2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Real-time Trajectory Optimization for Autonomous Vehicle Racing using Sequential Linearization*

Bassam Alrifaee[1] and Janis Maczijewski[2]

Abstract— **This paper presents a real-time Model Predictive Controller (MPC) for racing trajectory optimization. The vehicle must respect its dynamic limitations and the track boundaries while simultaneously minimizing lap times. Due to the track boundary constraints, the feasible set of the optimization problem is non-convex. This paper presents the method of sequential linearization to solve this non-convex optimization problem.**

I. Introduction

This paper approaches the essential problem of autonomous vehicle racing: How to guide a vehicle as quickly as possible along a race track? This requires an algorithm that can plan the vehicle's trajectory and simultaneously control the vehicle to follow this trajectory reliably. The trajectory must not only remain inside the track boundaries but also account for the vehicle's maximum speed and acceleration. While respecting these constraints, the trajectory should also minimize the time to reach the end of the track. Additionally, the trajectory should be updated regularly as the vehicle progresses along the race track. This imposes tight limits on the available amount of time to compute an optimal trajectory. The trajectory planning task falls into the category of kinodynamic planning problems in which both kinematic and dynamic constraints must be satisfied [1].

The trajectory planning task is formulated as an optimization problem using ideas from Model Predictive Control (MPC). The resulting optimization problem is non-convex due to the non-convex shape of a race track. Non-convex optimization problems in general are NP-hard [2].

To fulfill this challenging combination of requirements, this paper uses the mathematical formulation of a linearly constrained convex quadratic optimization problem (QP) as building block. Much prior work – both theoretically [3] and in software implementations [4], [5] – has been done to solve QPs. As a result, highly efficient and optimized software libraries for solving QPs are available. The key contribution of this paper lies in developing a QP approximation of the trajectory optimization problem and solving it iteratively. Finally, the developed method is tested against a high-fidelity vehicle dynamics simulation.

*Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - GRK 1856.

[1]Bassam Alrifaee is with the Chair of Embedded Software, Department of Computer Science, RWTH Aachen University, 52074 Aachen, Germany Bassam.Alrifaee@rwth-aachen.de

[2]Janis Maczijewski is with the Chair of Embedded Software, Department of Computer Science, RWTH Aachen University, 52074 Aachen, Germany Janis.Maczijewski@rwth-aachen.de

II. Trajectory Optimization Problem

To limit the complexity of the trajectory optimization problem we assume the following: (i) The race track is horizontal, i.e., there are no changes in elevation. This allows us to ignore the vertical vehicle dynamics. (ii) The vehicle side slip angle is negligible, i.e., the vehicle travels forwards. This allows us to ignore the change in the vehicle's dynamic behavior as it transitions from normal driving into drifting, spinning or sliding. (iii) There are no changes in vehicle and environmental conditions, e.g., weather, fuel mass, tire temperature, road surface properties. (iv) At all times the vehicle position, velocity, acceleration, and yaw angle are known through sensors and state estimation.

A. Coordinate Systems and Symbols

Fig. 1 and Tab. I illustrate the coordinate systems and variables used throughout this paper.

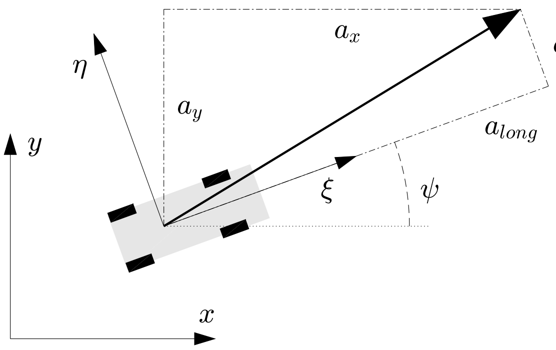

Fig. 1: Coordinate systems

TABLE I: Variables

x, y	Global coordinate system
ξ, η	Vehicle coordinate system
ψ	Vehicle yaw angle
$[a_{long}, a_{lat}]$	Vehicle acceleration vector in vehicle coordinates
$\boldsymbol{u} = [a_x, a_y]$	Vehicle acceleration vector in global coordinates
$\boldsymbol{v} = [v_x, v_y]$	Vehicle velocity vector in global coordinates
$v = \sqrt{v_x^2 + v_y^2}$	Vehicle speed
$\boldsymbol{p} = [p_x, p_y]$	Vehicle position vector in global coordinates
$\boldsymbol{x} = [p_x, p_y, v_x, v_y]$	Vehicle state vector

