2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

# Motion Planning for Autonomous Vehicles in Highly Constrained Urban Environments

Dennis Fassbender, Benjamin C. Heinrich and Hans-Joachim Wuensche*

*Abstract*— **In this paper, we present a motion planning algorithm for autonomous navigation in highly constrained urban environments. Since common approaches to on-road trajectory planning turned out to be unsuitable for this task, we instead extended an A*-based planner originally designed for navigation in unstructured environments. Two novel node expansion methods were added to obtain smooth and accurate trajectories that consider the structure of the environment. The first one attempts to find a trajectory connecting the current node directly to the goal by solving a boundary value problem using numerical optimization. The second method leverages a simulated pure-pursuit controller to generate edges (i.e. short motion primitives) that guide the vehicle toward or along the global reference path. As a result, the planner is able to produce smooth paths while retaining the explorative power of A* that is needed to deal with challenging situations in urban driving (e.g., reversing in order to pass a vehicle that stopped unexpectedly). Its practical usefulness was demonstrated during extensive tests on an electric vehicle navigating a mock urban environment as well as on our own autonomous vehicle MuCAR-3.**

## I. INTRODUCTION

The problem of autonomous urban driving has received considerable attention since the 2007 DARPA Urban Challenge (DUC). Nevertheless, motion planning in urban environments remains challenging for a variety of reasons. While the structure of the environment as well as the rules of the road usually allow trajectory planners to make a number of simplifying assumptions, an autonomous vehicle may still encounter difficult situations due to unexpected behavior by other traffic participants. Furthermore, it may find itself in areas that lack a clearly recognizable structure, thus necessitating the use of more powerful planning algorithms than those typically used for on-road driving.

The work presented here was motivated by the insight that common approaches to on-road driving fail when the environment is highly constrained, such as on narrow roads with sharp turns (see Figure 1). This is due to the fact that these methods generate candidate trajectories to different goal poses without considering obstacles at first. In cluttered environments, however, there may be situations where none of these trajectories are free of obstacles. Also, these on-road approaches are usually not powerful enough to plan complex maneuvers that include reversing. While others address these issues by switching between different motion planners (see Section II), we found this to be cumbersome and instead decided to develop a single algorithm capable of generating both smooth trajectories aligned with the course of the road and complex maneuvers.

*All authors are with the Department of Aerospace Engineering, Institute for Autonomous Systems Technology (TAS), University of the Bundeswehr Munich, Germany. Contact author email: dennis.fassbender@unibw.de

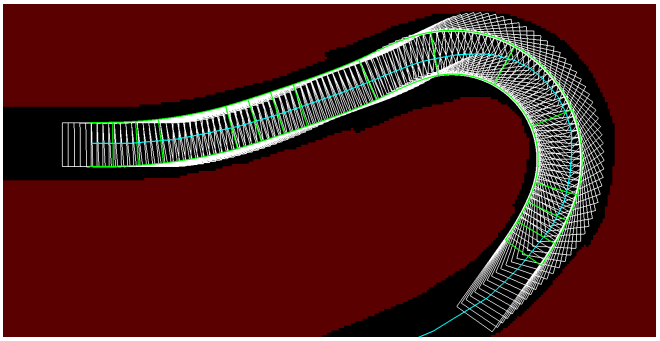


Fig. 1: The vehicle plans a trajectory (green, swept area in white) through a sharp turn. The reference path is shown in cyan, with black cells representing the free space of the road.

The paper is structured as follows. Section II covers related work. After the algorithm is introduced in Section III, Section IV presents experimental results. Finally, Section V concludes the paper and mentions ideas for future work.

## II. RELATED WORK

One of the most popular methods of on-road motion planning is to generate a set of candidate trajectories to different goal poses and pick the best trajectory based on a cost function. The teams that took first and second place in the DUC ([1], [2]) both used this approach. Urmson et al. [1] generate trajectories by solving a boundary value problem (BVP) using numerical optimization. In each step, they foward-simulate a vehicle model whose lateral control inputs are given in the form of a curvature polynomial [3]. By linearizing and inverting the vehicle model, the polynomial's parameters are adjusted based on the deviation of the current solution's end point from the goal. Similarly, Montemerlo et al. [2] generate paths with different lateral offsets relative to the road center by varying the steering parameters of a simulated vehicle.

