2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Trajectory Planning for Car-Like Robots in Unknown, Unstructured Environments

Dennis Fassbender, André Mueller and Hans-Joachim Wuensche*

***Abstract*— We describe a variable-velocity trajectory planning algorithm for navigating car-like robots through unknown, unstructured environments along a series of possibly corrupted GPS waypoints. The trajectories are guaranteed to be kinematically feasible, i.e., they respect the robot's acceleration and deceleration capabilities as well as its maximum steering angle and steering rate. Their costs are computed using LiDAR and camera data and depend on factors such as proximity to obstacles, curvature, changes of curvature, and slope. In a second step, velocities for the least-cost trajectory are adjusted based on the dynamics of the vehicle. When the robot is faced with an obstacle on its trajectory, the planner is restarted to compute an alternative trajectory. Our algorithm is robust against GPS error and waypoints placed in obstacle-filled areas. It was successfully used at euRathlon 2013[1], where our autonomous vehicle MuCAR-3 took first place in the "Autonomous Navigation" scenario.**

I. INTRODUCTION

In the years following the 2005 DARPA Grand Challenge (DGC) as well as the 2007 DARPA Urban Challenge (DUC), a variety of methods of trajectory planning for car-like robots have been proposed. However, the bulk of the publications in this field focuses on vehicles operating in highly structured environments such as highways or urban roads, where elements like lane markings and traffic barriers guide the vehicle and constrain the planner's solution space. In contrast, the algorithm presented here was developed for autonomous navigation in unstructured, obstacle-rich environments. Although it expects a series of GPS waypoints as input, it does not assume that all of the waypoints are actually reachable or that the lines between adjacent points represent driveable paths. Even if these assumptions were correct, strictly following the path formed by the waypoints could still cause collisions due to poor GPS reception, e.g. in a forest. Instead, the algorithm relies on grid maps constructed from LiDAR and camera data to analyze its surroundings and generate trajectories that take it closer to the next waypoint.

The paper is structured as follows: Section II covers related work. The planning algorithm itself is described in detail in section III. Section IV presents experimental results gathered by running the planner on our autonomous robot MuCAR-3 (Munich Cognitive Autonomous Robot Car, 3rd Generation). Finally, section V concludes the paper and gives some ideas for future extensions.

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

[1]http://www.eurathlon.eu/

Fig. 1. MuCAR-3 navigating around obstacles at the euRathlon 2013 competition.

II. RELATED WORK

Thrun et al. [1], who won the 2005 DGC with their robot *Stanley*, used a rather simple method of trajectory planning that relied on the road data definition format (RDDF) file provided to them. Before the race started, they computed a base trajectory from the available data. In order to avoid obstacles, the vehicle's lateral offset to this base trajectory was adjusted during the race. The necessary maneuvers were computed by a search algorithm that minimized a cost function while respecting the vehicle's kinematic constraints. Urmson et al. [2] (2nd & 3rd place) similarly computed a global pre-planned path and then used A* search to find a local path in a search graph constructed from the pre-planned path. Finally, Trepagnier et al. [3] (4th place) constructed obstacle-free, dynamically feasible trajectories by adjusting the control points of a cubic B-spline that initially runs along the center of the corridor defined by the RDDF file.

Von Hundelshausen et al. [4] implemented a set of pre-computed trajectories ("tentacles") that were evaluated on the same kind of grid maps used by the algorithm described here. An early version of the planner was used by Team AnnieWAY's entry to the 2007 DUC as a low-level fallback. Later on, the approach was successfully demonstrated at the ELROB[2] competitions. However, it was only able to generate simple trajectories computed for fixed velocities. Complex maneuvers had to be handled by a higher-level module.

Ziegler et al. [5] describe an A*-based algorithm that constructs trajectories from clothoid-like arcs assuming constant speed. The algorithm was used at the 2007 DUC by Team AnnieWAY's higher-level planning module to navigate unstructured areas such as parking lots. Its A* heuristic is

[2]http://www.elrob.org/

partly based on Dubins paths [6], an idea we borrowed for our planner. In order to stay clear of obstacles, it also considers the distance to the goal along the obstacle grid's Voronoi graph.

A similar technique is employed by Dolgov et al. [7]. They generate preliminary trajectories by performing hybrid-state A* search combined with analytic expansions that use Reed-Shepp paths to move the vehicle forward in the state space. The resulting trajectory is smoothed using conjugate gradient and moved away from obstacles with the help of a Voronoi field computed on the occupancy grid. Their planner also assumes a constant velocity.

