2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Autonomous Driving in Semi-Structured Environments: Mapping and Planning.

Dmitri Dolgov[1], Sebastian Thrun[2]

Abstract— **We consider the problem of autonomous driving in semi-structured environments (e.g., parking lots). Such environments have strong topological structure (graphs of drivable lanes), but maneuvers with significant deviations from those graphs are valid and frequent. We address two main challenges of operating in such environments: i) detection of topological structure from sensor data, and ii) using that structure to guide path planning. We present experimental results on both of these topics, demonstrating robust estimation of lane networks in parking lots and the benefits of using these topological networks to guide path planning.**

I. Introduction

Autonomous driving is a long-standing problem in robotics, interest in which has spiked in recent years in part due to the series of DARPA Grand Challenges [1], [2].

Much of the existing work in autonomous driving focuses either on highly structured environments, such as highways or city streets [4], [22], [23], [20], or on unstructured off-road driving [10], [19], [3], [18], [1], [11]. In the former case, it is typically assumed that there is a topological graph (i.e., lane-network graph) imposed on the environment, and the vehicle is constrained to drive on the graph with only small deviations admissible. In the case of unstructured driving, the robot is not constrained by a graph and is free to choose any path, subject to safety and kinodynamic constraints.

In reality, there are also many environments that are *semi-structured*, i.e., there is natural topological graph structure (which may or may not be known to the robot *a priori*), but maneuvers off the graph are valid and, in fact, quite frequent. Such driving conditions are common and arise, for example, in parking lots and garages, construction zones, around shopping centers. For instance, most people driving through a parking lot shown in Figure 1 will—for the most part—stay on a lane graph such as the one in the image. However, at times, they might significantly deviate from the graph while performing maneuvers such as turning around, pulling in and out of parking spaces, avoiding other cars, etc.

Autonomous driving in semi-structured environments has two main challenges. The first challenge is perceptual. The robot must not only sense its surroundings at a low level (detecting drivable and non-drivable terrain), but also understand the world at a higher level and infer the topological structure of the environment. In the first part of the paper, we discuss the problem of estimating topological structure from sensor data and present an algorithm for automatically constructing lane-network graphs. The graph in Figure 1 is one example. It was constructed automatically from sensor data obtained while driving through the parking lot.

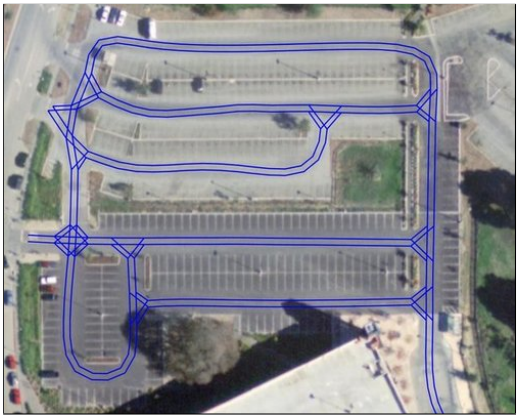


Fig. 1. Topological lane graph imposed on a parking lot. This graph was automatically constructed from sensor data.

The second challenge lies in planning. Unconstrained free-space path planning often produces trajectories that are inappropriate for semi-structured environments (e.g., cutting across parking lots). Conversely, constraining the robot to only drive on the lane graph might be too restrictive. In this work, we extend our previously-developed path-planning algorithm used in the DARPA Urban Challenge [6] to seamlessly integrate graph-based and free-space planning. The result is a faster planning method that produces trajectories conforming to the common driving conventions implied by the environment structure.

We present experimental results on mapping and path-planning in real semi-structured environments. All experiments were performed on a robotic vehicle—shown in Figure 2—equipped with a high-accuracy pose-estimation system (Applanix) and a number of laser range finders, of which the most important was the 3D LIDAR (Velodyne).

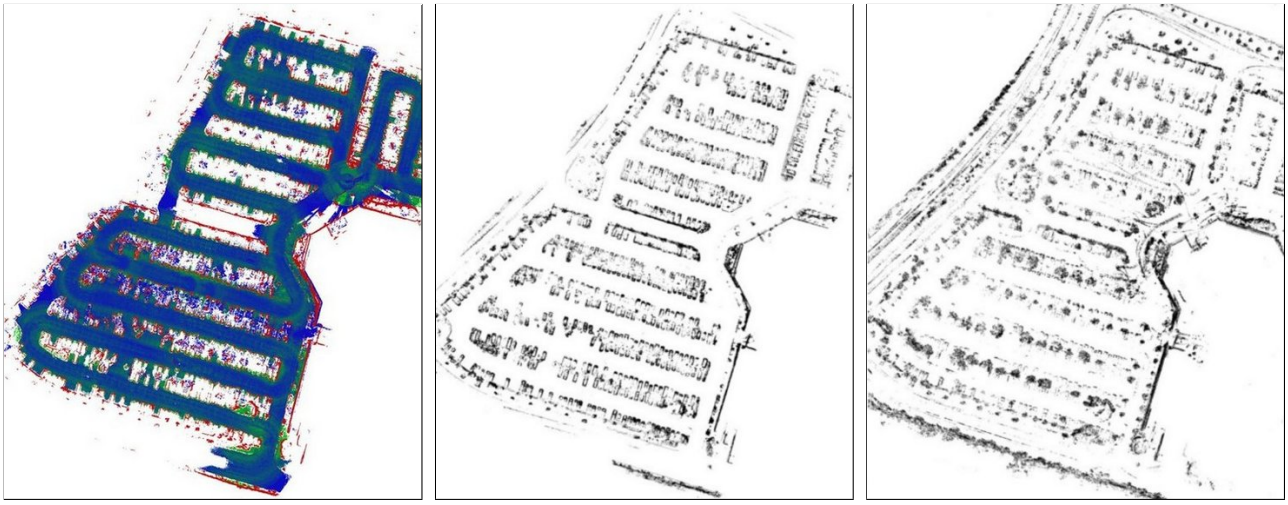
II. Mapping

To estimate topological structure of the environment from sensor data, we first build a grid-based map of static obstacles and then use that map to estimate a likely corresponding graph of drivable lanes.