For navigation in unstructured areas both teams switched to an A*-based motion planner. Urmson et al. perform Anytime Dynamic A* search in a variable-resolution lattice of motion primitives. In contrast, Montemerlo et al. rely on Hybrid A*, a planning algorithm that associates continuous vehicle states with grid cells. While the search tree is usually extended using short motion primitives, the planner sometimes computes a Reeds-Shepp path [4] to the goal or to a point on the lane net in an attempt to speed up the planning process. Finally, the resulting path is smoothed using the

978-1-5090-3762-9/16/$31.00 ©2016 IEEE

conjugate gradient method. The idea of applying numerical optimization to a rather coarse path found by A* was also adopted in works such as [5] and [6].

Other DUC contestants relied on a single planning algorithm during the whole race, with Bacha et al. [7] using A* to find an optimal sequence of precomputed motion primitives and Leonard et al. [8] using a Rapidly Exploring Random Tree-based approach. The latter requires cleverly biased sampling in order to make the vehicle perform the correct maneuvers.

The on-road planning approach in [1] was later extended by McNaughton et al. [9], who introduced a *spatiotemporal lattice*. Various lateral offsets were applied to a set of poses on the lane's center line (*stations* with different longitudinal offsets). Pairs of poses with different longitudinal offsets were connected using cubic curvature polynomials, resulting in a lattice structure that was searched for the least-cost trajectory.

Gu et al. [10] first construct a collision-free reference trajectory from a graph containing nodes with different longitudinal and lateral offsets to the lane center. Candidate trajectories passing through the nodes of the graph are then generated and evaluated based on a cost function that penalizes deviation from the reference trajectory and proximity to moving obstacles.

While Werling et al. [11] also obtain goals by regular sampling, they represent lateral and longitudinal motion in a Frénet frame by quintic and quartic polynomials, respectively. Trajectories that violate the vehicle's constraints are discarded. In contrast, Schwesinger et al. [12] only generate feasible trajectories to a set of goals by closed-loop forward simulation of a vehicle model.

Departing from (partly) sampling-based methods, Ziegler et al. [13] formulate motion planning as a constrained optimization problem. Their approach is promising as the algorithm was successfully tested during a $100\,\mathrm{km}$ drive between two German cities. Nonetheless, the authors themselves note that some situations still call for combinatorial planning to be used in addition to variational methods.

## III. PLANNING ALGORITHM

The planning algorithm presented here is an extension of the A*-based planner described in [14], which contains details regarding the cost function, search tree pruning and other aspects that will not be described again here. After an overview of the regular A* search process in Section III-A, Section III-B will introduce *one-shot expansions*: node expansions that attempt to connect the current node directly to the goal by solving a BVP. The second major extension, *pure-pursuit expansions*, will be covered in Section III-C. Note that both expansion methods are applied *in addition* to the regular expansions described below.

### A. A*-based Planning

In urban environments, the A* planner finds collision-free trajectories using a static obstacle grid map of the local environment as well as a set of dynamic obstacle grids. Each of the latter grids has an associated time interval, which means that occupied cells indicate space that may be blocked by a dynamic obstacle during the time interval for which the grid was created.

The planner works by constructing a tree consisting of nodes that represent vehicle configurations $(x, y, \psi, c_0, v)$, where $(x, y, \psi)$ is the 2D pose, $c_0$ is the curvature, and $v$ is the velocity. An edge between a parent node $n_{\text{parent}}$ and a child node $n_{\text{child}}$ represents a clothoid arc starting at $n_{\text{parent}}$'s configuration and ending at $n_{\text{child}}$'s configuration. If the velocities of $n_{\text{parent}}$ and $n_{\text{child}}$ differ, constant acceleration along the arc is assumed. Hence, paths in the tree represent continuous-curvature trajectories with linear velocity profiles. The length $L$ of the clothoid arcs connecting two nodes is kept fixed.

When a node $n$ with configuration $(x, y, \psi, c_0, v)$ is removed from the OPEN queue for expansion, a fixed number $B$ of child nodes is generated. If the vehicle is driving forward, the first half of the child nodes keep the velocity $v$ while the second half are assigned the velocity $v_{\text{reduced}}$ obtained by applying a negative acceleration $a_{\text{lon}}^{\text{neg}}$ for $L$ meters. The effect of this simulated braking maneuver is that higher curvatures are possible further down the tree, which is important when the vehicle approaches a sharp turn. If $v_{\text{reduced}} < v_{\text{min}}^{\text{pos}}$, where $v_{\text{min}}^{\text{pos}}$ is a positive minimum velocity, these 'slow' nodes are discarded. As a consequence, the planner never attempts to reverse when the car is driving forward. Figure 2 shows an example of the expansion process.