In an attempt to speed up the planning process, Likhachev et al. [8] plan trajectories based on a multi-resolution lattice on which they perform Anytime Dynamic A* search. That is, the algorithm may at first generate a feasible yet coarse trajectory which is refined as the robot keeps moving.

Trajectory generation in [8] is based on work by Howard et al. [9], who describe a general framework for rough-terrain navigation that computes trajectories that take into account terrain shape, vehicle dynamics (as specified by an arbitrary forward model that is inverted using numerical linearization), and wheel-terrain interaction. They use their approach to compute several exemplary ego-graphs for different levels of terrain roughness that can be used as a search space in local motion planning.

Our algorithm differs from the above-mentioned approaches in several ways. Rather than computing a trajectory that leads almost exactly to a certain goal pose, we simply try to move roughly in the direction of the next waypoint. This is what makes the approach robust against GPS error and waypoints surrounded by obstacles. Furthermore, by a simple A*-like search, and without the need for further optimization, we are able to generate variable-velocity trajectories that respect the kinematic constraints of the robot. Finally, our cost function does not just consider obstacles but a multitude of information about the environment, such as slopes and vegetation probabilities.

III. THE PLANNING ALGORITHM

The planner's basic task is the following: Given a 2D initial pose $p_0 = (x_0, y_0, \psi_0)$, an initial curvature $c_{0,0}$ (computed from the steering angle), an initial velocity v_0, and a 2D goal pose $p_{\text{goal}} = (x_{\text{goal}}, y_{\text{goal}}, \psi_{\text{goal}})$, construct an obstacle-free trajectory that starts at p_0, ends near p_{goal} and consists of a sequence of fixed-length clothoid arcs such that the final curvature of each clothoid arc is equal to the initial curvature of the next arc. Also, the trajectory must respect the vehicle's limitations w.r.t. steering angle, steering rate and longitudinal/lateral acceleration and end at a velocity of zero. The latter condition is necessary since the planned trajectories usually end close to the border of the grid map that represents the robot's environment. Hence, the area beyond the trajectory's end may contain unseen obstacles.

In the remainder of this section, we will first describe how the robot's environment is represented (subsection III-A), followed by a description of our collision checking procedure (subsection III-B). After the heuristic is introduced in subsection III-C, subsection III-D details how the search tree is constructed, explaining the node expansion, cost computation and a pruning technique that keeps the tree from growing excessively large. Subsection III-E presents the computation of the velocity profile, while the final subsection III-F describes how the robot is able to avoid dynamic obstacles and drive without stopping by restarting the planner from time to time.

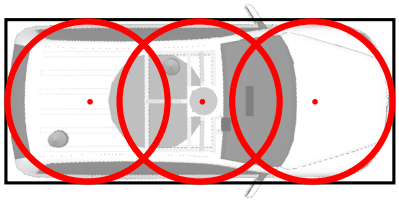

Fig. 2. MuCAR-3's bounding box (black) and the three circles used for collision checking (red). We chose not to include the vehicle's mirrors in the bounding box, which would obviously be unacceptable for passenger cars in urban environments.

A. Environment Representation

The grid map on which we evaluate potential trajectories is typically square-shaped (usually $80\,\text{m} \times 80\,\text{m}$) and consists of square-shaped cells (typically with a resolution of $20\,\text{cm}$). Each cell contains the following information: obstacle probability, height (relative to the robot), slope, vegetation probability, and road probability. These values are computed by the robot's environment mapping module, an early version of which is described in [10].

Before starting the planning process, we create a binary image whose width and height match the number of grid cells in the horizontal and vertical direction, respectively. Pixels whose corresponding grid cells have obstacle probabilities larger than 0.5 are set to 0. The remaining pixels are set to 1. We then compute the Euclidean distance transform [11] of the resulting binary image and store the product of the grid resolution (in meters) and the distance values in the grid map. Hence, each cell now contains its distance to the closest obstacle (i.e., a cell with an obstacle probability larger than 0.5).

B. Collision Checking

For collision checking purposes, the robot's 2D bounding box is approximated by three circles of equal radius r, an approach inspired by [12]. Unlike [12], however, we do not use circles that cover the whole bounding box. Instead, we choose $r = \frac{w}{2}$, where w denotes the robot's width. One circle is then placed at the bounding box's center, whereas the other two are placed on the central axis such that they touch the back and the front of the bounding box, respectively (see Fig. 2). Given this approximation, we first determine the grid cells that the centers of the three circles fall into at some robot pose p. If any of the obstacle distances stored in the cells is smaller than the sum of r and an operator-defined safety

margin d_{safety} (which we usually choose to be identical to the grid's resolution), the pose is invalid. Hence, at most three comparisons are required to check whether p is safe, which would not be the case if the complete bounding box or, say, an ellipse approximating it was checked for obstacles. Note that others (e.g., [8]) first check the bounding box's circumscribed circle for collisions. This speeds up collision checking in areas with few obstacles since no further checks need to be performed if the circumscribed circle is obstacle-free. However, it requires more time in narrow passages with obstacles close to the vehicle's sides (if there are obstacles within the circumscribed circle, the other circles still need to be checked). Since we routinely find ourselves in obstacle-rich scenarios, we decided to use the approach described above.