B. System Overview

Fig. 2 illustrates how the trajectory optimization interacts with the controlled vehicle. From the perspective of the trajectory optimization, the vehicle is modeled as a point mass where the acceleration vector $\boldsymbol{u} = [a_x, a_y]$ is the input variable. The choice of using the acceleration vector as the input variable is useful as it leads to a linear model in the optimization problem.

In order to realize the acceleration commands, an underlying control loop manipulates the steering angle, gas position, and brake position to match the reference acceleration $[a_{x,ref}, a_{y,ref}]$ given by the trajectory optimization.

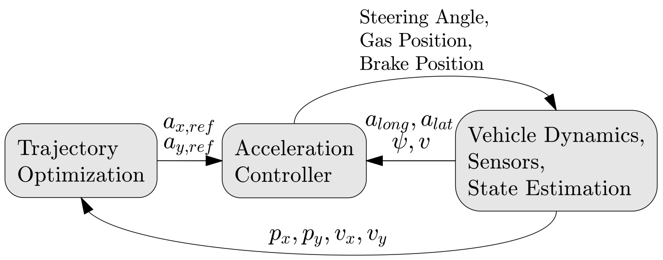

Fig. 2: System signal flow graph

C. Acceleration Constraints

The sum of all forces and thus the acceleration magnitude on a vehicle is always limited. The precise limits of those forces depend on many variables. Some of those variables may for example be tire pressure, suspension stiffness or the shape of aerodynamic surfaces. The relevant set of variables depends on the model used to describe the forces. More complex and accurate models will have more variables. Without choosing a particular model, one can assume that the maximum acceleration magnitude depends on the vehicle speed and acceleration direction. To give a concrete example, a vehicle driving at its maximum speed can accelerate backwards but can by definition not accelerate forwards. We assume any other variables that may affect the maximum acceleration magnitude to be constant. With this assumption the following acceleration dynamics can be modeled:

- The maximum forward acceleration decreases with increasing speed due to aerodynamic drag.
- The maximum backward acceleration increases with increasing speed due to aerodynamic drag.
- The maximum lateral and backwards acceleration increases with increasing speed due to aerodynamic downforce.

The acceleration limits should be considered in the trajectory optimization to ensure that the reference acceleration $[a_{x,ref}, a_{y,ref}]$ requested from the acceleration controller is physically realizable.

We empirically determined the maximum forward, backward and lateral accelerations for different speeds in a simulation. Piecewise linear functions $a_{forward,max}(v)$, $a_{backward,max}(v)$, $a_{lat,max}(v)$ interpolate these measurements, see Fig. 3. To determine the maximum acceleration for arbitrary directions at a particular speed, two half-ellipses are used as illustrated in Fig. 4. This is motivated by existing models where the maximum horizontal forces that can be transferred by a tire form an ellipse [6]. The lengths of the semi axes for the forward and backward directions are

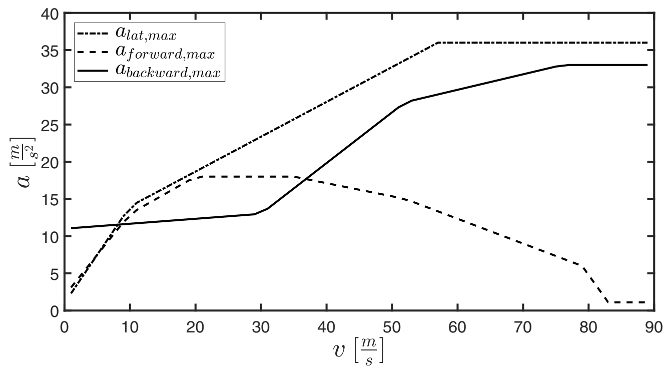

Fig. 3: Maximum accelerations for varying speeds

allowed to be different to account for backward acceleration due to aerodynamic drag. A more precise model is certainly possible. We consider this model as a good trade-off between complexity and precision. These acceleration limits can be written as:

$$c(a_{long}, a_{lat}, v) = \left(\frac{a_{long}}{a_{long,max}(v)}\right)^2 + \left(\frac{a_{lat}}{a_{lat,max}(v)}\right)^2 - 1 \le 0 \tag{1}$$

$$a_{long,max}(v) = \begin{cases} a_{forward,max}(v) & \text{if } a_{long} > 0 \\ a_{backward,max}(v) & \text{if } a_{long} \le 0 \end{cases} \tag{2}$$

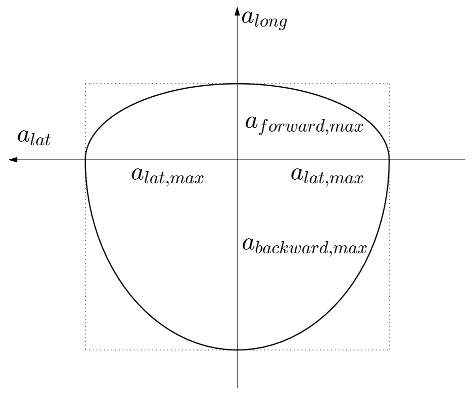

Fig. 4: Two half-ellipses model the maximum accelerations for varying acceleration directions

These acceleration limits are formulated in the vehicle coordinate system, but the trajectory optimization requires them in the global coordinate system. The coordinate transformation uses the vehicle yaw angle. The yaw angle is not an optimization variable, but it can be determined from the velocity vector as follows:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \sqrt{v_x^2 + v_y^2} \begin{bmatrix} \cos(\psi) \\ \sin(\psi) \end{bmatrix}. \tag{3}$$

This uses the assumption that the vehicle side slip angle is negligible. Thus, the transformation can be written as:

$$\begin{bmatrix} a_{long} \\ a_{lat} \end{bmatrix} = \frac{1}{\sqrt{v_x^2 + v_y^2 + \varepsilon}} \begin{bmatrix} v_x & v_y \\ -v_y & v_x \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}. \tag{4}$$

Note that this transformation fails at $(v_x, v_y) = (0,0)$, because the yaw angle can not be inferred at zero velocity. The small constant ε is introduced to avoid division by zero. As a consequence the acceleration constraint can be reduced to $-1 \leq 0$ which effectively removes it. Substituting (4) in (1) yields the final acceleration constraint:

$$c_{acc}(\boldsymbol{v},\boldsymbol{u}) = \frac{1}{v_y^2 + v_y^2 + \varepsilon}\left(\left(\frac{a_x v_x + a_y v_y}{a_{long,max}\left(\sqrt{v_x^2+v_y^2}\right)}\right)^2 + \left(\frac{a_y v_x - a_x v_y}{a_{lat,max}\left(\sqrt{v_x^2+v_y^2}\right)}\right)^2\right) - 1 \leq 0. \tag{5}$$

D. Model Predictive Control

The trajectory optimization relies at its core on the idea of Model Predictive Control (MPC). In MPC, a model of the controlled system is used to predict the system behavior in the near future for a given control input. The vehicle is modeled as a point mass, where the acceleration vector is the controlled input. The following equations describe the used model:

$$\dot{p}_x = v_x, \quad \dot{p}_y = v_y, \quad \dot{v}_x = a_x, \quad \dot{v}_y = a_y, \tag{6}$$

where all variables are defined in section II-A. The (6) are ordinary differential equations. They are solved analytically for a time-step $\Delta t \in \mathbb{R}^+$ and formulated in a discrete state space form as follows:

$$\underbrace{\begin{bmatrix} p_x^{(i)} \\ p_y^{(i)} \\ v_x^{(i)} \\ v_y^{(i)} \end{bmatrix}}_{=\,\boldsymbol{x}^{(i)}} = \underbrace{\begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{=\,A} \underbrace{\begin{bmatrix} p_x^{(i-1)} \\ p_y^{(i-1)} \\ v_x^{(i-1)} \\ v_y^{(i-1)} \end{bmatrix}}_{=\,\boldsymbol{x}^{(i-1)}} + \underbrace{\begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}}_{=\,B} \underbrace{\begin{bmatrix} a_x^{(i)} \\ a_y^{(i)} \end{bmatrix}}_{=\,\boldsymbol{u}^{(i)}}, \tag{7}$$

where $A \in \mathbb{R}^{4\times4}$ is the system matrix, $B \in \mathbb{R}^{4\times2}$ is the input matrix and $\boldsymbol{x}^{(i)} \in \mathbb{R}^4$ and $\boldsymbol{u}^{(i)} \in \mathbb{R}^2$ are the state and input vectors. Here a superscript in parentheses – for example i in $\boldsymbol{x}^{(i)}$ – denotes a temporal index. Each state vector applies to a particular point in time. Each input vector applies to the time span between two states. The (7) is repeated for $i \in \{1, 2, \ldots, H_p\}$, where $H_p \in \mathbb{N}$ is the prediction horizon.

E. Race Track Model

The race track model consists of two parts, the center-line $\boldsymbol{C}$ and the track area T. The center-line is used to measure the vehicle's progress along the track. The track area defines where the vehicle is allowed to be. Fig. 5 illustrates the center-line and the track area.

The track area $T \subset \mathbb{R}^2$ is a connected set of all points where the vehicle is allowed to be:

$$\boldsymbol{p}^{(i)} \in T \quad \forall i = 1, \ldots, H_p. \tag{8}$$

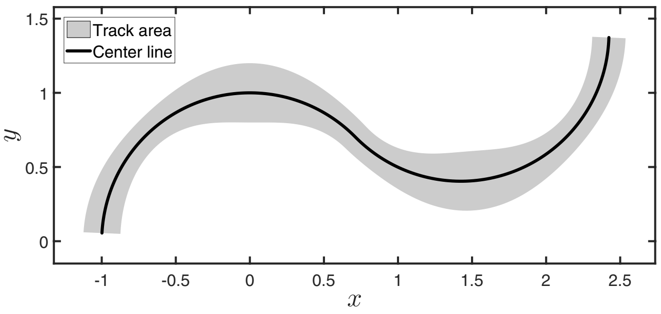

Fig. 5: Example of a race track model

The center-line is a differentiable curve $\boldsymbol{C} : [0, S] \to T$. $\boldsymbol{C}$ is parameterized by its arc length, i.e.:

$$\int_0^s \left\| \frac{d\boldsymbol{C}}{ds}(u) \right\|_2 du = s \qquad \forall s \in [0, S] \tag{9}$$

$$\implies \qquad \left\| \frac{d\boldsymbol{C}}{ds}(s) \right\|_2 = 1 \qquad \forall s \in [0, S], \tag{10}$$

where $S \in \mathbb{R}$ is the length of the track. If the vehicle starts at $\boldsymbol{C}(0)$ and travels exactly along the center-line then its position is $\boldsymbol{p} = \boldsymbol{C}(s)$ for some s. And, because of the arc length parameterization of $\boldsymbol{C}$, s is the distance travelled along the track. This quantity is important because we want to maximize it (see section II-F). The assumption that the vehicle travels exactly along the center-line is problematic. We want to evaluate the vehicle's progress along the track, even when the vehicle is close to, but not exactly on the center-line. To do this, we introduce the *progress function* $s : T \to [0, S]$:

$$s(\boldsymbol{p}) = \underset{u\in[0,S]}{\operatorname{argmin}} \|\boldsymbol{C}(u) - \boldsymbol{p}\|_2. \tag{11}$$

The progress function finds the center-line point closest to the vehicle's position and returns the arc length from the starting point to that point. Fig. 6 shows the contour graph of $s(\boldsymbol{p})$ for the following example of a center-line:

$$\boldsymbol{C}(s) = \begin{bmatrix} 2\cos(\sqrt{s}) + 2\sin(\sqrt{s})\sqrt{s} \\ 2\sin(\sqrt{s}) - 2\cos(\sqrt{s})\sqrt{s} \end{bmatrix}. \tag{12}$$