A. Building Grid-Based Obstacle Maps

The quality of the static grid-base obstacle map has a crucial effect on the quality of the final estimate of the topological structure of the environment. Fortunately, the mapping application has low requirements on detection delays, which permits a simple algorithm for generating high-accuracy obstacle maps. There is little novelty in this part of the algorithm, but we describe it for completeness.

[1]Toyota Research Institute, AI & Robotics Group, Ann Arbor
[2]Stanford University, Computer Science Department, Stanford



(a) (b) (c)

Fig. 3. Best viewed in color. Building grid-based obstacle maps. (a) Shows accumulated statistics; Green: ground-plane evidence; Blue: ray-traced evidence; Red: obstacle evidence. (b) shows the resulting grid-based obstacle map, (c) For comparison, shows the mapping results from a baseline approach [14], which was tuned for real-time obstacle detection.

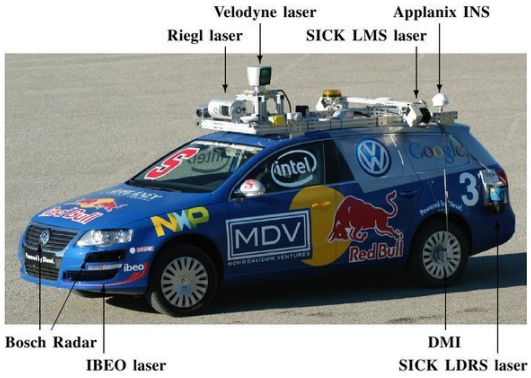


Fig. 2. Our robotic vehicle is equipped with an extensive sensor suite. For the experiments presented in this work, we used the high-precision pose-estimation GPS-IMU system by Applanix and the 3D LIDAR by Velodyne.

Our static mapper is based on the perception algorithm used by the Stanford's entry in the DARPA Urban Challenge, Junior [14]. The core of Junior's perception maintains a variant of an occupancy grid [21] around the robot, where the obstacle hits are computed from the 3D point cloud generated by the 3D LIDAR. It uses free-space analysis with 2D raytracing to clear dynamic obstacles from the map.

For the purposes of this work, we modified the algorithm to produce more accurate static maps at the expense of taking longer to detect new obstacles. We made two main changes.

First, for every cell in the grid, we accumulated statistics on how many times the cell has been a) within the observation range, b) contained an obstacle, c) contained the ground plane, d) was ray-traced through. We also retained an estimate of the height of an obstacle to compensate for the fact that lower obstacles have a lower detection rate and are more likely to be ray-traced through. We then used a classifier to label cells as free or occupied as a function of these features. We experimented with several classifiers (both manually tuned and automatically learned), and found a cascade of simple linear classifiers to work well.

The second modification of the perception algorithm was in the ray-tracing approach. Instead of using 2D ray-tracing as described in [14], we used a "2.5D" ray-tracing algorithm, developed by James [8]. The latter makes an assumption that dynamic objects don't change in height (typically holds for driving environments). This leads to a drastic reduction in the rate of false positives (phantom obstacles) at a fraction of the computational cost of full 3D ray-tracing.

Figure 3 illustrates our grid-based mapping approach. Figure 3a shows the accumulated statistics for the environment, Figure 3b shows the output of our classifier. For comparison, Figure 3c shows the output of the baseline approach [14]. Notice that the map Figure 3b is more accurate in terms of both false positives (phantom obstacles) and false negatives (holes in static obstacles). This is due to several factors: i) the use of 2.5D ray-tracing instead of 2D, ii) ray-tracing against sensor data accumulated over several cycles (causing dynamic obstacles to persist longer, but resulting in a more accurate static map), and iii) the use of accumulated statistics (which increases accuracy at the expense of detection time).

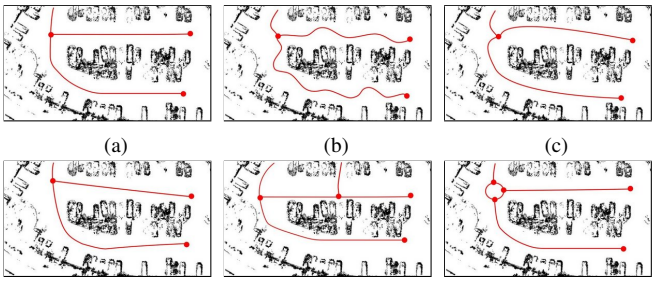
B. Estimating Topological Lane Graphs

Given a grid-based obstacle map computed as described above, our goal is to create a topological lane-network graph that best fits the map. We reduce the problem to first estimating a 2D graph of the center-line of the lane network and then constructing the lanes around this center-line graph. From a probabilistic perspective, our goal then is to find the most likely topological center-line graph corresponding to the observed discrete obstacle grid.