C. Heuristics

Like e.g. [5], [8], we compute our heuristic h by taking the maximum of two different heuristic functions, h_{dubins} and h_{obs}. h_{dubins} returns the length of the Dubins path connecting a given pose to the goal pose p_{goal}. As shown in [6], a Dubins path is the shortest path that connects two 2D poses without exceeding a given maximum curvature. It is constructed from circular arcs and line segments. However, while Dubins paths take into account the vehicle's kinematic constraints, they do not consider obstacles that may block its path. In contrast, our second heuristic h_{obs} tries to find an obstacle-free path to the goal pose without considering any kinematic constraints. It assumes that the vehicle, which is approximated by a circle of radius $\frac{w}{2}$, can turn around its center and move freely from one cell to another. It is computed as follows. Prior to the start of the planning process, the shortest obstacle-free path P from the cell corresponding to the start pose p_0 to the cell corresponding to the goal pose p_{goal} is determined by a Dijkstra search on the 8-connected occupancy grid. All cells on the path must have an obstacle distance of at least $\frac{w}{2} + d_{\text{safety}}$. If the goal pose cannot be reached, P is chosen to be the path that gets as close as possible to p_{goal}'s cell without colliding with an obstacle. Given an arbitrary pose p, we compute its distance to the closest cell c in P (again assuming an 8-connected grid). p's h_{obs} value is then given by the sum of its distance to c and the distance from c to the end of P. The two heuristics are illustrated in Fig. 3.

D. Constructing the Search Tree

The search tree is constructed on the fly. It is important to note that it is in fact a tree, not just a graph, and that we do not use true A* search but a method based on it. This is due to the fact that we do not have a single goal node that we try to reach. Rather, we usually have numerous leaf nodes that satisfy our termination condition (i.e., that represent paths that move the robot sufficiently close to the goal), in which case these nodes are not expanded anymore. The least-cost path is returned when the planner's time limit is reached. Hence, there is no guarantee that the resulting path is optimal. Unlike [8], we cannot even give a bound

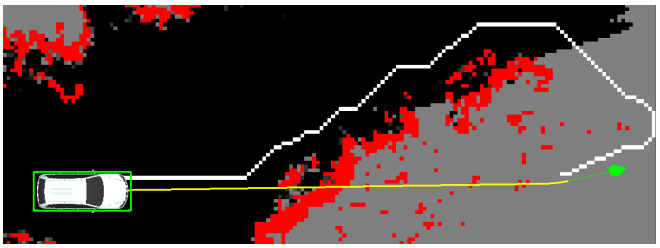

Fig. 3. The two heuristics at work. While the Dubins path (yellow) consists mostly of a line segment that leads right to the goal (green arrow), it runs through obstacles (red cells). The obstacle-aware heuristic (white) finds an obstacle-free path that is not kinematically feasible.

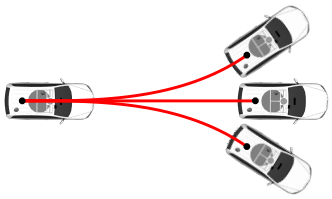

Fig. 4. Node expansion: Three new clothoid arcs are attached to the final pose of an existing node.

on the path's suboptimality due to the lack of a single goal node.

1) Preliminaries: Let $\mathcal{N}$ be the set of all nodes. Each node $N \in \mathcal{N}$ represents a clothoid arc originating from some start pose p_{N} with an initial curvature $c_{0,\text{N}}$, a rate of change of curvature $c_{1,\text{N}}$, and a terminal velocity v_{N}. Combined with the fixed arc length L, these values are sufficient to compute both the pose and the curvature at the end of the clothoid arc, which will in turn be used as the start pose and the initial curvature of any child node of N. Let $g(N)$ be the cost of the path from the root node of the search graph to N, with $h(N)$ denoting the heuristic cost of N's final pose. The value of the complete cost function $f(\mathcal{N})$ for a node N is given by $f(N) = g(N) + h(N)$. The cost of the single clothoid arc represented by N will be denoted by $c(N)$.