When the vehicle is standing or driving backward, the first half of the nodes is assigned $v_{\text{min}}^{\text{pos}}$, whereas the rest is assigned a small negative velocity $v^{\text{neg}}$. Hence, the vehicle will always drive backward at constant speed and consider changing directions at every node while doing so.

The 2D poses and curvatures of the child nodes $n_1, \ldots, n_B$ are computed by deterministically sampling $B$ curvature rates $c_1^1, \ldots, c_1^B$. Note that we define *rate* as rate w.r.t. distance in the context of curvatures. We then calculate the end poses and curvatures of the clothoid arcs emanating from the parent node's pose $(x, y, \psi)$ with initial curvature $c_0$ and the given curvature rates. The rates $c_1^i$ with $i \in \{1, \ldots, B\}$ are chosen such that two conditons are fulfilled: (1) the final curvature $c_0^i$ does not exceed the maximum curvature imposed by the vehicle's kinematic constraints and its maximum lateral acceleration; (2) the required steering rate does not exceed the vehicle's maximum steering rate.

Now that the child nodes' configurations have been computed, the clothoid arcs are checked for collisions with static and dynamic obstacles and their costs are evaluated (based on curvature, curvature rate and proximity to obstacles, among other things; see [14]). A node's heuristic cost is given by the length of the Dubins path [15] to the goal.

A newly generated node satisfies the termination condition if its heading is roughly identical to the goal's heading and the next expansion would get close to the goal longitudinally or even pass it. The lateral offset to the goal is not considered, which means a trajectory ending on an adjacent lane is a valid solution. If the termination condition is satisfied, the node's

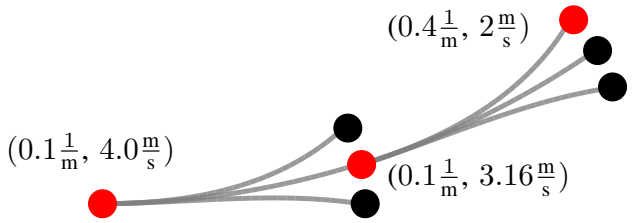



Fig. 2: Two node expansions. The thick dots represent nodes, while the gray lines represent edges (clothoid arcs with a length of 3 m). The tuples represent the respective curvatures and velocities stored in the red nodes. (For the sake of visualization, we allowed unrealistic curvatures and curvature rates.) Constant acceleration of $-1\frac{m}{s^2}$ was used on each arc.

path replaces the current best solution if its cost is lower, and it is discarded if its cost is higher. As the path usually does not reach the goal precisely, its final node's heuristic cost is non-zero and added to the final path cost. However, since some longitudinal distance to the goal remains, the Dubins path to the goal tends to be short, except in rare cases where the lateral offset is large.

If the termination condition is not satisfied, the node is added to the OPEN queue for future expansion.

When the time limit is reached, the planner generates an alternative velocity profile for the solution trajectory. While the solution already has a velocity profile, it may be overly cautious due to the simulated braking maneuver performed during node expansion. Hence, the planner attempts to increase the velocities. This is how the vehicle can accelerate even though the current velocity is never increased by A* (except when driving very slowly). If the new profile would lead to a collision with a dynamic obstacle, the existing conservative profile (which is guaranteed to be safe due to the collision checks at each node expansion) is kept.

### B. One-Shot Expansions

The idea behind one-shot expansions is to find a feasible trajectory taking the vehicle from the current node to the goal in a single step (or one shot). This is useful in cases where the regular A* expansions make slow progress toward the goal, e.g., because it lies in a high-cost region. While this resembles the Reeds-Shepp expansions in [16], our one-shot trajectories are different in that they are curvature-continuous even without further modification. Another benefit is that they tend to be smoother and reach the goal much more accurately than regular A* paths, which suffer from the need to discretize control inputs.

More precisely, our one-shot trajectories are found using the model inversion techniques described in [3]. We first apply Newton's method to adjust the parameters of a cubic curvature polynomial until it connects the configuration $(x^{\text{start}}, y^{\text{start}}, \psi^{\text{start}}, c_0^{\text{start}})$ of node $n_{\text{start}}$ to the goal configuration $(x^{\text{goal}}, y^{\text{goal}}, \psi^{\text{goal}}, c_0^{\text{goal}})$ with the desired accuracy. While the polynomial may be infeasible (e.g., its curvature may be too high), it can be found quickly and provides a good initial guess for the second optimization step, which generates a kinematically and dynamically feasible trajectory using a simulated vehicle model. As the resulting trajectory is a sequence of fixed-length clothoid arcs (i.e. it is $C^1$ continuous), it is not as smooth as the polynomial. However, we found this to be insignificant in practice.