The standard way to proceed in this situation is to define a potential that encodes our prior on how networks of driving lanes are typically constructed (connectivity, smoothness, curvature, width, etc.), as well as a potential that models how well a lane network fits the given obstacle map (distance to obstacles, directional alignment, etc). We can then apply an

(a) (b) (c) (d)

Fig. 4. The main steps of building a topological lane graph from a grid-based map. (a): Compute the Generalized Voronoi Diagram on the obstacle map. (b): Apply graph-simplification and pruning rules, then locally smooth each edge while holding all intersections fixed. (c): Iteratively apply global smoothing, while updating the continuity/discontinuity potentials across intersections. (d): Once the graph has converged, estimate the driving-lane network.



(a) (b) (c) (d) (e) (f)

Fig. 5. Illustration of the effects of priors over topological lane graphs.

appropriate inference technique to this model to estimate the most likely lane network, given the observed obstacle map.

Consider an obstacle map $O = \langle o_x, o_y \rangle$, defined as a collection of 2D points, and a center-line graph $\mathcal{G} = \langle V, E \rangle$ with vertices V and edges E. Also, define a set of *intersection* nodes $I \in V$ consisting of nodes with fewer than 2 neighbors (deadends or orphans) or more than 2 neighbors (intersections). Intersection nodes are shown as solid circles in Figure 5. We use the term *lane segment* $S_l = \{v \in V\}$, $l \in [1, L]$ to refer to a collection of connected nodes between intersections. The set of all such nodes is S.

Given an obstacle map O and a center-line graph $\mathcal{G}$, consider the following potential:

$$f(\mathcal{G}, O) = w_{sm} f_{sm}(S) + w_{in} f_{in}(I) + w_{\kappa} f_{\kappa}(S) + w_{dir} f_{dir}(S, O) + w_O f_O(V, O) + w_{topo} f_{topo}(\mathcal{G}),$$

where the terms have the following meaning (precise definitions of our realization of these priors will follow).

- $f_{sm}(S)$ defines the prior on the smoothness of lane segments. This introduces a bias towards graphs with smooth segments between intersections. For example, the term models our belief that a lane network centered around the graph in Figure 5a is more likely than the one in Figure 5b.
- $f_{in}(I)$ defines the constraints on the direction of lane segments that merge at an intersection. For example, this term encodes our belief that the three-way intersection in Figure 5a is more likely than the one in Figure 5c.
- $f_{\kappa}(S)$ is the prior on the maximum curvature of a lane segment. This models the fact that roads have a bounded curvature, limited by the turning radii of vehicles (if a car cannot follow a curve, it is probably not a lane segment).
- $f_{dir}(S, O)$ is the "directional alignment" potential. In a typical driving environment, there is a preferred driving direction $\alpha_O(x, y)$ at any $\langle x, y \rangle$ location. These directions are typically aligned with environmental features such as road-side curbs, parked cars, etc. A field of such principal directions can be estimated from the observed obstacle map using a Markov-Random-Field technique, as described in our previous work [5]. The potential $f_{dir}(S, O)$ introduces a bias towards lane segments S aligned with the field of principal directions, corresponding to the observed obstacle map O. For example, this term models the belief that Figure 5a is a better match for the obstacle map than Figure 5d.
- $f_O(V, O)$ is the potential on the distance from graph vertices to obstacles, which encodes our prior on typical road widths. For example, the term assigns a low probability to the graph in Figure 5e, modeling our belief that the center of a lane segment cannot pass too close to an obstacle.
- $f_{topo}(\mathcal{G})$ encodes the prior on topologies of lane-network graphs. For instance, this term would model our belief that Figure 5a is a more likely topology than Figure 5f.

Our goal is to find the graph that minimizes the sum of the above potentials. This leads to a complex modeling and computationally expensive optimization problem, because the optimization is over the space of 2D graphs, involving both continuous variables (coordinates of nodes) and discrete variables (number of nodes and topology of the graph).

To make the problem tractable, we make a simplifying assumption that the center-line graph is a homomorphism (continuous transformation) of a subset of the Generalized Voronoi Diagram (GVD) of the obstacle map. This holds when drivable lanes are separated by obstacles; for example in parking lots where zones of parking spots are separated

Algorithm 1: Estimating lane graph from an obstacle map.

$\mathcal{G} \leftarrow$ Generalized Voronoi Diagram of O
while $\mathcal{G}$ *not converged* **do**
1. apply discrete heuristic transformations to $\mathcal{G}$
2. locally smooth each edge of $\mathcal{G}$
3a. define intersection potential $f_{in}(I)$ on $\mathcal{G}$
3b. globally smooth $\mathcal{G}$ via $\min(f_{sm} + f_{in} + f_\kappa + f_{dir} + f_O)$

by curbs or parked cars. The assumption means that the approach will fail, for example, in empty parking lots. In such cases, it would be necessary to collect more data or use different sensors to generate the initial grid-based map. Our assumption constrains the search to a neighborhood of the GVD and provides a starting point for our estimation.

We start with the GVD of the obstacle map and then transform the corresponding graph in a gradient-descent manner. To further reduce computational overhead, we resort to an iterative procedure outlined in Algorithm 1.

The first step deals with discrete variables that define the graph topology. We consider several heuristic transformations of the graph topology with the effect of minimizing the topology potential $f_{topo}(\mathcal{G})$, given the current setting of continuous graph-node coordinates. We define several topology-transforming rules (delete edges, merge intersections, collapse loops, etc.) and apply all such transformations that lower the topology potential $f_{topo}(\mathcal{G})$.

In the second step, we locally smooth each lane segment S_l, while holding the positions of intersections fixed.