The search graph's leaf nodes are stored in a priority queue called OPEN, whose elements are sorted in ascending order by their f values. At the beginning, OPEN contains only the root node N_{root}. Its start pose, initial curvature and terminal velocity are simply the parameters p_0, $c_{0,0}$ and v_0 the planner was invoked with. The rate of change of curvature is set to zero, and unlike any other node, N_{root} is assumed to have an arc length of zero, meaning that its final pose is identical to its start pose. This is necessary so the node can be handled like any other node by the expansion function.

2) Node Expansion: The algorithm's main loop removes the top element N of OPEN for expansion. As mentioned before, the start pose and initial curvature of N's child nodes are predetermined by the values associated with N. Next, we compute the set V of potential terminal velocities of N's child nodes. Let v_{min}^{+} be the minimum velocity at which the robot is to drive forward (we choose $2\,\frac{\text{m}}{\text{s}}$ for our vehicle), and let v^{-} be the velocity at which it is to drive backward

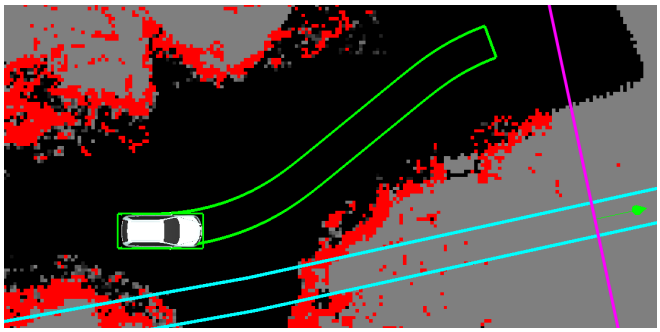

Fig. 5. MuCAR-3 navigating autonomously through a forest (same scenario as in Fig. 3). The cyan path connects the GPS waypoints. The goal pose (green arrow) lies beside the actual path and is surrounded by obstacles (red cells). The thick green path was returned by the planner. In order for a path to satisfy the termination condition, it must end close to the pink line and point roughly in the same direction as the goal pose.

($-1.5\,\frac{\mathrm{m}}{\mathrm{s}}$ in our case; no other velocities are allowed). We assume that $|v^-| \leq v^+_{\min}$ always holds. Finally, let $a^{\mathrm{x}}_{\min}$ be the strongest admissible negative longitudinal acceleration.

If $v_{\mathrm{N}} \leq v^+_{\min}$, then $V = \{v^-, v^+_{\min}\}$ (or $V = \{v^+_{\min}\}$ if driving backward is not allowed). The idea is to keep the velocities low in order to allow higher curvatures in nodes further down the search tree. Otherwise, we compute the velocity v_{reduced} the robot reaches when driving at v_{N} and then decelerating with $a^{\mathrm{x}}_{\min}$ for the next L meters by $v_{\mathrm{reduced}} = \sqrt{(v_{\mathrm{N}})^2 + 2a^{\mathrm{x}}_{\min}L}$. In case the sum under the square root is negative, we set $v_{\mathrm{reduced}} = v^+_{\min}$. If $v_{\mathrm{reduced}} < v^+_{\min}$, then $V = \{v_{\mathrm{N}}, v^+_{\min}\}$, else we set $V = \{v_{\mathrm{N}}, v_{\mathrm{reduced}}\}$. Here, we also try to sacrifice speed for the sake of higher curvatures by adding either $v^+_{\min}$ or v_{reduced}. However, we also keep the current velocity v_{N} in S since paths with high velocities and lower curvatures are preferable when the goal lies straight ahead and there are no obstacles in the way.

We then iterate over all velocities $v \in V$. Based on N's final curvature, the robot's maximum steering rate, its maximum steering angle, and v, we compute the minimum and maximum rate of change of curvature ($c_{1,\min}$ and $c_{1,\max}$, respectively) the clothoid arcs of N's children may have. Given the maximum number of children, B, (we typically use 3 for negative velocities and at least 5 for positive velocities), the rate of change of curvature increment $\Delta c_1 = \frac{c_{1,\max} - c_{1,\min}}{B-1}$ is computed. The set of admissible rates of change of curvature is then given by $C_1 = \{c_{1,0}, \ldots, c_{1,\mathrm{B}-1}\}$, with $c_{1,\mathrm{i}} = c_{1,\min} + i \cdot \Delta c_1 \;\; \forall i \in \{0, \ldots, B-1\}$.