Assuming that the cubic polynomial is parameterized by the vector $\mathbf{p} = [a, b, c, s]^\top$, where $a$, $b$ and $c$ are coefficients and $s$ denotes arc length, the state equations are given by

$$
\begin{aligned}
c_0(\mathbf{p}) &= c_0^{\text{start}} + as + bs^2 + cs^3, \\
\psi(\mathbf{p}) &= \psi^{\text{start}} + \int_0^s c_0([a, b, c, l]^\top)\,\mathrm{d}l, \\
x(\mathbf{p}) &= x^{\text{start}} + \int_0^s \cos(\psi([a, b, c, l]^\top))\,\mathrm{d}l, \\
y(\mathbf{p}) &= y^{\text{start}} + \int_0^s \sin(\psi([a, b, c, l]^\top))\,\mathrm{d}l.
\end{aligned}
$$

Let $f_{\text{poly}}(\mathbf{p}) = [x(\mathbf{p}), y(\mathbf{p}), \psi(\mathbf{p}), c_0(\mathbf{p})]^\top$, with $J_{\text{poly}} = \frac{df_{\text{poly}}}{d\mathbf{p}}$. The parameter update is then given by $\Delta\mathbf{p} = J_{\text{poly}}^{-1}\mathbf{e}$, where $\mathbf{e}$ is the error vector

$$[x^{\text{goal}} - x(\mathbf{p}), y^{\text{goal}} - y(\mathbf{p}), \psi^{\text{goal}} - \psi(\mathbf{p}), c_0^{\text{goal}} - c_0(\mathbf{p})]^\top.$$

In optimization step $k+1$, the parameter vector is updated by setting $\mathbf{p} = \mathbf{p} + \alpha_{k+1}\Delta\mathbf{p}$, where $\alpha_{k+1}$ is a step size parameter that is computed as follows. Let $\mathbf{e}_k$ and $\mathbf{e}_{k+1}$ denote 2-dimensional vectors containing the position errors (i.e. the first two elements of the error vector) in steps $k$ and $k+1$, respectively. We then set

$$\alpha_{k+1} = \min\left(1, \max\left(0.1, \alpha_k \frac{|\mathbf{e}_k|}{|\mathbf{e}_{k+1}|}\right)\right).$$

Hence, the step size increases if the $\mathrm{L}_1$ norm of the position error decreased after the previous step and vice versa, but it always stays within the interval $[0.1, 1]$. The errors in $\psi$ and $c_0$ are not considered in this computation as we can guarantee that they are zero after each parameter update using the technique in [17] (Appendix C).

In order to generate a feasible trajectory, we use the polynomial as lateral control input to the vehicle model shown in Algorithm 1. The inputs to the model are a vehicle configuration (including velocity) and the parameters of the curvature polynomial.

In line 11, the required number of clothoid arcs is determined. For the sake of brevity, we pretend that $s$ is an integer multiple of $L$ (the length of regular A* edges) but this is not required in practice. In line 14, the polynomial's curvature at the current distance is computed. It is passed to the *FeasibleCurvatureRate* function, which limits it based on the vehicle's maximum curvature $c_0^{\max}$ (a function of the maximum steering angle and the wheelbase $w$ according to the bicycle model; line 2) and its maximum curvature at the given velocity $v$ (line 3). The latter depends on the maximum lateral acceleration $a_{\text{lat}}^{\max}$. In line 4, the required curvature rate $c_1$ is determined based on the current curvature $c_0$, the next curvature $c_0^{\text{next}}$ and the arc length $L$. It is limited in line 6 by taking the maximum steering rate $\dot{\lambda}^{\max}$, the velocity $v$ and the wheelbase $w$ into account (line 5). The resulting feasible

