Junior (Figure 1) during the DUC. Trajectories in Figures 12(a)–(f) were driven as part of the National Qualifying Event. Trajectories shown in Figures 12(g)–(i) were executed during the race itself. Obstacles from current sensor scan are shown as 3D pillars (red), while the obstacle map obtained by integrating earlier sensor scans is show as 2D (black).

Figures 12(a), (b), and (i) show trajectories driven within free-navigation zones. Figure 12(a) is notable because it involved parking in a spot between real vehicles. Figure 12(b)

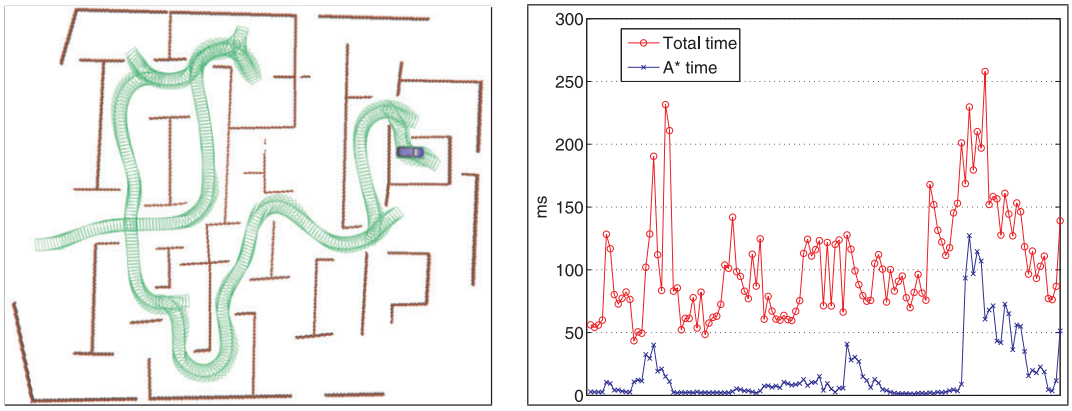


Fig. 13. Left: Trajectory driven in simulation using the free-space version of our planner. The robot had to replan in response to obstacles being detected by its sensors; this explains the apparent sub-optimality of the trajectory. Right: re-planing times for the maze-like environment (total time = A* time + smoothing time.)

shows Junior driving through a parking zone, while two other cars are present in the same zone.

Figure 12(i) is interesting because Junior had to navigate around other cars near the entrance into the zone. A video of the parking task in Figure 12(a) is available at http://ai.stanford.edu/%7Eddolgov/gpp/duc_nqe_park.mpg.

Figures 12(c)–(f) show U-turns on blocked roads that were performed using the free-space planner. Videos of Junior performing U-turns are available at http://ai.stanford.edu/%7Eddolgov/gpp/duc_nqe_uturn.mpg and http://ai.stanford.edu/%7Eddolgov/gpp/duc_nqe_uturn2.mpg.

Figure 12(h) shows a parking task during the DUC race; the maneuver was straightforward, because there were no obstacles in the parking lot. After parking in the designated spot, in accordance with the DUC rules, Junior backed out of the spot before proceeding to the parking-lot exit.

Figure 12(i) shows the start of one of the missions during the DUC race; each DUC mission started with a free-navigation zone, which was traversed using the free-space planner described in this paper.

Most of the path-planning tasks in the DUC were fairly simple. As an example of the performance of our free-space planner in a more complex environment, consider the trajectory shown in Figure 13. This example was generated in simulation; the simulated vehicle was equipped with a single planar laser range finder mounted on the front of the car. Such intentionally poor (simulated) sensing led to frequent replanning as obstacles were incrementally detected; this is the source of the apparent sub-optimality of the path shown in Figure 13. A video showing the robot driving through the environment and replanning as it detects new obstacles and builds an obstacle map in scenario of Figure 13 is available at http://ai.stanford.edu/%7Eddolgov/gpp/maze.mpg.