For each $c_{1,\mathrm{i}} \in C_1$ a new node N_{i} with rate of change of curvature $c_{1,\mathrm{N_i}} = c_{1,\mathrm{i}}$ is created. Its start pose $p_{\mathrm{N_i}}$ and initial curvature $c_{0,\mathrm{N_i}}$ are identical to the final pose and the final curvature of N, while the terminal velocity $v_{\mathrm{N_i}}$ is set to v. Before the node is added to the search tree, collision checking needs to be performed. To this end, the clothoid arc represented by N_{i} is sampled with a sample distance of $1\,\mathrm{m}$ and the algorithm checks whether the poses are safe for the robot to assume using the circle-based technique described in III-B. If all of the arc's sample poses are collision-free, N_{i} will be added to the tree as a child of N *unless* it already satisfies the termination condition. In order to check whether this is the case, let $p^{\mathrm{final}}_{\mathrm{N_i}}$ be the final pose of N_{i}, and let g_{term} be the line that runs through p_{goal} and is perpendicular to p_{goal}'s yaw angle ψ_{goal} (as illustrated by the pink line in Fig. 5). The termination condition holds if $p^{\mathrm{final}}_{\mathrm{N_i}}$ has a distance of less than $(L+1)$ from g_{term} and if the absolute value of the difference of its yaw angle and ψ_{goal} is smaller than $\frac{\pi}{10}$, a value that has proven useful in practice. In this case, N_{i} is not added to the OPEN queue but to a priority queue called CLOSED, whose elements are also sorted in ascending order by their values of f.

3) Cost Function: The next step is to determine the cost $c(N_{\mathrm{i}})$ of the new clothoid arc. Based on a subset C of the grid cells covered by the robot at the final pose of N_{i}, we compute the average absolute slope ($\mu_{|\mathrm{slope}|}$), the maximum height difference (Δ_{height}), the average vegetation probability (p_{veg}), the inverse of the average road probability ($p_{\neg\mathrm{road}}$), and the set of unobserved cells $C_{\mathrm{unobs}} \subseteq C$. Furthermore, let d_{obs} be the minimum obstacle distance of any of N_{i}'s arc's sample poses that were determined during collision checking. The cost of N_{i} is then given by

$$
\begin{aligned}
c(N_i) = L\big(&w_0\mu_{|\mathrm{slope}|} + w_1\Delta_{\mathrm{height}} + w_2 p_{\mathrm{veg}} + \\
&w_3 p_{\neg\mathrm{road}} + w_4\frac{|C_{\mathrm{unobs}}|}{|C|} + w_5\frac{1}{d_{\mathrm{obs}}+1} + \\
&w_6\frac{1}{|v_{\mathrm{N_i}}|+1} + w_7(e^{w_8|c_{0,\mathrm{N_i}}|}-1) + w_9(e^{w_{10}|c_{1,\mathrm{N_i}}|}-1)\big)
\end{aligned}
$$

where $w_0, \ldots, w_{10}$ are manually tuned weights. Hence, the function tries to avoid slopes, height differences, vegetation, areas that have not been classified as road, unobserved areas, proximity to obstacles, as well as trajectories with low velocity, high curvature, or high rate of change of curvature.

4) Pruning: With a large branching factor, it is not unusual for two different nodes to have almost identical final poses and velocities, in which case the node with the higher cost g will be removed from the OPEN queue. To this end, we create a grid of the same dimension as the grids representing the robot's environment. Each grid cell stores pointers to the nodes whose final poses fall into it. When the final pose of a new node N_{new} falls into the cell, it is compared with the final poses of the other nodes pointed to by the cell. If there is a node with an almost identical final pose that has a lower value of g than N_{new}, N_{new} will not be added to the OPEN queue. Likewise, any node N_{old} will be removed from the OPEN queue if it is almost identical to N_{new} but has a higher value of g. The concept is illustrated in Fig. 6. Without this pruning technique, the search tree would grow prohibitively large.

When the planner reaches its time limit, the node at the top of the CLOSED queue is the final node of the least-cost path. We denote said path by the sequence $P = \langle N_0, \ldots, N_{\mathrm{m}}\rangle$ consisting of $m+1$ nodes. The final step in the planning process is to compute the velocity profile.