**Algorithm 1** The Vehicle Model

1: **procedure** FeasibleCurvatureRate($c_0$, $c_0^{\text{next}}$, $v$)
2: $\quad c_0^{\text{limited}} \leftarrow \text{sgn}(c_0^{\text{next}}) \cdot \min(|c_0^{\text{next}}|, c_0^{\max})$
3: $\quad c_0^{\text{limited}} \leftarrow \text{sgn}(c_0^{\text{next}}) \cdot \min(|c_0^{\text{limited}}|, \frac{a_{\text{lat}}^{\max}}{v^2})$
4: $\quad c_1 \leftarrow \frac{c_0^{\text{limited}} - c_0}{L}$
5: $\quad c_1^{\max} \leftarrow \dot{\lambda}^{\max} \cdot \frac{1+w^2c_0^2}{|v|w}$
6: $\quad c_1^{\text{limited}} \leftarrow \text{sgn}(c_1) \cdot \min(|c_1|, c_1^{\max})$
7: **return** $c_1^{\text{limited}}$
8: **end procedure**
9:
10: **procedure** Predict($x$, $y$, $\psi$, $c_0$, $v$, $a$, $b$, $c$, $s$)
11: $\quad n \leftarrow \frac{s}{L}$
12: $\quad i \leftarrow 1$
13: $\quad$ **while** $i \leq n$ **do**
14: $\quad\quad c_0^{\text{next}} \leftarrow \text{EvaluatePolynomial}(a, b, c, i \cdot L)$
15: $\quad\quad c_1 \leftarrow \text{FeasibleCurvatureRate}(c_0, c_0^{\text{next}}, v)$
16: $\quad\quad (x, y, \psi, c_0) \leftarrow \text{ClothoidArc}(x, y, \psi, c_0, c_1, L)$
17: $\quad\quad i \leftarrow i + 1$
18: $\quad$ **end while**
19: **return** $(x, y, \psi, c_0)$
20: **end procedure**

curvature rate is returned. In line 16, the vehicle's predicted configuration is set to the configuration it would assume after driving along the clothoid arc starting at pose $(x, y, \psi)$ with initial curvature $c_0$, curvature rate $c_1$ and length $L$. After covering the entire distance $s$, the predicted configuration is returned. What is not shown here is that *Predict* can also reduce $v$ in each iteration using constant deceleration.

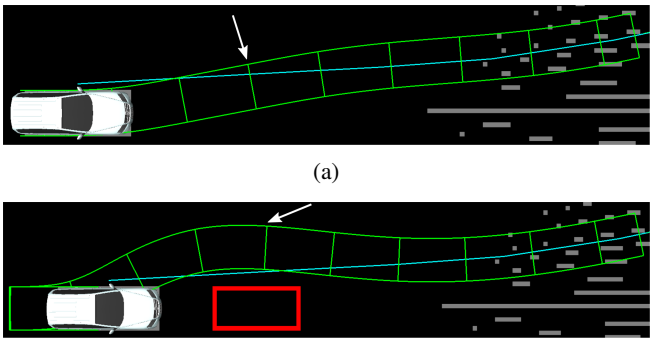
Let $f_{\text{model}} \colon \mathbb{R}^4 \to \mathbb{R}^4$ denote a function that maps a vector of polynomial parameters to the predicted vehicle configuration $[x, y, \psi, c_0]^\top$ returned by the model. We can then numerically compute the Jacobian of $f_{\text{model}}$ w.r.t. $\mathbf{p}$ and again use Newton's method to iteratively adjust $\mathbf{p}$ until $f_{\text{model}}(\mathbf{p})$ is close to the goal configuration. However, we found that representing the polynomial by equally spaced knot points, as described by McNaughton [18], leads to more reliable and faster convergence. [18] shows that by setting

$$q_1 = c_0([a, b, c, \frac{s}{3}]^\top),$$
$$q_2 = c_0([a, b, c, \frac{2s}{3}]^\top),$$
$$q_3 = c_0([a, b, c, s]^\top),$$

the parameter vector $\mathbf{p}$ can be expressed as a function of $\mathbf{q} = [q_1, q_2, q_3, s]^\top$, i.e. $p(\mathbf{q}) = [a(\mathbf{q}), b(\mathbf{q}), c(\mathbf{q}), s]$ with

$$a(\mathbf{q}) = -\frac{11c_0^{\text{start}} - 18q_1 + 9q_2 - 2q_3}{2s},$$
$$b(\mathbf{q}) = \frac{9(2c_0^{\text{start}} - 5q_1 + 4q_2 - q_3)}{2s^2},$$
$$c(\mathbf{q}) = -\frac{9(c_0^{\text{start}} - 3q_1 + 3q_2 - q_3)}{2s^3}.$$

Due to this relationship, we can now numerically compute the Jacobian of $f_{\text{model}} \circ p$ w.r.t. $\mathbf{q}$ and iteratively adjust $\mathbf{q}$ until the configuration given by $f_{\text{model}}(p(\mathbf{q}))$ is sufficiently accurate. Note that it is possible to seed the model-based optimization using polynomial parameters that have not been optimized beforehand. However, we opted for this two-step approach since the first optimization step is faster than the model-based one and often provides an initial guess that requires just one further model-based iteration.



