Spline-based Trajectory Optimization for Autonomous Vehicles with Ackerman drive

Martin Gloderer **Andreas Hertle**
Department of Computer Science, University of Freiburg

Abstract

Autonomous vehicles with a car-like motion model, i.e. an Ackerman drive, can only follow trajectories with a smooth curvature. These trajectories are difficult to create with conventional planning algorithms. In this paper we present a method to create a spline-based trajectory from a given sequence of waypoints. We guess an initial trajectory and optimize it to minimize the time of travel. The resulting trajectory satisfies a number of constraints based on physical properties of the vehicle and the environment. Our experiments indicate that few iterations of our optimization algorithm are sufficient to produce valid and fast trajectories.

1 Introduction

Anyone who has tried to find its way around with the help of an automotive navigation system knows it for certain: the shortest path is in most cases not the fastest. A similar problem arises in autonomous driving: many path planning algorithms, e.g. A*, will find the shortest path, but not necessarily the fastest. These paths are constructed of waypoints connected with straight lines. Vehicles with an omni-directional drive or a differential drive can actually follow such a path in a drive-and-turn fashion, though it is highly inefficient. On the other hand, vehicles with a car-like motion model, i.e. an Ackerman drive (see Fig. 1), require a smooth trajectory without sharp turns since they are unable to turn on the spot.

In this paper we present an approach to generate a smooth trajectory suitable for vehicles with an Ackerman drive. We represent our trajectory in form of Bézier splines. We assume a static world and an obstacle-free corridor around the waypoints that the trajectory is not to leave.

Using suitable heuristics, we obtain an initial guess for the spline parameters, which is then optimized against time of travel using a modified RPROP algorithm, a gradient descent method. After optimization, the smooth trajectory satisfies a number of constraints, most notably the maximum steering angle of the vehicle and the maximum centripetal acceleration during turns. We complement the optimized trajectory with a velocity profile and a steering profile.

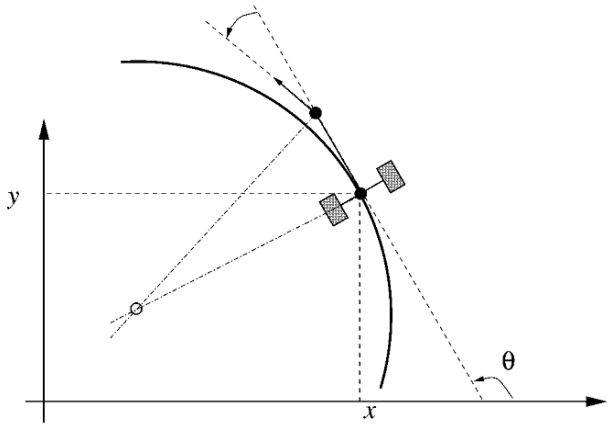


Figure 1: Model of a vehicle with an Ackerman drive [Lamiraux and Laumond, 2001].

2 Related work

In racing sports, the racing line is the route the vehicle must take in order to minimize the time taken to complete the course. By fitting a curve with the widest possible radius into each corner of the course, the higher speed which can be maintained more than compensates for the extra distance traveled.

In contrast, conventional trajectories for autonomous vehicles often simply consist of waypoints connected by straight lines. As long as we deal with a vehicle that can turn on the spot, these are feasible paths, but certainly not fastest. Several approaches were made to overcome this limitation by using curve-based trajectories to interpolate between waypoints, with ([Magid *et al.*, 2006], [S. Thrun and many others, 2006]) as well as without ([Lamiraux and Laumond, 2001]) splines.

The approach detailed in [Sprunk, 2008] belongs to the former group, employing Quintic Bézier splines to plan motion trajectories for autonomous robots with a synchro or a differential drive. This approach presents suitable heuristics for the generation of smooth trajectories as well as an optimization technique to further improve the time of travel. We extend this work to deal with the challenges specific to robots with a car-like motion model.