E. Computing the Velocity Profile

While the nodes in P already have terminal velocities attached to them, they do not satisfy the afore-mentioned

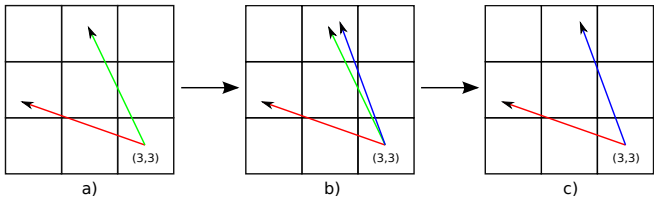

Fig. 6. Pruning the search tree. a) Cell (3,3) already points to two nodes, whose final poses are shown here. b) A third one (blue) is generated; it is almost identical to an existing node (green). c) Since its path is cheaper, the old node is removed from the OPEN queue.

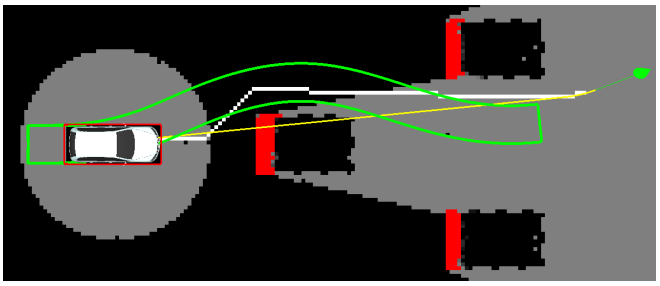

Fig. 7. Simulated scenario featuring three wall-like obstacles.

constraint $v_{\mathrm{N_m}} = 0$, which ensures that the vehicle is able to comfortably stop at the end of the trajectory. Also, some of the velocities may be unnecessarily low since our expansion function tries to reduce the velocity to allow higher curvatures further down the path. Note that N_0 is the root node with an arc length of 0, which is why we will not change its velocity. (In fact, N_0 will be removed before the final trajectory is returned.) Furthermore, we will not optimize negative velocities since we only allow one fixed velocity for driving backward.

There are three things we need to consider when computing the terminal velocity v_{N} for a node N: (1) v_{N} may not exceed the maximum velocity imposed by N's curvature and the vehicle's maximum lateral acceleration $a^{\mathrm{y}}_{\max}$; (2) If N' is a child node of N and $v_{\mathrm{N}} < v_{\mathrm{N}'}$, then the longitudinal acceleration required to accelerate the vehicle from v_{N} to $v_{\mathrm{N}'}$ may not exceed the maximum longitudinal acceleration $a^{\mathrm{x}}_{\max}$; (3) If N' is a child node of N and $v_{\mathrm{N}} > v_{\mathrm{N}'}$, then the longitudinal acceleration required to decelerate the vehicle from v_{N} to $v_{\mathrm{N}'}$ may not exceed the maximum negative longitudinal acceleration $a^{\mathrm{x}}_{\min}$. Note that $a^{\mathrm{y}}_{\max}$, $a^{\mathrm{x}}_{\max}$ and $a^{\mathrm{x}}_{\min}$ should be chosen conservatively instead of using the actual maximum values allowed by the kinematic constraints of the robot. E.g., we set $a^{\mathrm{x}}_{\max}$ to one fourth of the maximum longitudinal acceleration our vehicle is capable of.

Let $v^{\mathrm{org}}_{\mathrm{N_i}} = v_{\mathrm{N_i}}$ for all $i \in \{1, \ldots, m\}$. The values of $v_{\mathrm{N_i}}$ are adjusted in a loop starting at $i = 1$ and ending at $i = m$. If the sign of $v_{\mathrm{N_i}}$ is different from the sign of $v_{\mathrm{N_{i+1}}}$ (i.e., a stop is required), we set $v_{\mathrm{N_i}} = 0$. If $v_{\mathrm{N_i}} \leq 0$, nothing else needs to be done. Otherwise, let $c_0^{\max}$ be the maximum of the absolute values of N_{i}'s initial and final curvatures. The maximum admissible velocity allowed by the curvature $c_0^{\max}$ is given by $v_1 = \sqrt{\frac{a^{\mathrm{y}}_{\max}}{c_0^{\max}}}$. (Note that using the maximum of the initial and final curvature is a conservative way of computing the maximum velocity.) Next, we compute v_2, the maximum velocity achievable when starting at $v_{\mathrm{N_{i-1}}}$ and accelerating with $a^{\mathrm{x}}_{\max}$: $v_2 = \sqrt{(v^{\max}_{N_{\mathrm{i-1}}})^2 + 2a^{\mathrm{x}}_{\max}L}$. We then set $v_{\mathrm{N_i}} = min\{v_1, v_2\}$. When the forward iteration is completed, we iterate backward over all nodes N_{i} with $i \in \{1, \ldots, m-1\}$. For each node N_{i}, we compute the maximum velocity the robot may have at the end of N_{i}'s arc if it uses its maximum negative acceleration to reach $v_{\mathrm{N_{i+1}}}$ after L meters. This velocity is given by $v_3 = \sqrt{(v_{\mathrm{N_{i+1}}})^2 - 2a^{\mathrm{x}}_{\min}L}$. If $v_{\mathrm{N_i}} > v_3$, we set $v_{\mathrm{N_i}} = v_3$.