(a)

(b)

Fig. 3: One-shot-based solution trajectories (green) in simulated scenarios (reference path: cyan). In the obstacle-free case (a), regular A* expansions generated the first three segments, while the remaining segments are part of a one-shot path. The white arrow indicates the transition point. In (b), an obstacle (red) causes A* to reverse and maneuver around the obstacle before a one-shot path reaches the goal.

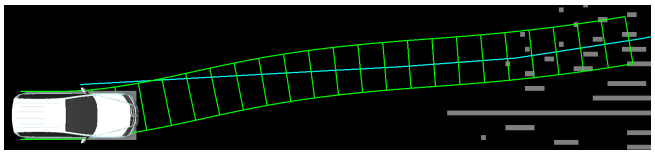
The trajectory computed by the model for the final value of $\mathbf{q}$ is checked for collisions (as obstacles are not considered during optimization) and evaluated using A*'s cost function. By attaching the trajectory to the initial segment ending at node $n_{\text{start}}$, we obtain a valid solution and store it if its total cost is lower than the cost of the best solution found so far. An example of a one-shot trajectory is shown in Figure 3.

### C. Pure-Pursuit Expansions

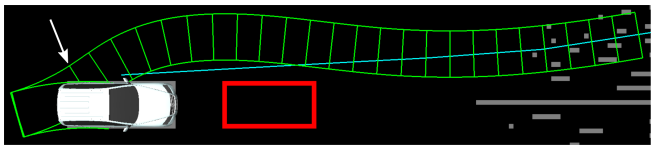
As mentioned before, the need for discretization of control inputs in A* can result in paths that are not smooth, in the sense that they contain unnatural swerves. This makes it hard to accurately follow a smooth reference path such as the center line of a lane. While finer discretization alleviates this problem, it comes at the cost of higher computational load. Other authors (e.g., [2], [6], [5]) address this issue by first planning a coarse path using A* with a limited number of motion primitives to retain efficiency. In a second step, the path is smoothed using numerical optimization. The problem with this approach is that it can fail in scenarios requiring highly precise maneuvering (see Figure 1), as the coarse discretization in the first step may prevent the planner from finding any solution at all.

This is why we developed *pure-pursuit expansions*, an alternative method of generating trajectories that smoothly follow the course of the road. Pure pursuit [19] is a well-known path-tracking algorithm that is easy to implement and tune. Moreover, its computational requirements are modest, which makes it suitable for use in combination with A*.

Given a reference path, the intersection point $p_i$ of the path and a circle of radius $l$ (the *lookahead* distance) around



(a)



(b)

Fig. 4: Pure-pursuit-based trajectories (green) in simulated scenarios (reference path: cyan). In the obstacle-free case (a), the trajectory consist entirely of one-meter pure-pursuit segments. In (b), regular A* expansions reverse and turn left in order to avoid the obstacle (red) before pure-pursuit expansions take over (indicated by the white arrow).

the vehicle's rear axle center is chosen as the next goal. [19] shows that the vehicle would reach $p$ by instantaneously assuming (and keeping) a curvature of $c_0^{\text{pp}} = \frac{2}{l^2}\Delta y$, where $\Delta y$ is the signed lateral distance between the vehicle's heading vector and $p_i$.

When performing a pure-pursuit expansion of a node with configuration $(x, y, \psi, c_0, v)$, we determine the goal point on the reference path and compute the required curvature $c_0^{\text{pp}}$ as described above. We then generate a one-meter clothoid segment starting at $(x, y, \psi, c_0)$. Its curvature rate is obtained by passing $c_0$, $c_0^{\text{pp}}$ and $v$ to *FeasibleCurvatureRate* in Algorithm 1. Next, a new goal point is chosen based on the end position of the newly added clothoid arc, and the procedure is repeated until the total length of the added arcs reaches the regular A* arc length $L$. We then evaluate the arcs' costs and add a node whose configuration corresponds to the final configuration of the last arc.

The rationale behind computing multiple short arcs is that pure pursuit needs to frequently choose new goal points to yield smooth trajectories. Figure 4 shows two examples of pure-pursuit-based trajectories.

## IV. EXPERIMENTAL RESULTS

### A. Practical Experiments

The algorithm was evaluated over the course of several weeks of daily test drives lasting multiple hours each. It was used on two autonomous vehicles equipped with standard multi-core machines. The first one was a regular electric car retrofitted with sensors and actuators for autonomous driving. It perceived its environment using two Velodyne HDL-32E LiDARs at the front as well as a Velodyne VLP-16 mounted at the back. RTK-GPS was used for accurate localization on a lane net constructed from OpenStreetMap[1] data. However, an a-priori grid map of the environment was not available, which means all obstacles had to be detected on-the-fly.