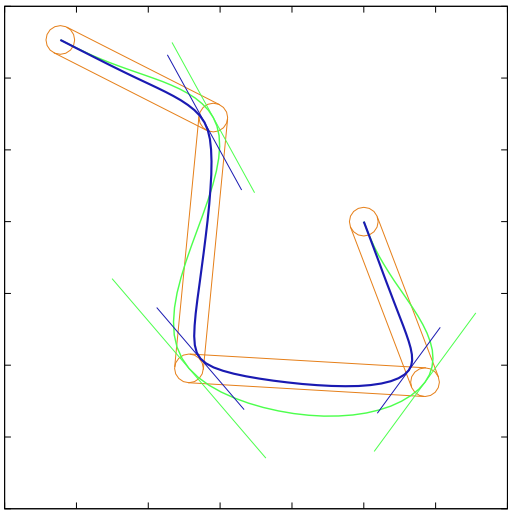


Figure 2: A smooth trajectory suitable for vehicles with Ackerman drive as produced by our planning algorithm. The distance between two tics equals $5\,\mathrm{m}$. In this example, the virtual corridor around the initial waypoints has a width of $2\,\mathrm{m}$. The thinner green curve represents the initial trajectory with tangents at the inner waypoints. Clearly the initial trajectory leaves the corridor and is therefore invalid. Respectively, the thicker blue line shows the optimized trajectory.

3 Our Approach

We want our vehicle to take the fastest path from the start point to the goal point. Therefore, the final trajectory has to fulfill two goals— the vehicle has to be able to travel along it, and it has to reach the goal point in the shortest time possible (see Fig. 2). We generate a trajectory that specifies at which position the vehicle is supposed to be at any given time during its traversal of the path. In addition, our trajectory provides velocity and steering angle information. In section 3.1 we discuss the spatial representation of the trajectory and in section 3.2 we explain how to generate the trajectory from a set of waypoints. How we ensure that the generated trajectory is traversible is shown in section 3.3. Finally we present our optimization algorithm in section 3.4.

3.1 Bézier Splines

It is common practice in robotics applications to represent trajectories with Bézier splines. A trajectory is assembled from several Bézier segments, each interpolating between two key positions. To assemble a smooth trajectory from multiple Bézier segments, two segments must have equal values at the meeting point between the two segments. The third degree Bernstein Polynomials, defining cubic Bézier segments, have four degrees of freedom: position and first derivative at start point and end point of the segment. Quintic Bézier segments, defined by Bernstein Polynomials of fifth degree, offer two more degrees of freedom: the second derivative at start point and end point. To represent 2-dimensional trajectories, we use separate Bézier splines for the dimensions x and y. An in-depth discussion of various Bézier splines types and their advantages can be found in [Sprunk, 2008].

To generate a trajectory suitable for vehicles with Ackerman drive we need curvature to be continuous along the trajectory. The curvature k of a trajectory is given as

$$k = \frac{p'_x p''_y - p'_y p''_x}{\left(p'^2_x + p'^2_y\right)^{\frac{3}{2}}}, \tag{1}$$

where p'_x and p''_x correspond to the first and second derivative of the underlying polynomials of the trajectory in x dimension, while $p'_y$and p''_y correspond to the derivative in y dimension. Cubic Bézier segments lack the possibility to adjust the second derivatives at their end points. Therefore we use Quintic Bézier segments to ensure continuous second derivatives and thus a continuous curvature along the trajectory.

We obtain the interpolated values of a Bézier segment by evaluating the polynomials and their derivatives using the internal parameter $u = [0, 1]$. $u = 0$ corresponds to the start point of a segment and $u = 1$ to its end point. The derivation of the underlying Bernstein polynomials is too lengthy to cover in this paper — for a detailed description see [Sprunk, 2008].

3.2 Trajectory Generation

In this section we describe the generation of an initial trajectory and the heuristics we employ in this process. During this section we use the subscript i to refer to the index of a waypoint, where $i = 1$ refers to the start position and $i = m$ to the goal position.

We start with a list of waypoints, assumed to be the result of a preceding path finding stage. From the list we get the positions p_i for each waypoint in our trajectory. Furthermore we need the first derivatives p'_i and second derivatives p''_i for each waypoint to fully specify the Quintic Bézier segment between two waypoints.