Our practical tests have shown that this way of computing the velocity profile causes frequent (albeit light) jerks that are a result of local maxima in the velocity profile. We remove them by setting $v_{\mathrm{N_i}} = max\{v_{\mathrm{N_{i-1}}}, v_{\mathrm{N_{i+1}}}\}$ for all $i \in \{1, \ldots, m-1\}$ with $v_{\mathrm{N_i}} > max\{v_{\mathrm{N_{i-1}}}, v_{\mathrm{N_{i+1}}}\}$. This completes the computation of the terminal velocities of each path segment. The velocity of a point on segment N_{i} ($i \in \{1, \ldots, m\}$) at a distance of s meters from N_{i}'s start pose (measured along the clothoid arc) is given by $v_{\mathrm{N_i}}(s) = v_{\mathrm{N_{i-1}}} + (v_{\mathrm{N_i}} - v_{\mathrm{N_{i-1}}})\frac{s}{L}$. This causes the vehicle to accelerate constantly on each clothoid arc. If $|v_{\mathrm{N_i}}(s)| < 1$, we set $v_{\mathrm{N_i}}(s) = 1$ if $v^{\mathrm{org}}_{\mathrm{N_i}} > 0$ and $v_{\mathrm{N_i}}(s) = -1$ otherwise. While this results in jumps in the velocity profile, it is necessary in practice to ensure that the commanded velocities are high enough to make the vehicle move. Fig. 8 shows the velocity profile computed for the path in Fig. 7.

F. Driving along the Trajectories

The trajectories returned by the planner are passed to a lower-level module which localizes the robot on the path and computes the input to the controller. Every time the environment mapping module (which runs at 10 Hz) creates a new occupancy grid, we check the trajectory for collisions by sampling poses from it and checking if any of the three circles approximating the robot's shape are too close to an obstacle. If one of these critical poses is dangerously close to the robot's current position, the planner is restarted to compute a path around the obstacle. The values for the vehicle's pose, curvature and velocity that the planner receives are predicted using the planner's maximum planning time as time horizon. Even if there are no obstacles on the path, the planning process is started again before the robot starts decelerating toward the end of the trajectory, where it would stop if no new path was found. This ensures that the robot can keep driving until it reaches its final destination rather than planning, driving to the resulting trajectory's end and stopping there, and then restarting the planning process.

IV. EXPERIMENTAL RESULTS

The planner was successfully demonstrated at euRathlon 2013, a robotics competition that took place from September 23rd-27th in Berchtesgaden, Germany. The "Autonomous Navigation using GPS, GLONASS and GALILEO" scenario required the teams to drive up a mountain road leading through a forest. Several obstacles (see Fig. 1) were placed

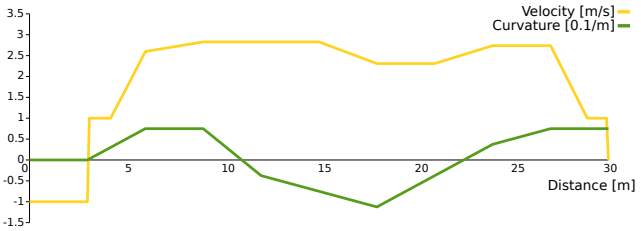

Fig. 8. Velocity (yellow) and curvature (green) of the trajectory in Fig. 7 as a function of distance driven. It can be seen that high curvatures cause the velocity to decrease.

on the road. The course was divided into three levels of difficulty. MuCAR-3 managed to complete the first two levels, requiring a single manual intervention in a narrow passage where bushes on both sides obstructed the vehicle's path. The third part of the course was too narrow for our vehicle and thus not driveable in autonomous mode. Nonetheless, we achieved the highest score and took 1st place.

At euRathlon 2013, the planning algorithm was used in combination with an improved version of the road following algorithm described in [13]. When the road tracker failed to detect an obstacle-free road segment, the planner took over. Fig. 5 illustrates the kinds of scenarios the algorithm routinely had to deal with during the tests leading up to the competition. Video footage of the tests can be found online[3].