In the third step (3a and 3b in Algorithm 1), we first infer the (dis)continuity potentials across intersections. Then, we solve a global continuous-variable optimization problem on the coordinates of the graph nodes that minimizes the sum of the continuous potentials $f_{sm}, f_{in}, f_{dir}, f_O$.

We now define the potentials more precisely. Consider a graph $\mathcal{G} = \langle V, E\rangle$ with vertices $V = \{\mathbf{x}_i\} = \{\langle x_i, y_i\rangle\}$, intersections I, and lane segments $S = S_l$. Let $\mathcal{N}(k),\ k \in I$ be the set of neighbors of an intersection node, on which a smoothness potential is imposed (during step 3a of Algorithm 1). Let $\mathbf{\Delta}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ be a vector between adjacent nodes i and j. Further, let $\theta_i = \tan^{-1}\frac{y_{i+1}-y_i}{x_{i+1}-x_i},\ i \in S_l$ be the tangential angle of a lane segment at node i, let $\kappa_i = \frac{\|\theta_i - \theta_{i-1}\|}{\|\mathbf{x}_i - \mathbf{x}_{i-1}\|}$ be the curvature at node i, and let $\alpha_O(\mathbf{x}_i)$ be the principal-direction angle at location $\mathbf{x}_i$, inferred from the obstacle map O (as described in [5]).

We can then define the continuous potentials as:

$$
\begin{aligned}
f_{sm}(S) &= \sum_{i\in S} \|(\mathbf{x}_{i+1} - \mathbf{x}_i) - (\mathbf{x}_i - \mathbf{x}_{i-1})\|, \\
f_{in}(I) &= \sum_{i\in I} \|\sum_{j,k\in\mathcal{N}(i)} (\mathbf{\Delta}_{ij} - \mathbf{\Delta}_{ik})\|, \\
f_\kappa(S) &= \sum_{i\in S} (\kappa_i - \kappa_{max})^2, \\
f_{dir}(S,O) &= \sum_{i\in S} \sin^2\left(2(\theta_i - \alpha_O(\mathbf{x}_i))\right), \\
f_O(V,O) &= \sum_{i\in V} \left(|\mathbf{x}_i - \mathbf{o}_i| - d_{\max}\right)^2,
\end{aligned}
\tag{1}
$$

where $\mathbf{o}_i$ is the location of the obstacle from O closest to $\mathbf{x}_i$, and κ_{max} and $d_{\max}$ are the maximum admissible road curvature and width, respectively.

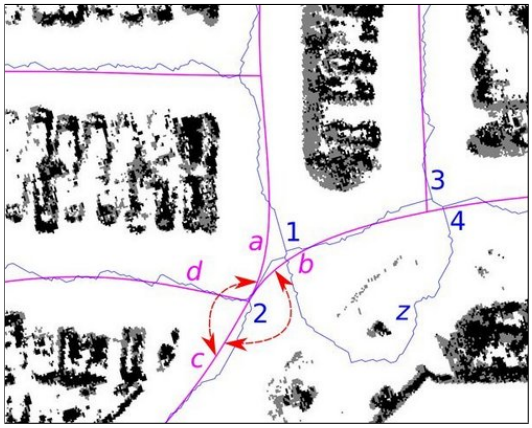


Fig. 6. Comparison of the raw GVD to the output of our algorithm after discrete heuristic transformations and global smoothing using conjugate gradient. Discrete transforms deleted edge "z" and merged intersections "1-2" and "3-4". Intersection "a-b-c-d" had smoothness potentials between edges "c-a" and "c-b", but not between other edges (for example, notice the discontinuity between "c-d").

The optimization with these potentials is illustrated in Figure 4. Figure 4a shows the raw GVD for the given obstacle map O. We first modify the topology of the graph by removing low-probability lane segments, collapsing some short loops, and merging nearby intersections (e.g., notice how two intersections in the middle of the map in Figure 4a were transformed into one four-way intersection in Figure 4b). This corresponds to step 1 in Algorithm 1.

Next, we apply an intermediate optimization step, corresponding to step 2 in Algorithm 1, where each lane segment of the graph is smoothed independently. This is done by minimizing $f_{sm} + f_\kappa + f_O$, while holding fixed the coordinates of all intersections. The output of this step is shown in Figure 4b. Given the locally-smoothed graph, the algorithm then analyzes each intersection and constructs the neighbor sets $\mathcal{N}(i),\ i \in I$, which define the intersection prior f_{in} (step 3a in Algorithm 1). Recall that this prior preserves segment continuity across some intersections. We then solve a global optimization problem on the coordinates of all nodes (step 3b in Algorithm 1), leading to the graph shown in Figure 4c. This iterative process continues until convergence of the graph $\mathcal{G}$. Finally, we estimate the lane-network around $\mathcal{G}$. An example of the resulting lanes is shown in Figure 4d.

Figure 6 further illustrates the process by comparing the raw GVD to the output of the global optimization. In this example, heuristic topology transformations (step 1 of Algorithm 1) removed the edge marked as z and merged two sets of intersections ("1-2" and "3-4"). After local lane-segment smoothing (step 2 in Algorithm 1), an intersection prior was introduced, imposing a smoothness potential between edges "c-a" and "c-b" (marked with arrows in Figure 6). Notice that the process correctly preserved the discontinuity between edge "d" and the other edges at that intersection.

Both the local optimization with fixed intersection nodes, as well as the global graph optimization were carried out using conjugate gradient descent [16].