To obtain the first derivative for a waypoint, we estimate the tangent of the segment at the waypoint. For the inner waypoints we consider the line segment defined by the previous and the current waypoint and the line segment defined by the current and the next waypoint. We initially set the angle α of the tangent to be half way between the angles of those two line segments:

$$\alpha_i = \angle p_{i-1} p_i + \frac{\angle p_i p_{i+1} - \angle p_{i-1} p_i}{2}. \tag{2}$$

The estimation of the tangent length has tremendous influence on the quality of the generated trajectory. If the value is too low, the resulting trajectory will have a sharp turn at the waypoint. On the other hand if the value is too high, the trajectory will have sharp turns half way between two waypoints. We determined experimentally a good value for the tangent length l to be the length of the smaller of the two segments:

$$l_i = min\left(\|p_{i-1} - p_i\|, \|p_i - p_{i+1}\|\right). \tag{3}$$

We assume that the vehicle starts in the direction of the second waypoint. Since for the first waypoint of the trajectory we have only one line segment to consider, the above equations simplify to

$$\alpha_1 = \angle p_1 p_2, \tag{4}$$

$$l_1 = \|p_1 - p_2\|. \tag{5}$$

The equations for the last waypoint are analogous.

With the tangent angle and length we compute the first derivatives at the waypoints as

$$p_i' = l_i * \begin{bmatrix} \cos(\alpha_i) \\ \sin(\alpha_i) \end{bmatrix}. \tag{6}$$

The heuristic we used for the second derivative is less intuitive and too lengthy to cover here— for a detailed description see [Sprunk, 2008].

At a later stage we want to optimize our trajectory with regard to the time of travel. For each waypoint we can change the position p_i, the first derivative p_i' and second derivative p_i'' in the dimensions x and y, resulting in six degrees of freedom per waypoint. Each additional degree of freedom increases the computation time of the optimization process—therefore we optimize only over the most influential parameters. Furthermore, we determined experimentally that our heuristic for the second derivative already gives us good results which require no further optimization.

As our optimized trajectory remains inside the fixed virtual corridor, we expect little improvements when changing the angle α of the tangent. In contrast, the length l of the tangent has significant influence on the quality of the resulting trajectory, as stated above. Additionally, we add an offset to the position p_i of each waypoints when generating the trajectory. We represent this offset in local coordinates to get a more intuitive representation—we can move each waypoint in direction of and perpendicular to its tangent. To summarize, this leaves us with three parameters per waypoint to optimize:

- position offset in tangent direction,
- position offset perpendicular to the tangent,
- offset of the length of the tangent.

All parameters are initialized with zero. For obvious reasons, we exclude the start and the goal point from the optimization process. Thus, the dimensionality of our optimization problem amounts to $3 \cdot (m-2)$, where m is the number of waypoints.

3.3 Trajectory Evaluation

Once we have a trajectory, we must ensure that our vehicle can actually drive along it. We consider a trajectory valid if it satisfies the following constraints:

- Corridor distance d_{max}: The trajectory must remain inside a virtual corridor around the straight line between the waypoints. This corridor is assumed to be free of obstacles and d_{max} to be half the width of the corridor.
- Steering angle φ_{max}: The vehicle has a maximum steering angle that it cannot exceeded.
- Velocity v_{max}: The vehicle cannot not drive faster than its maximum velocity.
- Acceleration a_{max}: The vehicle can not accelerate or decelerate faster than its specified limits.
- Centripetal acceleration $a_{\text{centripetal}}$: The centripetal acceleration of the vehicle along the trajectory must not exceed a specified limit to avoid skidding out of the turn.

To ensure that all constraints are met we inspect them at regular intervals along the trajectory. We start at the first Bézier segment and sample points at equal intervals Δu of the internal Bézier parameter $u \in [0, 1]$. We proceed in such a manner with each Bézier segment. During this section we use the subscript j to refer to the index of a sample point, where $j = 1$ refers to the start position of the trajectory and $j = n$ to the goal position.

Immediate data: First we compute data that does not require knowledge of values of surrounding sample points. We evaluate the Bézier equations at the sample points to get the position p_j, the first derivative p_j' and second derivative p_j''. We use the position information p_j to calculate the distance d_j to the center of the corridor. In addition we calculate the distance Δs_j between each sample point at index j and its predecessor at index $j-1$. We compute the curvature k_j at each sample point as defined in Eq. (1). To compute the steering profile we need to consider the length of the vehicle l_{vehicle}, which is the distance between the back wheels and the front wheels. The relation between steering angle φ and curvature k is as follows:

$$\varphi_j = \arctan(l_{\text{vehicle}} \cdot k_j) \tag{7}$$

To ensure that the vehicle never exceeds the maximum centripetal acceleration $a_{\text{centripetal}}$, we limit its maximum allowed velocity $v_{j,\text{allowed}}$ according to the curvature k_j at each point. The relation between the centripetal acceleration and the velocity of the vehicle is described by

$$a_{\text{centripetal}} = k \cdot v^2 \tag{8}$$

Solving this equation for the velocity allows us to compute the admissible velocity

$$v_{j,\text{adm}} = \min\left(v_{\text{max}}, \sqrt{\frac{a_{\text{centripetal}}}{k_j}}\right) \tag{9}$$