The test site was a mock urban environment consisting of narrow lanes and multiple sharp turns whose curvature exceeded the vehicle's maximum curvature, forcing it to veer off the lane's center line in order to make the turn (see Figure 1). Moreover, pedestrians, cyclists and other drivers had to be avoided. As the tests were part of an industrial project subject to a confidentiality agreement, we cannot provide images or video footage at this point.

The second test platform was our own autonomous VW Touareg MuCAR-3 (Munich Cognitive Autonomous Robot Car, 3rd Generation). As with the first vehicle, RTK-GPS was used for localization on the lane net of our test site. Static and dynamic obstacles were extracted from the point cloud of a single Velodyne HDL-64E mounted atop the vehicle.

Figure 5 shows part of a route driven during testing. While the road's dimensions were normal in this case, the scenario featured several challenges, as the vehicle had to give right of way, navigate through a chicane, react to a pedestrian jumping in front of the car, and deal with a wrong-way driver. Video footage of the test is available online[2]. As can be seen in the video, the vehicle actually followed the lane's center line reasonably closely where possible, even though the inaccurate aerial image in Figure 5 suggests otherwise.

The distance driven in this scenario was $750\,\text{m}$. In total, the planner was invoked 288 times. It was triggered whenever an obstacle appeared on the existing trajectory or after covering a distance of at least $3\,\text{m}$ on the trajectory. Goal poses were extracted from the global route $30\,\text{m}$ ahead of the vehicle. During each regular A* expansion, up to 18 child nodes were generated (9 with the same velocity, 9 with reduced velocity) when the vehicle was driving. When it was standing, only 14 new nodes were added (9 with the minimum positive velocity $1\,\frac{\text{m}}{\text{s}}$, 5 with the fixed negative velocity $-1\,\frac{\text{m}}{\text{s}}$). The arc length was set to $3\,\text{m}$. From each expanded node, the planner attempted to find one-shot trajectories to two goals: one on the same lane and one on the opposite lane for obstacle avoidance. The average times for the computation of one-shot trajectories and pure-pursuit edges were $2.5 \cdot 10^{-2}\,\text{ms}$ and $7.8 \cdot 10^{-3}\,\text{ms}$, respectively. The end point accuracy of one-shot trajectories was set to $0.1\,\text{m}$ in $x$ and $y$, $1\,\text{deg}$ in $\psi$ and $10^{-5}\,\frac{1}{\text{m}}$ in $c_0$.

On average, the planner found a feasible trajectory within less than $0.01\,\text{s}$, with a worst-case time of $0.108\,\text{s}$. As it usually finds better trajectories after this initial solution, its time-limit was set to $0.4\,\text{s}$. During this time, it found an average of $3211$ potential solutions in each cycle.

We also looked at the time needed to find a trajectory generated purely from A* expansions (a *pure A* solution*) compared to trajectories containing one-shot segments (a *one-shot solution*). On average, the first pure A* solution was found after $0.0218\,\text{s}$, whereas the first one-shot solution was already found after $0.0104\,\text{s}$, a speed-up of approximately $52\%$. In the worst case, the ratio is similar: while the planner needed $0.3213\,\text{s}$ to find the first pure A* solution, the first one-shot solution was already found after $0.1621\,\text{s}$.