Figure 14 illustrates the benefits of using the Analytic Reed–Shepp expansions described in Section 2.2. The graph

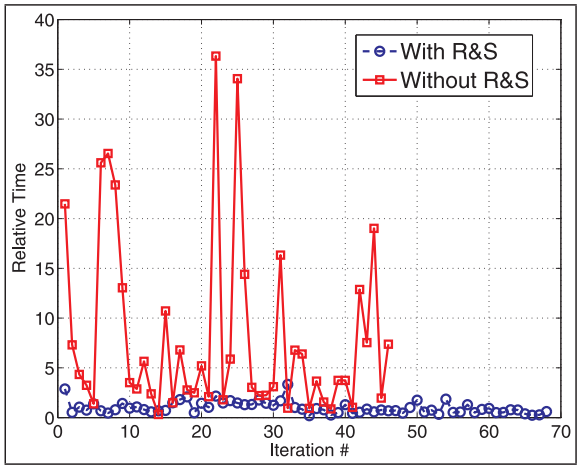


Fig. 14. Comparison of A* with and without Reed–Shepp analytic node expansions. The graph shows data for a typical run in units of relative time, normalized by the average planning time when using Reed–Shepp expansions. The red solid line is the re-planning time without Reed–Shepp expansions, while the blue dashed line is the re-planning time with Reed–Shepp expansion.

shows re-planning time for a representative run in a parking lot with and without the Reed–Shepp expansions. The units are relative time normalized by the average planning time when using Reed–Shepp expansions. As was mentioned earlier in Section 2.2, Reed–Shepp expansions are not strictly guaranteed to improve planning time (because of the constant-time overhead), but in practice lead to noticeable efficiency gains.

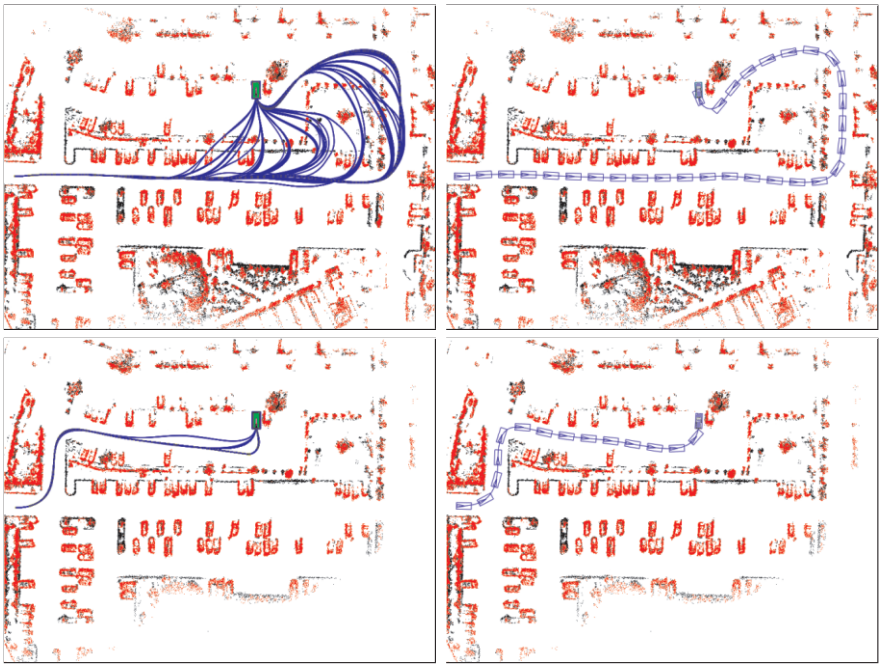


Fig. 15. Free-space path planning (top) versus graph-guided planning (bottom). The left column shows the set of all trajectories generated as the vehicle detects new obstacles and replans. The right column shows the final trajectory driven.

Free-space	4,902 14,102	6,802 42,202	3,402 120,702	7,502 70,302	3,002 86,402	7,304 63,002	10,002 46,002	10,140 43,602	12,102 33,202	15,405 40,002	8,902
Graph	4,902	2,502	7,602								

Fig. 16. Number of nodes expanded by each of the replanning iterations in Figure 15. The free-space search performed 22 iterations, while the graph-guided search performed only 3.