Dependent data: At this point we know the arc length Δs_j between each two subsequent sample points along the trajectory. Also, we have determined allowed velocity $v_{j,\text{adm}}$ for each sample point. With these values we construct the velocity profile in two passes. The forward pass considers only the acceleration of the vehicle. We start at the beginning of the trajectory with the initial velocity. Then we assume constant acceleration between two sample points. Eq. (10) shows how we compute the velocity $v_{j,\text{fw}}$ of the vehicle for each sample point.

$$v_{j,\text{fw}} = \min\left(v_{j,\text{adm}}, \sqrt{(v_{j-1,\text{fw}})^2 + 2 \cdot a_{\text{max}} \cdot \Delta s_j}\right) \tag{10}$$

Analogous to the forward pass, the backward pass considers only the deceleration of the vehicle. We start at the end of the trajectory with the given final velocity and compute for each sample point the velocity as

$$v_j = \min\left(v_{j,\text{fw}},\ \sqrt{(v_{j+1})^2 + 2 \cdot a_{\max} \cdot \Delta s_{j+1}}\right) \quad (11)$$

In the last step we compute the time the vehicle requires to reach the goal point, as shown in equations (12) and (13).

$$\Delta t_j = \frac{2 \cdot \Delta s_j}{v_{j-1} + v_j} \quad (12)$$

$$t = \sum_{j=1}^{n} \Delta t_j \quad (13)$$

Finally, we have the velocity profile and the steering profile, both with respect to time. Figure 3 shows the velocity and steering profile of the trajectory that can be seen in Fig. 2.

Cost function: We want to optimize the generated trajectory with respect to time of travel. Also we want a trajectory that satisfies the steering angle constraint and the corridor distance constraint. Therefore our cost function takes these into account and gives trajectories with constraint violations a very high penalty. We use an exponential function as displayed in equation (14). This function returns values close to zero for $0 \leq c \leq 1$ and exponentially growing values for $c > 1$.

$$\text{penalty}\,(c) = \exp\left(25 \cdot (c - 0.9)\right) \quad (14)$$

Most initial trajectories violate some of our constraints. Our tangent heuristic was specifically chosen to avoid steering angle constraint violations. As a drawback, the trajectory usually leaves the designated corridor. The trajectory can never violate the constraints for acceleration, velocity or centripetal acceleration, as these depend on the velocity profile.

Equation (15) shows the cost function we use to rate a trajectory. t stands for the time the vehicle requires to move from start to goal, φ_j corresponds to the steering angle at each sample point along the trajectory and φ_{max} represents the maximum steering angle of our vehicle. Analogously, d_j stands for the corridor distance at each point and d_{max} for the maximum allowed distance.

$$E = t + \sum_{j=1}^{n} \left(\text{penalty}\left(\frac{\|\varphi_j\|}{\varphi_{max}}\right) + \text{penalty}\left(\frac{\|d_j\|}{d_{max}}\right)\right) \quad (15)$$

3.4 Trajectory Optimization

For the optimization of the trajectories we use an algorithm inspired by resilient back propagation (RPROP) [Riedmiller and Braun, 1994]. RPROP was originally proposed as a learning algorithm for artificial neural networks, but it can be used for any task where the minimum or maximum of a function is sought. In contrast to other optimization algorithms such as gradient descent, RPROP does not take the magnitude of the derivative into account, but only whether the sign of the partial derivatives has changed since the last update to avoid the harmful influence of the size of the gradient apparent in vanilla gradient descent.

In contrast to the original implementation of RPROP, we decided against first calculating the partial derivatives for all dimensions k of the parameter vector x and only then updating the trajectory parameters, and instead chose to treat each parameter as a one-dimensional optimization problem. This further accelerates the convergence of our optimization procedure by allowing for substantially more trajectory parameter updates per optimization step compared to batch learning while nonetheless yielding highly satisfactory trajectories, as we show in the discussion of the experimental results in chapter 4.