[1] http://www.openstreetmap.org/

[2] http://www.mucar3.de/iros2016-planning

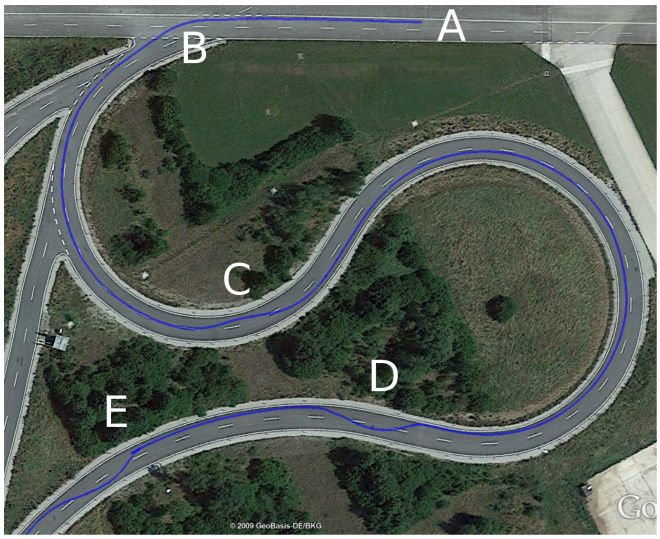



Fig. 5: Part of MuCAR-3's route (blue) during a practical experiment. (Note that inaccuracies in the aerial image wrongfully suggest that the car left its lane unnecessarily in some cases.) Starting at A, the vehicle had to give right of way to another car before making a left turn (B). At C, an obstacle forced MuCAR-3 to leave the right lane and return almost immediately to avoid a second obstacle in the left lane. Next (D), a pedestrian jumped in front of the car, forcing it to perform an emergency stop and plan an evasive trajectory. At (E), a wrong-way driver blocked the vehicle's path, causing it to stop again, reverse and move around the obstacle. (Aerial image: GeoBasis-DE/BKG, Google)

### B. Simulation Experiment

In order to analyze whether the extensions actually help in producing trajectories that closely follow the course of the road, we had a simulated vehicle drive 777 m along a mostly smooth, easily drivable reference path. (Simulation was chosen so we could guarantee that any deviation from the path was the planner's fault.) Using regular A* without extensions, the average Euclidean distance between the reference path and the vehicle's rear axle center was $0.089\,\mathrm{m}$. With one-shot expansions added, it was $0.049\,\mathrm{m}$. Using both one-shot and pure-pursuit expansions, the distance was only $0.013\,\mathrm{m}$. This shows that our extensions improve accuracy when a high-quality reference path exists.

## V. Conclusion and Future Work

We have presented a real-time trajectory planning algorithm for urban autonomous driving in highly constrained environments. Our approach combines combinatorial planning with numerical optimization and simulated control. As a result, the planner is capable of generating both complex maneuvers and smooth paths without the need for parameter adjustment or switching of algorithms by a behavioral layer. The algorithm was successfully evaluated on two different vehicles during weeks of practical experiments.

In the future, we intend to improve the algorithm's handling of dynamic objects in order to enable it to perform more aggressive maneuvers instead of acting overly cautious, as it does now. Furthermore, we would like to analyze whether the extensions described here can improve the planner's performance in unstructured environments where reliable lane net data is not available.

## References

[1] C. Urmson, J. Anhalt, *et al.*, "Autonomous Driving in Urban Environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[2] M. Montemerlo, J. Becker, *et al.*, "Junior: The Stanford entry in the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[3] T. Howard and A. Kelly, "Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[4] J. A. Reeds and L. A. Shepp, "Optimal Paths for a Car that Goes both Forwards and Backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 1961–1976, 1990.

[5] H. Andreasson, J. Saarinen, *et al.*, "Fast, Continuous State Path Smoothing to Improve Navigation Accuracy," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, Seattle, WA, USA, 2015, pp. 662–669.

[6] J.-W. Choi and K. Huhtala, "Constrained Path Optimization with Bézier Curve Primitives," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, Chicago, IL, USA, 2014, pp. 246–251.

[7] A. Bacha, C. Bauman, *et al.*, "Odin: Team VictorTango's Entry in the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[8] J. Leonard, J. How, *et al.*, "A Perception-Driven Autonomous Urban Vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[9] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, Shanghai, China, 2011, pp. 4889–4895.

[10] T. Gu, J. Atwood, *et al.*, "Tunable and Stable Real-Time Trajectory Planning for Urban Autonomous Driving," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015, pp. 250–256.

[11] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal Trajectories for Time-critical Street Scenarios Using Discretized Terminal Manifolds," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 346–359, 2012.

[12] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A Sampling-Based Partial Motion Planning Framework for System-Compliant Navigation along a Reference Path," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, Gold Coast, QLD, Australia, 2013, pp. 391–396.

[13] J. Ziegler, P. Bender, *et al.*, "Making Bertha Drive - An Autonomous Journey on a Historic Route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[14] D. Fassbender, A. Mueller, and H. J. Wuensche, "Trajectory Planning for Car-Like Robots in Unknown, Unstructured Environments," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, Chicago, IL, USA, 2014, pp. 3630–3635.

[15] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[16] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[17] A. Kelly and B. Nagy, "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control," *The International Journal of Robotics Research*, vol. 22, no. 7–8, pp. 583–601, 2003.

[18] M. McNaughton, "Parallel Algorithms for Real-time Motion Planning," Ph.D. dissertation, Carnegie Mellon University, 2011.

[19] J. M. Snider, "Automatic Steering Methods for Autonomous Automobile Path Tracking," Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Tech. Rep. CMU-RI-TR-09-08, February 2009.