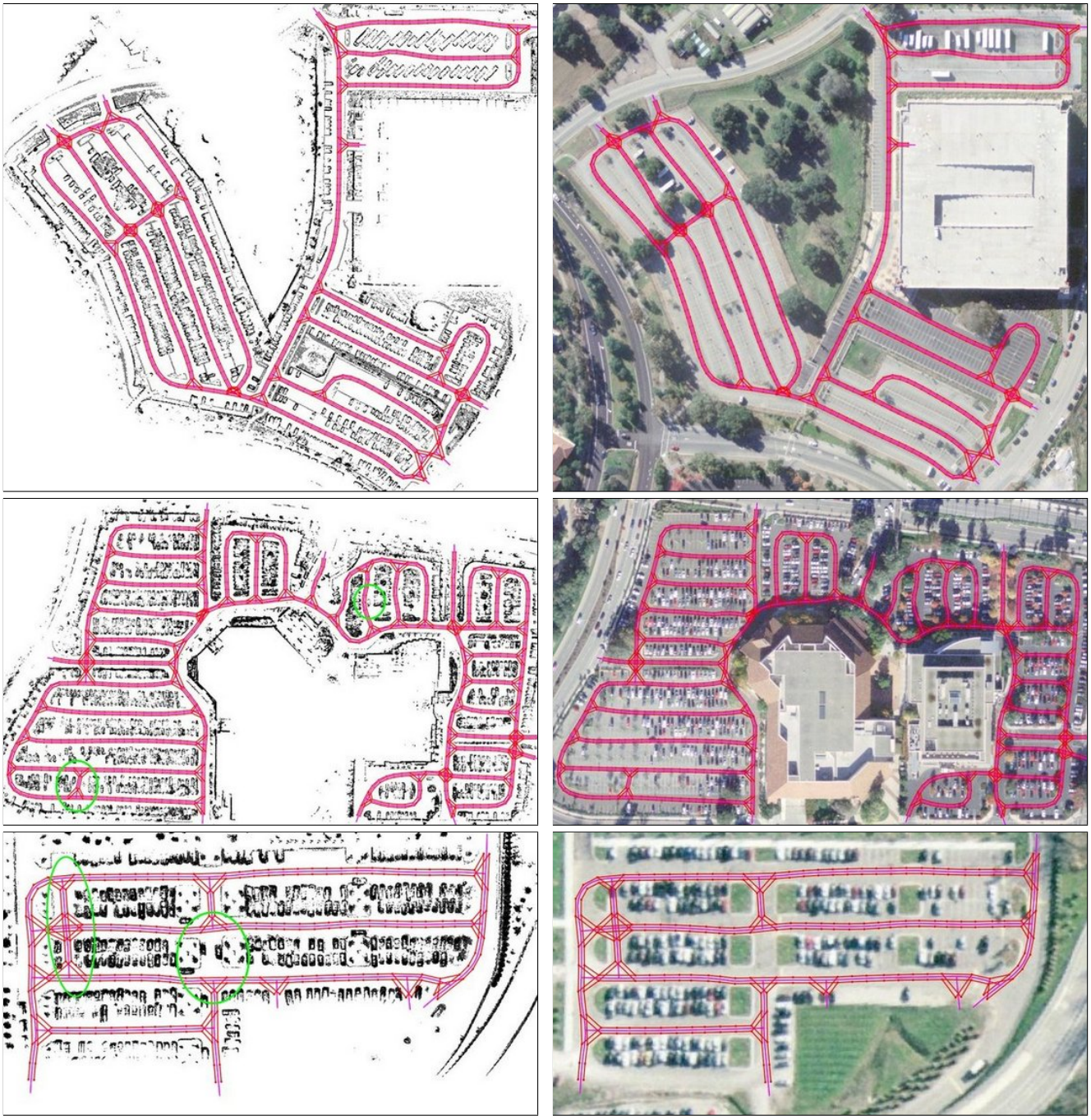


Fig. 7. Best viewed in color. Lane-network graphs overlaid on the obstacle maps generated from sensor data (left) and aerial imagery (right). The estimation in the first-row example is correct. In the second and third rows, green ovals highlight problematic areas (see Section II-C for analysis).

C. Experimental Results

A few experimental results of running the mapping algorithm described above on real data in parking lots are shown in Figure 7. The left column overlays the computed lane network on the obstacle map, while the right column shows the lane network on top of aerial imagery (taken at a different time than the obstacle map) of the same environment.

The estimation in the top row is correct. For the second and third row, we have highlighted some problematic areas in the left column of Figure 7. In the second row, there are two problems: the algorithm introduces an incorrect vertical lane segment (lower left), and a lane segment swerves around a poorly parked car (top right) that was stationary throughout our data collection. The third row also has two problems: two false lane segments are introduced (left) and a lane segment is not detected (center). The false positives are detected in the handicapped zones that are right next to a curb. From the perspective of our algorithm, such zones are indistinguishable from a real lane. To correctly map such areas, additional features—e.g., road markings—would be needed. The false negative is due to two cars poorly parked

on the sides of a lane, making the lane barely passable, which causes our algorithm to disregard the passage as a valid lane.

III. Semi-Structured Path Planning

We now discuss the problem of path planning in semi-structured environments. We assume that a graph of driving lanes—such as the ones generated by our algorithm from the first part of the paper—is available, but maneuvers off the graph are allowed. The goal of the path planner is to find a trajectory from the initial state of the vehicle $\langle x, y, \theta \rangle_0$ to a goal state $\langle x, y, \theta \rangle_g$. The trajectory has to be safe, kinematically-feasible, smooth, and near-minimal in length.