To avoid getting stuck in local minima over-optimizing one parameter and neglecting all others, we consecutively optimize each parameter in turn. Thus, an optimization step of our algorithm is performed as follows. For each parameter x^k of the trajectory parameters described in section 3.2 we use the RPROP algorithm to find a value that results in a better trajectory, i.e. a trajectory which reduces the cost of the trajectory when evaluated with Eq. 15. As soon as we found the first such value, we stop the optimization of this parameter, generate a new trajectory with the updated parameter values and move on to the next parameter. In the case that RPROP could not find a better value for the current parameter, we also move on to the next parameter but do not generate a new trajectory as we couldn't improve on the previous one by changing the value of the current parameter.

The two steps of the update rule of RPROP as used in this paper are as follows. First, we compute the value of x_n^k at iteration n as

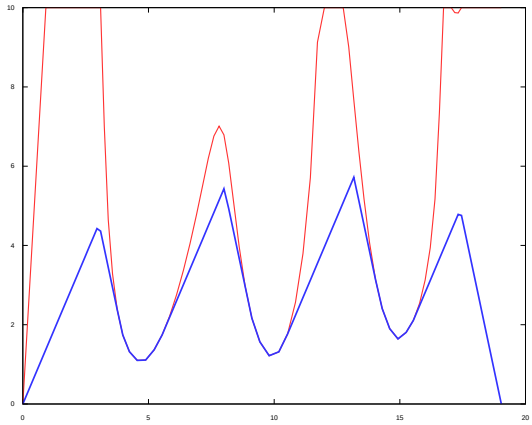
$$x_n^k = x_{n-1}^k + \begin{cases} -\Delta_{n-1}^k & \text{if } \frac{\partial E^{n-1}}{\partial x^k} > 0 \\ +\Delta_{n-1}^k & \text{if } \frac{\partial E^{n-1}}{\partial x^k} < 0 \\ 0 & \text{else,} \end{cases} \quad (16)$$

where $\frac{\partial E^{n-1}}{\partial x^k}$ is short for $\frac{\partial E^{n-1}}{\partial x^k}(x_n^k)$. From the new parameter vector x_n we can then easily calculate the partial derivative $\frac{\partial E^n}{\partial x^k}$. Using $\frac{\partial E^n}{\partial x^k}$, we compute the update value Δ_n^k for this parameter as

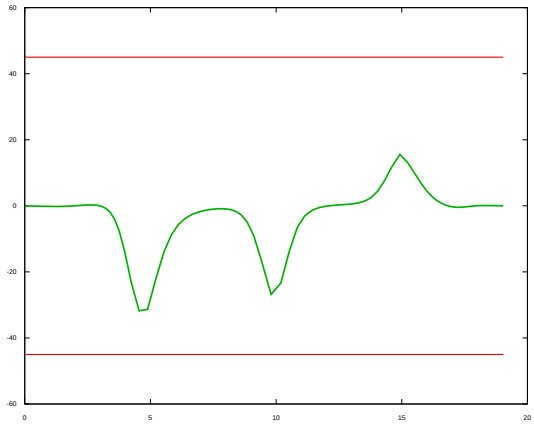
$$\Delta_n^k = \begin{cases} \eta^+ \Delta_{n-1}^k & \text{if } \frac{\partial E^{n-1}}{\partial x^k} \cdot \frac{\partial E^n}{\partial x^k} > 0 \\ \eta^- \Delta_{n-1}^k & \text{if } \frac{\partial E^{n-1}}{\partial x^k} \cdot \frac{\partial E^n}{\partial x^k} < 0 \\ \Delta_{n-1}^k & \text{else,} \end{cases} \quad (17)$$

where $0 < \eta^- < 1 < \eta^+$. If the update value Δ_n^k becomes smaller or greater than a predefined threshold $\Delta_{\min}$ resp. $\Delta_{\max}$, the RPROP algorithm terminates.