5.2. Path Planning in Semi-structured Environments

This section presents experimental results comparing free-space planning to semi-structured planing guided by a lane-network graph. Experiments presented in this section used A* with the following resolution 1 m x–y in x–y and 5° resolution for the heading θ.

Figure 15 illustrates the benefit of using a topological graph to guide path planning, compared to a free-space planner. The left column of Figure 15 shows the set of all trajectories generated as the vehicle moves towards its goal, detects new obstacles, and replans. The right column shows the final trajectory that was driven. The top row shows the results for a free-space planner, while the bottom row shows the results for a planner guided by a lane-network graph. Due to a strong prior provided by the graph, the latter performs fewer replanning cycles (3 compared with 21) and results in a better final trajectory. Figure 16 shows the number of expanded nodes per replanning cycle for the two cases. Summing up over all the replanning cycles, the graph-guided planner expands a significantly lower total number of nodes (≈15,000 versus ≈635,000).

A video illustrating the difference between free-space and graph-guided planning for the scenario in Figure 15 as well as another parking task is available at http://ai.stanford.edu/%7Eddolgov/gpp/structure.mpg.

Figure 17 shows a few path-planning experiments performed in a real parking lot. All the generated trajectories follow the topological structure of the environment, but in this case this behavior is not due to the lane-graph guidance, but is simply due to the fact that the environment is fairly densely populated. However, even in such densely populated environments, the efficiency gains of using an environment structure for planning are significant when the robot does not have a full obstacle map and must incrementally explore the environment using limited-range sensors (similarly to the example illustrated in Figure 15).