$s(\boldsymbol{p})$ can be undefined at some points in $\mathbb{R}^2$ when the *argmin*

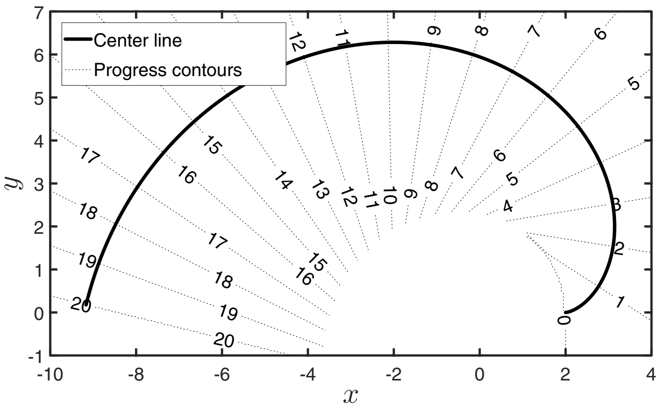

Fig. 6: Center-line and contours of the progress function

in (11) is not unique. However, the center-line and track area are chosen such that $s(\boldsymbol{p})$ is defined everywhere on T.

F. Trajectory Optimization Problem

This section formulates the trajectory optimization problem. Its goal is to define which trajectories are feasible and which are optimal. The usual goal in racing is to minimize the time to reach the end of the track. The goal of minimizing time is problematic, because the prediction (7) is non-linear with respect to Δt. We choose Δt to be constant, which makes (7) linear. With a constant Δt only a constant time period can be predicted. Thus, we change the optimization goal to maximizing the progress along the track as defined by the progress function in (11). While a precise equivalence between these optimization goals could not be demonstrated, maximizing the progress function still proves to be a useful optimization goal in practice. To adhere to the convention of formulating minimization problems, we minimize the negative progress function $-s(\boldsymbol{p}^{(H_p)})$.

The optimization problem respects the physical constraints of the vehicle and the track constraints introduced in the sections II-C, II-D and II-E. The decision variables in the optimization problem are:

$$\boldsymbol{\mathcal{X}} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(H_p)}) \tag{13}$$

$$\boldsymbol{\mathcal{U}} = (\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(H_p)}), \tag{14}$$

where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{4H_p}$ is the state prediction vector and $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{2H_p}$ is the input prediction vector. The initial state $\boldsymbol{x}^{(0)}$ is a parameter in the optimization problem. Thus, the optimization problem can be formulated as follows:

$$\underset{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{U}}}{\text{minimize}} \quad -s(\boldsymbol{p}^{(H_p)}) \tag{15}$$

$$\text{subject to } \boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)} \quad \forall i = 1, \ldots, H_p \tag{16}$$

$$\boldsymbol{p}^{(i)} \in T \quad \forall i = 1, \ldots, H_p \tag{17}$$

$$c_{acc}(\boldsymbol{v}^{(i)}, \boldsymbol{u}^{(i)}) \leq 0 \quad \forall i = 1, \ldots, H_p \tag{18}$$

$$\boldsymbol{v}^{(H_p)} = \boldsymbol{0} \tag{19}$$

The equations (15), (16), (17) and (18) were introduced in the sections II-E, II-D, II-E and II-C respectively. The terminal constraint in (19) is added to the optimization problem to handle the problem of a finite prediction horizon. Without it the next track constraint $\boldsymbol{p}^{(H_p+1)} \in T$ may become infeasible for all possible inputs $\boldsymbol{u}^{(H_p+1)}$. It ensures that the vehicle can always come to a standstill within the prediction horizon.

1) Existence and Uniqueness of Solutions: In general, the optimization problem (15-19) does not have a solution. This can be demonstrated with a simple example: If the initial speed $\|\boldsymbol{v}^{(0)}\|$ is sufficiently large and the accelerations $\|\boldsymbol{u}^{(i)}\|$ and positions $\|\boldsymbol{p}^{(i)}\|$ have some upper bounds imposed by the (18) and (17) then the problem has no solution, because the vehicle is too fast to remain on the track with the available acceleration.