As can be seen in Eq. 17, the new update value Δ_n^k depends not on the size of the partial derivatives, but instead only on whether the sign of the current partial derivative $\frac{\partial E^n}{\partial x^k}$ has stayed the same compared to the sign of the previous partial derivative $\frac{\partial E^{n-1}}{\partial x^k}$. If it has, we are still moving towards a local minimum and we increase the update value by the factor



(a) The velocity profile in $^{\mathrm{m}}/_{\mathrm{s}}$ against time of travel in s.



(b) The steering profile in degree against time of travel in s.

Figure 3: The velocity and steering profile of the vehicle along the optimized trajectory (fig.2). The thick blue line shows the current velocity (a). The thick green line shows the current steering angle (b). The thin red lines indicate the constraints on each control, i.e. the maximum velocity above which the vehicle would skid out of the turn (a) and the maximum steering angle of the vehicle (b). The vehicle has a maximum acceleration of $1.5\,^{\mathrm{m}}/_{\mathrm{s}^2}$, a maximum deceleration of $3\,^{\mathrm{m}}/_{\mathrm{s}^2}$ and a maximum steering angle of $45°$, as described in section 4.

η^+ to achieve faster convergence. A sign change of the partial derivative indicates that the last update step was too big and we have jumped over a local minimum, therefore we reduce the update value by the factor η^- and invert the direction of the next update.

We iterate over the two steps of this update rule until our termination condition, i.e. finding a parameter value that improves the trajectory, is satisfied or until RPROP terminates, as previously described.

At the start of each single parameter optimization, x_0 is initialized with the parameters of the current trajectory. For $\frac{\partial E^0}{\partial x^k}$ and Δ_0^k there are mainly two reasonable variants, initializing them with a fixed value and initializing them with the value they were left at the previous optimization step for parameter k. For a comparison between these variants, see the experimental results in chapter 4.

4 Experimental Results

As described in section 3, our goals were to find trajectories that are valid, i.e. don't leave the corridor and respect the vehicle's maximum steering angle, as well as minimize the time of travel. To test our method we performed a series of simulated experiments implemented in GNU Octave, a high-level language primarily intended for numerical computations.

Waypoint generation. Our optimization algorithm expects a list of waypoints as input. We start at an arbitrary location and orientation. To compute the next waypoint in the list we add a random value between $-120\,°$ and $+120\,°$ to the orientation and move in the direction of the new orientation by a random value between $5\,\mathrm{m}$ and $20\,\mathrm{m}$. For the following experiments the number of waypoints was set to 5.

Input parameters. For the simulated experiments a vehicle of the length $0.75\,\mathrm{m}$ was assumed with a maximum velocity of $10\,^{\mathrm{m}}/_{\mathrm{s}}$ and a maximum acceleration and deceleration of $1.5\,^{\mathrm{m}}/_{\mathrm{s}^2}$ resp. $3\,^{\mathrm{m}}/_{\mathrm{s}^2}$. The maximum steering angle executable by the vehicle was set to $45°$. We constrained the maximum centripetal acceleration to $1\,^{\mathrm{m}}/_{\mathrm{s}^2}$ and the allowed distance to the center of the corridor to $1\,\mathrm{m}$.

Optimization parameters. We received good experimental results with most of the optimization parameters' values proposed in the original implementation of RPROP. We kept the parameters $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_{\text{min}} = 0.000001$ and $\Delta_{\text{max}} = 50$. We increased the initial update value Δ_0 from 0.01 to 0.5 to achieve a faster convergence.

Results. Data gathered during 1000 trial runs indicates that the optimization algorithm is able to cope with a wide range of initial waypoint configurations. While slightly less than two third of the initial trajectories of the trial runs violated some constraint (see Fig. 4), after less than ten steps of our optimization algorithm this number shrunk to $10\,\%$. After 13 optimization steps, only $2.5\,\%$ of the generated trajectories remained invalid.

We repeated the experiment three times with different variants of the algorithm.