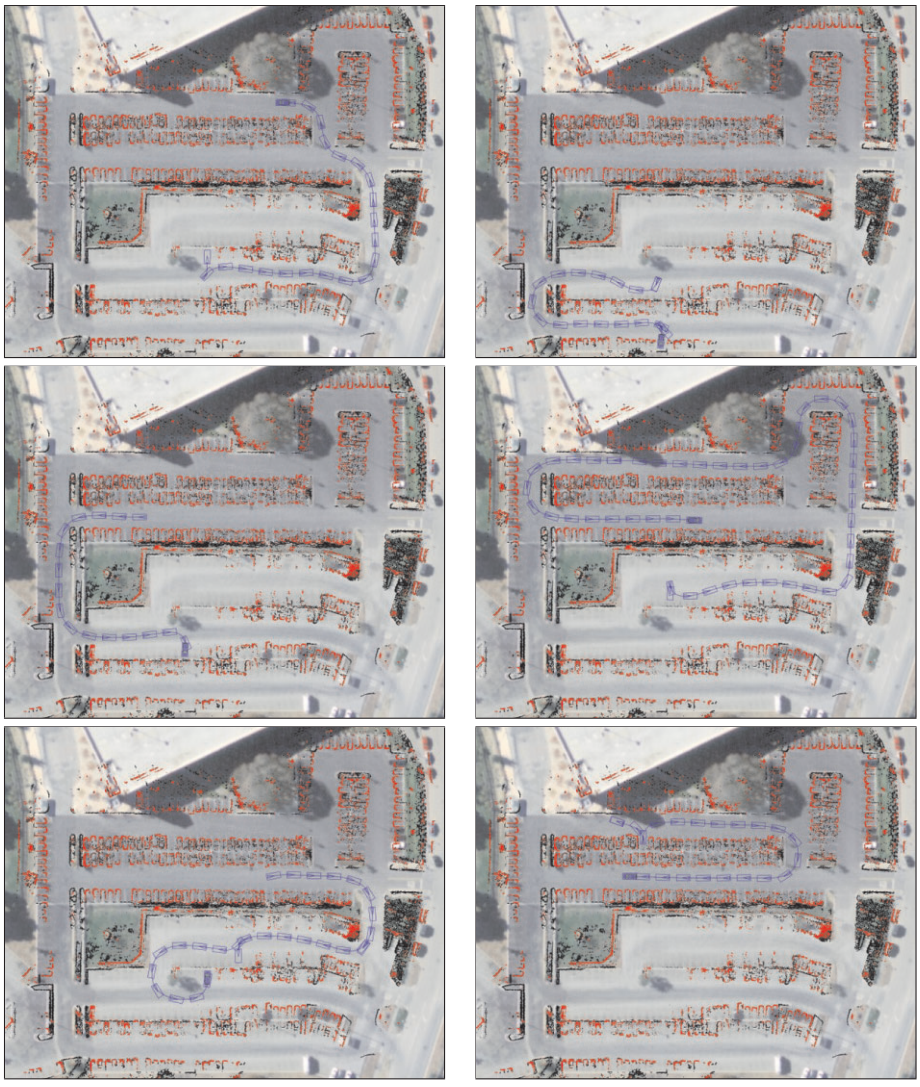


Fig. 17. Best viewed in color. Examples of trajectories generated by our graph-guided planner and driven in a standard parking lot.

6. Related Work

A* search (Hart et al. 1968) is one of the most commonly used search techniques in motion planning. Strong evidence of this is the fact that almost all of the DUC teams used A* (or a variant thereof such as Stentz (1995), Ersson and Hu (2001), Koenig and Likhachev (2002), Ferguson and Stentz (2005), and Nash et al. (2007)) for free-space motion planning.

Our A* heuristics are essentially identical to those independently derived by the CMU Urban Challenge team (Urmson et al. 2008) and are similar to the one used by Team Annieway (Kammel et al. 2008). The latter used simpler heuristics based on RTR (rotation-translation-rotation). In general, such "obstacle-free" heuristics are based on the notion of non-holonomic distance metrics (Laumond et al. 1998; LaValle 2006).

The approach of using discrete global search in combination with post-processing smoothing and the trajectory-optimization technique is also common in motion planing (see Choset et al. (2005) and reference therein). There is a significant body of previous work on analytical global trajectory optimization for obstacle-free environments (see, e.g., Scheuer et al. (1998), Fraichard et al. (1999), and Fraichard et al. (2001)). In environments with obstacles, the term "smooth" is frequently used to refer to trajectories corresponding to C^2 curves, i.e. with continuous derivatives, and does not reflect our notion of having "no wiggles" (see, e.g., Bekris and Kavraki (2007)). Global trajectory optimization in the presence of obstacles often relies on local smoothing methods by

generating trajectories from locally-smooth control actions or geometric primitives such as splines (see, e.g., Lamiraux and Laumond (2001) and Pivtoraiko et al. (2007)).

Most similar to our global trajectory smoothing technique for a non-holonomic vehicle in the presence of obstacles are the techniques used in the Grand Challenge by team Caltech (Cremean et al. 2006) and the one used in the Urban Challenge by team Cornell (Miller et al. 2008). Caltech used a formulation in the space of parametrized splines, while Cornell used a non-parametric curve representation similar to ours, where optimization variables are coordinates of waypoints and associated features. Our approaches differ in the form of the potentials responsible for kinematic-feasibility constraints and obstacle avoidance, as well as our use of the Voronoi field.

Potential fields (see, e.g., Andrews and Hogan (1983(@), Pavlov and Voronin (1984), Miyazaki and Arimoto (1985), and Khatib (1986)) are another common tool in motion planning. Navigation functions (Koditschek 1987; Rimon and Koditschek 1992) and Laplace potentials (Connolly et al. 1990) construct fields free of local minima, which can be used for global navigation. We did not rely on the Voronoi field for global search, but we note that for convex obstacles, the Voronoi field (combined with an attractive potential towards the goal) has no local minima and can be used for global navigation.