No proofs or counterexamples for the uniqueness of solutions for the optimization problem (15-19) were found.

2) Approximate Solutions: The optimization problem (15-19) is in general non-convex. For example, (17) can create a non-convex constraint. As Fig. 7 illustrates, the connecting line between two points on the race track may not be part of the track.

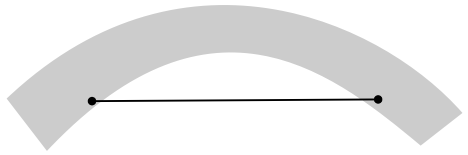

Fig. 7: Non-convexity at the inside of a turn.

A method to solve the optimization problem (15-19) reliably in real time[1] is not known to us. The next section presents a method that finds an approximate solution to problem (15-19).

III. Sequential Linearization

This section presents the method termed Sequential Linearization (SL). Its goal is to quickly find an approximate solution to the original optimization problem (15-19). To achieve a low computation time, the mathematical form of a QP is utilized. The original optimization problem is not a QP. To obtain a QP, the non-linear parts of the original optimization problem are linearly approximated based on an initial guessed solution. The resulting QP is then solved and its solution is used to create a new QP approximation. This process is repeated several times to iteratively improve the initial guessed solution. Algorithm 1 shows the pseudocode for the SL method. This iterative scheme is based on the concept of sequential convex programming [7].

Algorithm 1 Pseudocode for the SL method

```
1: $\boldsymbol{\mathcal{X}} \leftarrow [0\ 0\ \ldots\ 0]$
2: repeat forever
3:     $\boldsymbol{x}^{(0)} \leftarrow$ receiveMeasurements()
4:     for $i = 1$ to $N$ do
5:         QP $\leftarrow$ QP-Approximation($\boldsymbol{\mathcal{X}}, \boldsymbol{x}^{(0)}$)
6:         ($\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{U}}, \boldsymbol{u}^{(1)}$) $\leftarrow$ QP-Solver(QP)
7:     end for
8:     sendReferenceAcceleration($\boldsymbol{u}^{(1)}$)
9: end
```

Line 1 of Algorithm 1 initializes the first guessed solution with zeros. This assumes that the vehicle starts at the origin of the coordinate system and at a standstill. The loop that starts in line 2 runs concurrently while the vehicle drives. In line 3, the algorithm receives the vehicle's measured position and velocity. In lines 4-7, the QP approximation is formed and solved N times. While in theory one may want to run this inner loop until some convergence criterion is fulfilled, in practice a small[2], constant number of iterations is sufficient to obtain a good solution. A constant number of iterations

[1]In our case a response time of roughly 50ms – 200ms is required.
[2]In the tested scenarios $N \leq 5$ was sufficient.

is advantageous as it leads to a fairly constant computation time of the inner loop. In line 8, only the first predicted input $\boldsymbol{u}^{(1)}$ is sent to the acceleration controller. This assumes that the next iteration of the outer loop takes less time than the prediction sample time Δt. The QP approximation is always based on the latest QP solver result.

Because SL uses the previous solution as a starting point to create the next QP approximation, we need a notation to refer to the previous solution. A tilde over a variable, e.g., $\tilde{\boldsymbol{\mathcal{X}}}$, $\tilde{\boldsymbol{p}}^{(H_p)}$, marks a known numerical value from the previous solution.

The following sections describe how the progress function in (15), the track constraints in (17) and the acceleration constraints in (18) are represented and approximated in the SL method. Because the kinematic model in (16) and the final velocity constraint in (19) are linear they can be used in the QP without any changes.