In unpredictable environments such as forests, we found it beneficial to reduce the size of the grid maps (e.g., to no more than $60\,\mathrm{m} \times 60\,\mathrm{m}$) and thus plan shorter paths. These environments are unpredictable in that the LiDAR grids often contain large numbers of unobserved cells (the grey areas in Fig. 5) due to occlusion by vegetation, and in that the obstacle probability of some cells may fluctuate, e.g. due to high grass swaying in the wind. The velocity was usually limited to $4\,\frac{\mathrm{m}}{\mathrm{s}}$ or even $3\,\frac{\mathrm{m}}{\mathrm{s}}$ in these cases. In less dynamic environments with more free space, we typically chose a grid size of $80\,\mathrm{m} \times 80\,\mathrm{m}$ and routinely achieved velocities of up to $9\,\frac{\mathrm{m}}{\mathrm{s}}$. A planning time of $200\,\mathrm{ms}$ was usually sufficient. Reducing the branching factor generally speeds up the planning process but may prevent the planner from finding any solution in obstacle-rich environments that require precise maneuvering. When the planner failed to find a trajectory in time, its time limit was automatically increased (up to several seconds in particularly challenging scenarios) and it was restarted, allowing it to consider changes in the environment that may have occurred in the meantime.

We also conducted several tests to analyze the effects of the different heuristics and check if it is even necessary to use two heuristics. E.g., we had the planner generate paths through the scenario shown in Fig. 7 three times, with a maximum number of child nodes (per node and velocity) of 9 and a time limit of $200\,\mathrm{ms}$. First, it only used the Dubins path heuristic and found a total of 524 paths. Using only the obstacle-aware heuristic, it found 614 paths. By combining the two, a total of 670 paths was found. It is worth noting that the best path found this way (shown in Fig. 7) was in fact cheaper than the best paths found using a single heuristic. Tests in other scenarios produced comparable results, which shows that the combination of the two heuristics is in fact beneficial.

[3] http://www.mucar3.de/iros2014

V. CONCLUSION

We have presented a robust, extensively tested trajectory generation algorithm for navigation in unknown, unstructured environments. It uses a unique variant of A* to find a dynamically feasible path to the goal even in the face of GPS error and badly placed waypoints. There are several ways in which the algorithm may be extended. E.g., we would like to consider the dynamic objects detected by our LiDAR-/vision-based tracking algorithms ([14], [15]). Another idea is to employ machine learning methods in order to find good weights automatically by observing human driving behavior.

REFERENCES

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

[2] C. Urmson, C. Ragusa, *et al.*, "A Robust Approach to High-Speed Navigation for Unrehearsed Desert Terrain," *Journal of Field Robotics*, vol. 23, no. 8, pp. 467–508, 2006.

[3] P. G. Trepagnier, J. Nagel, *et al.*, "KAT-5: Robust Systems for Autonomous Vehicle Navigation in Challenging and Unknown Terrain," *Journal of Field Robotics*, vol. 23, no. 8, pp. 509–526, 2006.

[4] F. von Hundelshausen, M. Himmelsbach, *et al.*, "Driving with Tentacles: Integral Structures of Sensing and Motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008.

[5] J. Ziegler, M. Werling, and J. Schroeder, "Navigating Car-like Robots in Unstructured Environments Using an Obstacle Sensitive Cost Function," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, Eindhoven, The Netherlands, 2008, pp. 787–791.

[6] 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.

[7] 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.

[8] M. Likhachev and D. Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[9] 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.

[10] M. Himmelsbach, T. Luettel, *et al.*, "Autonomous Off-Road Navigation for MuCAR-3 – Improving the Tentacles Approach: Integral Structures for Sensing and Motion," *Kuenstliche Intelligenz*, vol. 25, no. 2, pp. 145–149, 2011, special Issue on Off-Road-Robotics.

[11] A. Rosenfeld and J. L. Pfaltz, "Distance Functions on Digital Pictures," *Pattern Recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[12] J. Ziegler and C. Stiller, "Fast Collision Checking for Intelligent Vehicle Motion Planning," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, San Diego, CA, USA, 2010, pp. 518–522.

[13] M. Manz, M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Detection and Tracking of Road Networks in Rural Terrain By Fusing Vision and LIDAR," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, 2011, pp. 4562–4568.

[14] M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Real-Time Object Classification in 3D Point Clouds Using Point Feature Histograms," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, Oct. 2009, pp. 994–1000.

[15] C. Fries, T. Luettel, and H.-J. Wuensche, "Combining Model- and Template-based Vehicle Tracking for Autonomous Convoy Driving," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, Gold Coast, Queensland, Australia, 2013.