Our path planner for semi-structured environments is based on the free-space planner for a robotic vehicle used by the Stanford racing team in the DARPA Urban Challenge (DUC).[1] This DUC path-planning algorithm consists of two main phases. Phase I uses A* search to find a safe, feasible, and approximately optimal trajectory. The trajectory is then improved during phase II by solving a non-linear optimization problem in continuous coordinates, initialized with the A* solution from phase I. This two-phase approach is detailed in our previous work [6]. Here, we outline the extensions necessary to take advantage of the known topological structure in a semi-structured environment.

This section as well our earlier work builds on extensive body of existing research on search algorithms and path planning for wheeled robots (e.g., [7], [13], [9], [12], [15]).

A. Phase I: A* Search

The first phase of the algorithm uses A* with a discrete set of control actions, operating on the 4-dimensional kinematic state of the vehicle $\langle \mathbf{x}, \theta, r \rangle$, where $\langle \mathbf{x}, \theta \rangle = \langle x, y, \theta \rangle$ define the position and orientation of the car, and $r = \{0, 1\}$ specifies the direction of motion (forward or backward).

The role of the lane network in this 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 well tuned to the current environment.

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 to the graph:

$$\mathcal{D}(\mathcal{G}, \langle \mathbf{x}, \theta \rangle) = \min \big\{ E : |\alpha_E - \theta| < \alpha_{min} \big\} \mathcal{D}(E, \mathbf{x}), \quad (2)$$

where $\mathcal{D}(E, \mathbf{x})$ is the Euclidean distance between a line segment E and point $\mathbf{x}$. In words, the distance between a state of the vehicle and the lane-network graph is the Euclidean distance between the $\langle x, y \rangle$ position of the car and the nearest edge, whose orientation is close (within α_{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:

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

[1]Here, we refer to the path planner used for navigating parking lots during the DUC, which is different from the planner used for DUC street driving.

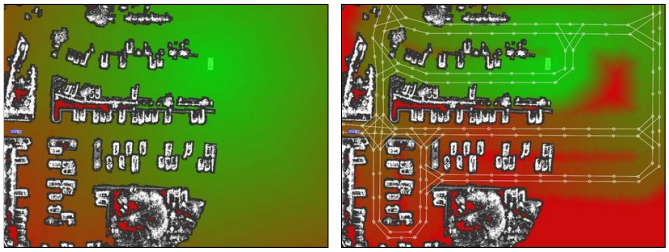


Fig. 8. Best viewed in color. Comparison of the 2D heuristic for the same environment with and without using a lane network. Green color component decreases and red component increases with distance to the goal. Notice the difference in the costs in the obstacle-free region on the right. The lane network is shown in white for the case that uses it (on the right).

where $\delta(i)$ is the Kronecker delta ($\delta(0) = 1,\ \delta(i \neq 0) = 0$); C_{rev} and C_{sw} are costs for driving in reverse and switching the direction of motion, respectively; $C_{\mathcal{G}}$ is the penalty for deviating from the lane graph by more than $\mathcal{D}_{min}$.

Our free-space A* uses two heuristics to estimate distance to the goal. The first heuristic takes into account the non-holonomic nature of the vehicle, but ignores obstacles. This heuristic is unchanged for the case of semi-structured planning. The second heuristic ignores the non-holonomic nature of the car, but takes into account the current obstacle map. Basically, the heuristic assumes that the robot is a disk with the diameter equal to the width of the vehicle and that it can move in any direction. The heuristic computes for any $\langle x, y \rangle$ position the minimal cost to the goal. In the case of semi-structured planning, this heuristic remains admissible in the A* sense (deviating from the lane graph increases costs), but it provides poor guidance for search. The heuristic can be modified to take into account the graph-modified traversal costs defined above. We retain the admissibility of the heuristic in the 2D 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 8 illustrates the modifications to the 2D holonomic-with-obstacles heuristic. The figure on the left shows the heuristic for the free-space planning problem, while the figure on the right shows the version using the modified traversal costs that take the lane network into account.

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. For each node, we generate a fixed set of children 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 [17] model to analytically compute a path to nearby nodes of the lane-network graph. If the resulting Reed-Shepp curve is collision-free, the corresponding child state is added to the search tree.

Figure 9 illustrates the process.[2] 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

[2]For illustration purposes (to minimize clutter from multiple free-space expansions), the traversal cost for macro-actions was discounted (0.1) compared to free-space expansions.

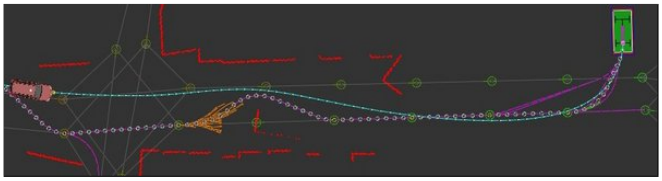


Fig. 9. Best viewed in color. Combining topological graph search and free-space path planning. The Redd-Shepp transitions to nodes of the lane-network graph are shown in purple. The forward free-space expansions are shown in yellow. The cyan curve is the output of the smoother (phase II).

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.