1. First we optimized only the position offset perpendicular to the tangent and the length of the tangent of each waypoint ignoring the position offset in tangent direction. This was done to reduce the number of parameters of the optimization and therefore the computation time of each optimization step. The update value Δ^k was re-

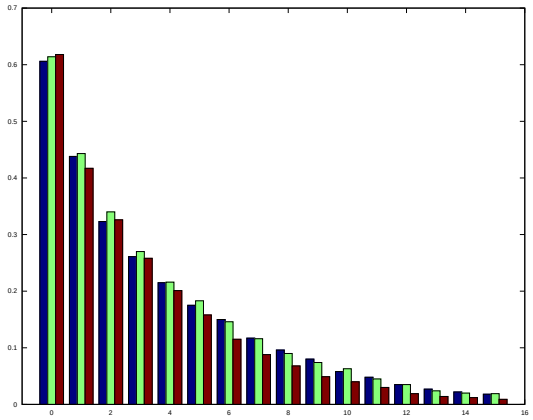


Figure 4: Percentage of remaining invalid trajectories after each optimization step. The data gathered during 1000 trial runs for each experiment shows how successful the optimization algorithm satisfied constraints that were violated by the initial trajectory. The left, middle and right bar in each group corresponds to the results of the first, second and third experiment, which are described in section 4.

set to its default value of 0.5 at the start of each new optimization step.

2. In the second experiment we optimized the same set of parameters as in the first experiment. This time, Δ^k was not reset to its default value and instead continued where it was left off at the previous optimization step.
3. The third experiment is a variant of the second one that also optimizes the position offset in tangent direction of each waypoint to explore the influence of this parameter.

The results displayed in Fig. 4 show that the first two experiments perform equally well with only insignificant fluctuations. Therefore the choice whether to reset Δ^k to its initial value has no significant influence on the overall performance of the optimization.

Including the third parameter in the optimization problem resulted in a slight improvement over the variants that optimize only two parameters. This improvement however is bought with a significant increase in computation time by roughly $50\,\%$ when moving from two parameters to three, thus resulting in a worse overall performance than the other variants.

After 15 optimization steps of our algorithm not only were the most trajectories valid, but also close to the optimal solution, as can be seen in Fig. 2.

5 Conclusions

In this paper we presented an approach to generate a smooth trajectory suitable for vehicles with an Ackerman drive. Starting with a number of waypoints our approach guesses an initial trajectory. The trajectory contains all necessary data for the vehicle: a 2d path, a velocity profile and a steering profile. We optimize the initial trajectory with a modified version of the RPROP algorithm. Our experiments show that only after as few as 13 optimization steps, $97.5\,\%$ of all trajectories we examined satisfied all constraints.

Our approach does not rely on a specific technique to acquire the initial waypoints. Thus it is possible to combine the approach with any waypoint generating planner. Our trajectory optimization algorithm is easy to adapt to a variety of vehicles—the values of the constraints can be changed to reflect the properties of the vehicle.

During the implementation phase we discovered two limitations of our approach. Since we consider each parameter independently during optimization, we can end up in deadlock-like situations where the trajectory is improved slowly at best. Changing any single parameter value results in a worse trajectory, therefore the changes are discarded. Only when changing multiple parameters simultaneously the trajectory can be improved.

The second limitation results from our kinematics model of the Ackerman drive. As our current approach only allows for driving forward, a certain combination of waypoints can lead to unsatisfiable constraints—it is impossible to remain inside the corridor while satisfying the steering angle constraint.

At this point we see three possible directions to extend our approach. Instead of the virtual corridor a map of the environment could be used to more accurately describe the limited mobility of the vehicle. This would allow the vehicle to exploit available free space beyond the fixed width corridor. To avoid parameter dead-locks during optimization, a batch optimization strategy could be implemented. Finally, the kinematics model of the vehicle could be extended to allow for backward driving.

To summarize, we presented an approach that is able to generate a smooth motion trajectory from a set of given waypoints for vehicles with an Ackerman drive. The resulting trajectory takes the kinodynamic properties of the vehicle into account while minimizing the time of travel.

References

[Lamiraux and Laumond, 2001] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. *IEEE Transactions on Robotics and Automation*, 17:498–502, 2001.

[Magid *et al.*, 2006] Evgeni Magid, Daniel Keren, Ehud Rivlin, and Irad Yavneh. Spline-Based Robot Navigation. In *IROS*, pages 2296–2301. IEEE, 2006.

[Riedmiller and Braun, 1994] M. Riedmiller and H. Braun. RPROP – Description and Implementation Details, 1994.

[S. Thrun and many others, 2006] M. Montemerlo S. Thrun and many others. The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23:661–692, June 2006.

[Sprunk, 2008] Christoph Sprunk. Planning Motion Trajectories for Mobile Robots Using Splines. October 2008.