A frequent use of Voronoi diagrams in motion planning is to compute a Voronoi skeletonization (Choset and Burdick 2000) of the configuration space and then use the resulting graph for path planning by either searching on the graph or using the graph to inform heuristics. Several teams in the Urban Challenge employed similar uses of Voronoi diagrams. In particular, team Annieway (Kammel et al. 2008) used the Voronoi diagram to inform the A* search by constructing a heuristic where the cost of a state was the sum of the straight-line distance path to the nearest Voronoi edge and the distance along the Voronoi diagram. Our novel contribution is in the use of the Voronoi diagram to create the Voronoi field as described in Section 3.3 with the properties that make it particularly suitable for smoothing via non-linear optimization as described in Section 3.

Acknowledgments

We would like to thank Dirk Haehnel, Jesse Levinson, and other members of the Stanford Racing Team for their help with implementing and testing our planner. We would also like to thank our colleagues Michael Samples and Michael James for useful discussions related to this work. We also gratefully acknowledge DARPA's financial support of our team in the Urban Challenge program.

References

Andrews, J. and Hogan, N. (1983). Impedance control as a framework for implementing obstacle avoidance in a manipulator. *Control of Manufacturing Processes and Robotic Systems*, Boston: ASME: pp. 243–251.

Bekris, K. E. and Kavraki, L. E. (2007). Greedy but safe replanning under kinodynamic constraints. *International Conference on Robotics and Automation*. Piscataway, NJ, IEEE Press, pp. 704–710.

Buehler, M., Iagnemma, K. and Singh, S. (eds) (2005). *The 2005 DARPA Grand Challenge: The Great Robot Race*. Berlin, Springer.

Buehler, M., Iagnemma, K. and Singh, S. (eds) (2008a). Special Issue on the 2007 DARPA Urban Challenge, Part I. *Journal of Field Robotics*, **25**(8): 423–566.

Buehler, M., Iagnemma, K. and Singh, S. (eds) (2008b). Special Issue on the 2007 DARPA Urban Challenge, Part II, *Journal of Field Robotics*, **25**(9): 567–724.

Choset, H. and Burdick, J. (2000). Sensor-based exploration: The hierarchical generalized Voronoi graph. *The International Journal of Robotics Research*, **19**: 96–125.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E. and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA, MIT Press.

Connolly, C., Burns, J. and Weiss, R. (1990). Path planning using Laplace's equation. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2102–2106.

Cremean, L. B., Foote, T. B., Gillula, J. H., Hines, G. H., Kogan, D., Kriechbaum, K. L., Lamb, J. C., Leibs, J., Lindzey, L., Rasmussen, C. E., Stewart, A. D., Burdick, J. W. and Murray, R. M. (2006). Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, **23**(9): 777–810.

Dolgov, D. and Thrun, S. (2009). Autonomous driving in semi-structured environments: Mapping and planning. *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA-09)*, Kobe, Japan.

Dolgov, D., Thrun, S., Montemerlo, M. and Diebel, J. (2008). Practical search techniques in path planning for autonomous driving. *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, Chicago, IL. Menlo Park, CA, AAAI.

Ersson, T. and Hu, X. (2001). Path planning and navigation of mobile robots in unknown environments. *IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Ferguson, D. and Stentz, A. (2005). Field d*: An interpolation-based path planner and replanner. *Proceedings of the International Symposium on Robotics Research (ISRR)*.

Fraichard, T. and Ahuactzin, J.-M. (2001). Smooth path planning for cars. *Proceedings of the IEEE Interna-*

tional Conference on Robotics and Automation, pp. 3722–3727.

Fraichard, T., Scheuer, A., and Desvigne, R. (1999). From Reeds and Shepp's to continuous-curvature paths. *Proceedings of the IEEE International Conference on Advanced Robotics*, pp. 1025–1035.

Hart, P. E., Nilsson, N. J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, **4**(2): 100–107.

Hart, P. E., Nilsson, N. J. and Raphael, B. (1972). Correction to "A formal basis for the heuristic determination of minimum cost paths". *ACM SIGART Bulletin*, **37**: 28–29.