B. Phase II: Path Smoothing

The second phase of our path-planning algorithm takes a uniformly discretized version of the A* solution and formulates a non-linear continuous-variable optimization problem on the coordinates of the vertices of the path. The objective function used in the optimization has terms for ensuring safety and kinematic feasibility of the trajectory, while biasing it towards shorter and smoother curves. The details are described in our previous work [6]. Here, we describe the modifications to the objective function 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

$$\rho(\mathbf{s}) = \rho_0(\mathbf{s}) + w_{\mathcal{G}} \sum_i \mathcal{D}(\langle \mathbf{x}_i, \theta_i \rangle, \mathcal{G}), \tag{3}$$

where $\rho_0(\mathbf{s})$ includes the potential terms used in the free-space planner [6], while the second term provides a penalty for deviating from the graph ($w_{\mathcal{G}}$ is the associated weight).

We once again use conjugate gradient (CG) to obtain a solution to this minimization problem. An efficient implementation of CG requires an analytical gradient of the objective function, which for the lane-graph attraction potential is computed as follows.

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 $\mathbf{x}$ 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 convenience, let us also refer to the current trajectory vertex $\mathbf{x}_i$ as $\mathbf{d}$. Our goal is to compute the derivative of the distance between point $\mathbf{d}$ and edge $\mathbf{p}$ w.r.t. the coordinates of $\mathbf{d}$. Let $\mathbf{e}$ be the projection of $\mathbf{d}$ onto $\mathbf{p}$.

$$\mathbf{e} = \mathbf{p}_1 + \beta \mathbf{p}, \qquad \beta = \frac{(\mathbf{d} - \mathbf{p}_1)(\mathbf{p})}{\|\mathbf{p}\|^2}. \tag{4}$$

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 endpoint (otherwise, use the derivative of distance between two points).

We seek the derivative of the length of $\mathbf{r} = \mathbf{d} - \mathbf{e}$:

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

Expanding the last term

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

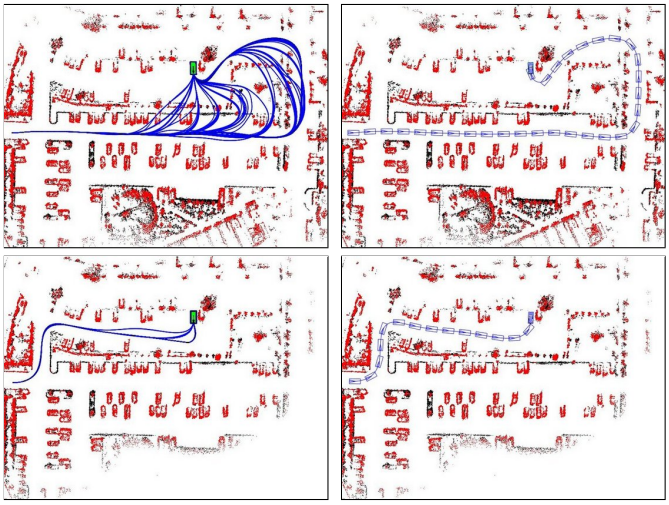


Fig. 10. Free-space path planning (top) vs. graph-guided planning (bottom). Left column shows the set of all trajectories generated as the vehicle detects new obstacles and re-plans. Right column shows the final trajectory driven. The graph-guided planner used the lane-network graph shown in Figure 1

we obtain:

$$\frac{\partial \|\mathbf{r}\|}{\partial \mathbf{d}} = \frac{1}{\|\mathbf{r}\|} \Big(\mathbf{r}^T - \frac{\mathbf{p}^T}{\|\mathbf{p}\|^2} (\mathbf{r}\mathbf{p}) \Big). \tag{5}$$

Combined with the derivatives of other terms from the free-space planner, the realization of CG is standard.

C. Experimental Results

Figure 10 illustrates the benefit of using a topological graph to guide path planning, compared to a free-space planner. The left column 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 driven. The top row depicts results for a free-space planner, while the bottom row shows the results for a planner guided by a lane-network graph. Due to the prior provided by the graph, the latter is more efficient: it performs fewer re-planning cycles (3 compared to 22) and the total number of nodes expanded by A* is ≈15,000 vs. ≈650,000 for the free-space planner. The graph-guided planner also produces a final trajectory better suited to the environment.

A video illustrating the difference between free-space and graph-guided planning for the scenario in Figure 10 as well as another parking task is available at `http://ai.stanford.edu/~ddolgov/icra09/parking.mpeg`. The planner in the video uses the lane network from Figure 1.

Figure 11 shows a few path-planning experiments performed in a real parking lot. Notice that all generated trajectories follow the topological structure of the environment.

IV. Conclusions

We analyzed two aspects of autonomous driving in semi-structured environments: estimation of lane-network graphs from sensor data and the use of such topological graphs in path planning. The benefits of using topological graphs in path planning are twofold. First, such graphs provide a good set of macro actions that lead to computational benefits. Second, the resulting paths adhere to the commonly accepted

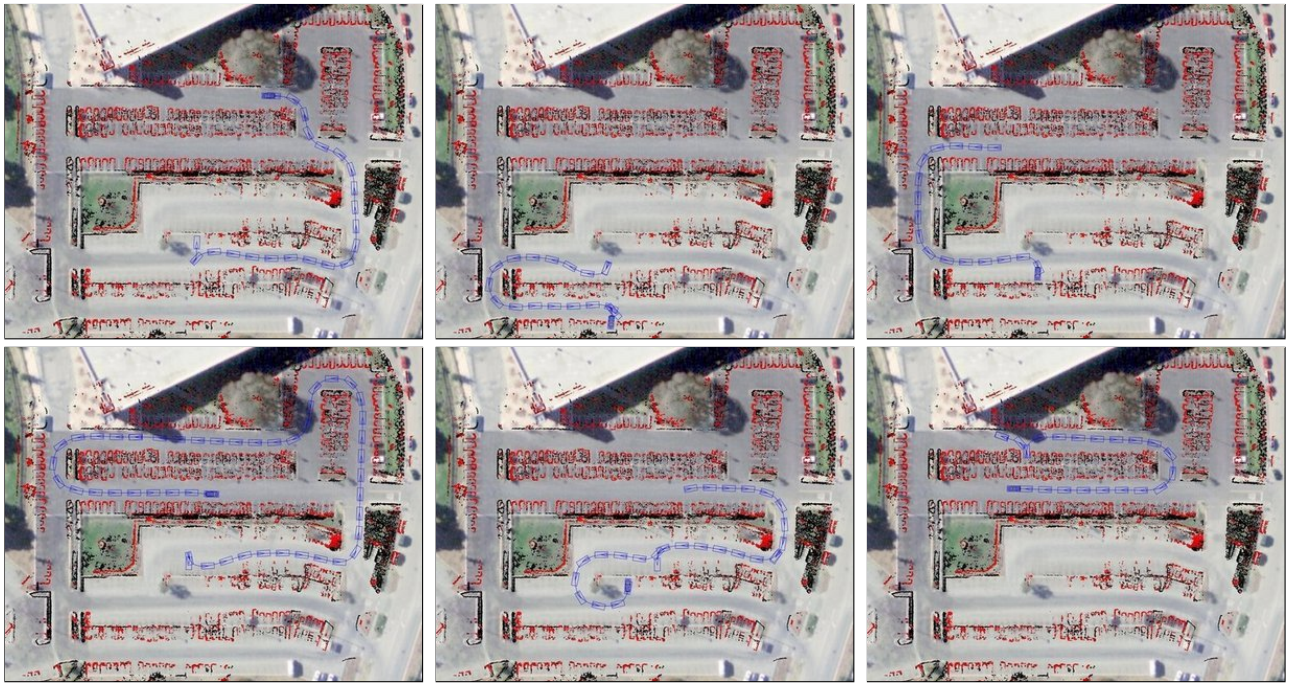


Fig. 11. Autonomous driving in a parking lot, with path planning guided by a topological lane-network graph. An obstacle map (red) is overlaid on aerial imagery. The driven trajectories are shown in blue.

driving conventions, which makes autonomous cars more predictable (and thus safer) for humans and other robots.

Acknowledgments

We gratefully acknowledge the invaluable contributions of the following people. Michael Montemerlo wrote much of the autonomous-driving and sensor-processing software used in this work. Dirk Haehnel wrote the perception code, used as the core of our grid-based obstacle map generation. Michael James wrote 2.5D ray-tracing used in map generation. James Diebel wrote the conjugate-gradient optimization library used in lane-network estimation and path smoothing.

REFERENCES

[1] *The 2005 DARPA Grand Challenge: The Great Robot Race*. Springer, 2005.

[2] Special issue on the 2007 DARPA Urban Challenge, part I,II. *Journal of Field Robotics*, 25(8):423–566, August 2008.

[3] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *International Conference on Robotics and Automation (ICRA)*, 1999.

[4] E D Dickmanns and A Zapp. Autonomous high speed road vehicle guidance by computer vision. *Triennial World Congress of the International Federation of Automatic Control*, 4:221–226, July 1987.

[5] Dmitri Dolgov and Sebastian Thrun. Detection of principal directions in unknown environments for autonomous navigation. In *Proc. of Robotics: Science and Systems (RSS-08)*, 2008.

[6] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous driving in unknown environments. In *Proceedings of the Eleventh International Symposium on Experimental Robotics(ISER-08)*, Athens, Greece, July 2008.

[7] David Ferguson and Anthony Stentz. Field d*: An interpolation-based path planner and replanner. In *Proceedings of the Int. Symp. on Robotics Research (ISRR)*, October 2005.

[8] Michael James. A novel free-space analysis algorithm for mapping complex dynamic environments. In *Under Review*, 2008.

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

[10] A. Kelly. An intelligent predictive control approach to the high speed cross country autonomous navigation problem, 1995.

[11] Sascha Kolski, David Ferguson, Mario Bellino, and Roland Siegwart. Autonomous driving in structured and unstructured environments. In *IEEE Intelligent Vehicles Symposium*, 2006.

[12] S. LaValle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[13] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.

[14] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun. Junior: The stanford entry in the urban challenge. *J. of Field Robotics*, 25(9), 2008.

[15] E. Plaku, L. Kavraki, and M. Vardi. Discrete search leading continuous exploration for kinodynamic motion planning. In *Robotics: Science and Systems*, June 2007.

[16] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[17] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific J. of Mathematics*, 145(2), 1990.

[18] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr. Recent progress in local and global traversability for planetary rovers. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2000*, 2000.

[19] Anthony (Tony) Stentz and Martial Hebert. A complete navigation system for goal acquisition in unknown environments. In *Proceedings 1995 IEEE/RSJ International Conference On Intelligent Robotic Systems (IROS '95)*, volume 1, pages 425 – 432, August 1995.

[20] C. Thorpe, T. Jochem, and D. Pomerleau. Automated highway and the free agent demonstration. In *Robotics Research – International Symposium*, volume 8, 1998.

[21] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005.

[22] S Tsugawa, N Watanabe, and H. Fujii. Super smart vehicle systemits concept and preliminary works. *Vehicle Navigation and Information Systems*, 2:269 –277, October 1991.

[23] P. Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2), February 1993.