Kammel, S., Ziegler, J., Pitzer, B., Werling, M., Gindele, T., Jagzent, D., Schröder, J., Thuy, M., Goebl, M., Hundelshausen, F. v., Pink, O., Frese, C., and Stiller, C. (2008). Team Annieway's autonomous system for the 2007 DARPA Urban Challenge. *Journal of Field Robotics*, **25**(9): 615–639.

Kavraki, L., Svestka, P., Latombe, J.-C. and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, **12**(4): 566–580.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, **5**(1): 90–98.

Koditschek, D. E. (1987). Exact robot navigation by means of potential functions: some topological considerations. *IEEE International Conference on Robotics and Automation (ICRA)*.

Koenig, S. and Likhachev, M. (2002). Improved fast replanning for robot navigation in unknown terrain. *IEEE International Conference on Robotics and Automation (ICRA)*.

Koren, Y. and Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. *IEEE International Conference on Robotics and Automation (ICRA)*.

Lamiraux, F. and Laumond, J.-P. (2001). Smooth motion planning for car-like vehicles. *IEEE Transactions On Robotics and Automation*, **17**(4): 498–501.

Laumond, J.-P., Sekhavat, S. and Lamiraux, F. (1998). Guidelines in nonholonomic motion planning for mobile robots. *Robot Motion Planning and Control*, Laumond, J.-P. (ed) Berlin, Springer-Verlag, pp. 1–53.

LaValle, S. (1998). Rapidly-exploring random trees: a new tool for path planning.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge, Cambridge University Press. Also available at http://planning.cs.uiuc.edu/.

Likhachev, M. and Ferguson, D. (2008). Planning long dynamically-feasible maneuvers for autonomous vehicles. *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland.

Miller, I., Campbell, M., Huttenlocher, D., Kline, F.-R., Nathan, A., Lupashin, S., Catlin, J., Schimpf, B., Moran, P., Zych, N., Garcia, E., Kurdziel, M., and Fujishima, H. (2008). Team cornell's skynet: robust perception and planning in an urban environment. *Journal of Field Robotics*, **25**(8): 493–527.

Miyazaki, F. and Arimoto, S. (1985). Sensory feedback for robot manipulators. *Journal of Robotic Systems*, **2**(1): 53–71.

Nash, A., Daniel, K., Koenig, S. and Felner, A. (2007). Theta*: any-angle path planning on grids. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Menlo Park, CA, AAAI Press, pp. 1177–1183.

Pavlov, V. and Voronin, A. N. (1984). The method of potential functions for coding constraints of the external space in an intelligent mobile robot. *Soviet Automatic Control*, **17**(6): 45–51.

Pivtoraiko, M., Knepper, R. A. and Kelly, A. (2007). *Optimal, Smooth, Nonholonomic Mobile Robot Motion Planning in State Lattices*. Technical Report CMU-RI-TR-07-15, Robotics Institute, Pittsburgh, PA.

Plaku, E., Kavraki, L. and Vardi, M. (2007). Discrete search leading continuous exploration for kinodynamic motion planning. *Robotics: Science and Systems*.

Reeds, J. A. and Shepp, L. A. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, **145**(2): 367–393.

Rimon, E. and Koditschek, D. E. (1992). Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, **8**(5): 501–518.

Rosenfeld, A. and Pfaltz, J. L. (1966). Sequential operations in digital picture processing. *Journal of the ACM*, **13**(4): 471–494.

Scheuer, A. and Laugier, C. (1998). Planning sub-optimal and continuous-curvature paths for car-like robots. *Proceedings of the IEEE-RSJ International Conference on Intelligent Robots and Systems*, pp. 25–31.

Seo, Y.-W., Ratliff, N. and Urmson, C. (2009). Self-supervised aerial image analysis for extracting parking lot structure. *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-09)*. Menlo Park, CA, AAAI Press.

Stentz, A. (1995). The focussed d* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1652–1659.

Tilove, R. (1990). Robotics and automation. *IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 566–571.

Urmson, C. et al. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, **25**(8): 425–466.