A. Acceleration Constraints

The original acceleration constraints are defined as two half ellipses, see section II-C. A QP can only be subject to linear equations and inequalities. Thus a linear approximation of the acceleration constraints is required. To do this, we construct several tangents at the boundary of the original acceleration constraint, as illustrated in Fig. 8. The

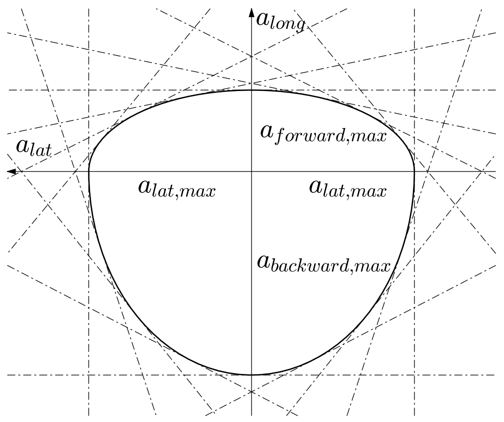

Fig. 8: Tangent approximation of acceleration constraints

tangents are distributed evenly around the boundary of the original acceleration constraint. The choice of the number of tangents n is a trade-off between the precision of the approximation and the size of the optimization problem. In the limit $n \to \infty$ the linear approximation becomes identical to the original acceleration constraints. To limit the size of the optimization problem, we choose $n = 16$ in the implementation. The area enclosed by the tangents is a convex polygon that is slightly larger than the original constraint region. In other words, the approximation is a slight relaxation of the original acceleration constraints. We consider this as an acceptable error, since it vanishes as $n \to \infty$. The approximate acceleration constraints in Fig. 8 are formulated as a set of linear inequalities as follows:

$$A_{acc}(\tilde{\boldsymbol{v}}^{(i)}, \tfrac{2\pi k}{n})\, \boldsymbol{u}^{(i)} \leq b_{acc}(\tilde{\boldsymbol{v}}^{(i)}) \quad \forall i = 1, \dots, H_p \quad \forall k = 1, \dots, n \qquad (20)$$

where $A_{acc} : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$ and $b_{acc} : \mathbb{R}^2 \to \mathbb{R}$.

B. Race Track Model

1) Linear Approximation of the Track Boundaries: Because a QP only allows linear constraints, the race track model must be approximated by a set of linear constraints. The chosen approach in the SL method is to use two linear constraints for the left and right track boundaries at each prediction time-step as illustrated in Fig. 9. The linear

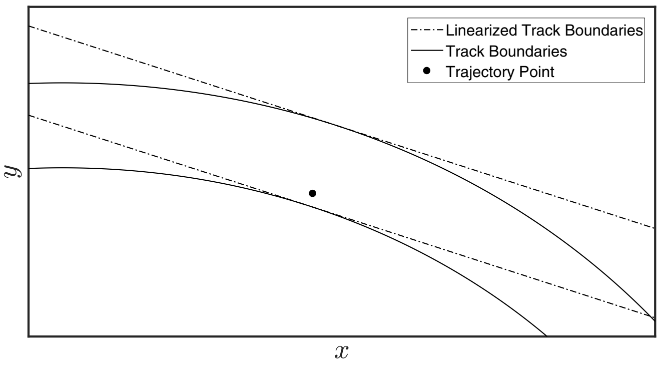

Fig. 9: Linear approximation of the track boundaries for one trajectory point

approximation of the track for the trajectory point $\boldsymbol{p}^{(i)}$ is created close to that point from the previous solution $\tilde{\boldsymbol{p}}^{(i)}$. This linear approximation of the track is good if the distance between two iterations $\|\boldsymbol{p}^{(i)} - \tilde{\boldsymbol{p}}^{(i)}\|$ is small. Section III-B.2 will introduce a trust region, which limits this distance.

To simplify the computation of the linear approximation of the track, the track center-line is represented with a set of discrete points rather than a continuous curve. The general concept of the track area T in section II-E is simplified to a track of constant width around the center-line. A varying track width would be possible, but is not explored in this paper. Fig. 10 shows the discrete points $\boldsymbol{T}_{L,j}$, $\boldsymbol{T}_{R,j}$, $\boldsymbol{T}_{C,j}$ on the left, right and center of the track and the unit vectors $\boldsymbol{f}_j$ and $\boldsymbol{n}_j$ which point forward and to the left of the center-line. The center-line is discretized on a regular grid $s_j = j\,\Delta s$ where $\Delta s \in \mathbb{R}^+$, $j \in \{1, \dots, N\}$ and $N = \lfloor \frac{S}{\Delta s} \rfloor$. The discretization shown in Fig. 10 is expressed as follows:

$$\boldsymbol{f}_j = \nabla \boldsymbol{C}(s_j) \qquad (21)$$

$$\boldsymbol{n}_j = \left[\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right] \boldsymbol{f}_j \qquad (22)$$

$$\boldsymbol{T}_{C,j} = \boldsymbol{C}(s_j) \qquad (23)$$

$$\boldsymbol{T}_{L,j} = \boldsymbol{C}(s_j) + \tfrac{w}{2}\boldsymbol{n}_j \qquad (24)$$

$$\boldsymbol{T}_{R,j} = \boldsymbol{C}(s_j) - \tfrac{w}{2}\boldsymbol{n}_j, \qquad (25)$$

where the points on the left and right boundaries of the track $\boldsymbol{T}_{L,j}$ and $\boldsymbol{T}_{R,j}$ are constructed by translating the center point $\boldsymbol{T}_{C,j}$ along the normal direction $\boldsymbol{n}_j$ of the track by half the track width $\frac{w}{2}$.

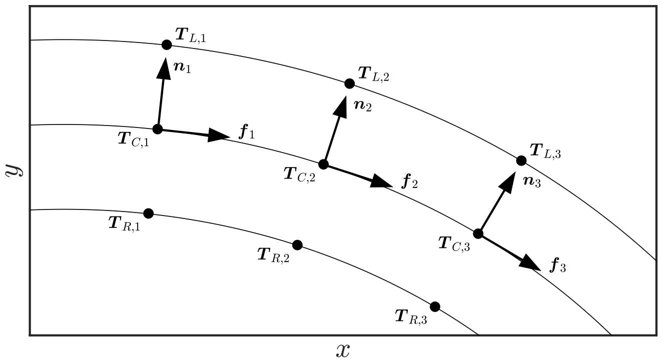

Fig. 10: Track representation in the SL method

To formulate the linear track constraints as shown in Fig. 9 the index j of the center-line point closest to $\tilde{\boldsymbol{p}}^{(i)}$ is required. This is done using the following equation:

$$j(\boldsymbol{p}) = \underset{k \in \{0,\dots,N\}}{\operatorname{argmin}} \|\boldsymbol{T}_{C,k} - \boldsymbol{p}\|_2 . \tag{26}$$

The linear track constraints are then formulated as follows:

$$(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{L,j(\tilde{\boldsymbol{p}}^{(i)})})^T(\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}) \leq 0 \quad \forall i = 1,\dots,H_p \tag{27}$$
$$(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{R,j(\tilde{\boldsymbol{p}}^{(i)})})^T(-\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}) \leq 0 \quad \forall i = 1,\dots,H_p \tag{28}$$

where the dot product is used to project the deviation from the point on the track boundary onto the outer normal vector. This projection must be negative if $\boldsymbol{p}^{(i)}$ is inside the linearized track.

The inequalities (27) and (28) could cause infeasibility of the optimization problem. This can for example happen because of a bad initialization. To alleviate this problem, the track constraints are softened using a slack variable [8]. We introduce the slack variable ξ to the optimization problem. It is constrained to be non-negative $\xi \geq 0$. The slack variable is added to the optimization objective: minimize $(\dots) + q\xi^2$. The slack weight q has a positive value. Thus, if there are no other constraints on ξ, its solution is $\xi = 0$. If there are constraints that combine ξ with other optimization variables, there will be a trade-off between the slack penalty $q\xi^2$ and other optimization goals. The slack variable is used to soften the linear track constraints as follows:

$$(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{L,j(\tilde{\boldsymbol{p}}^{(i)})})^T(\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}) \leq \xi \quad \forall i = 1,\dots,H_p \tag{29}$$
$$(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{R,j(\tilde{\boldsymbol{p}}^{(i)})})^T(-\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}) \leq \xi \quad \forall i = 1,\dots,H_p \tag{30}$$

One can interpret this geometrically: If for example $\xi = 1$, then the linearized track is widened by one unit of distance on either side. The slack weight q can be interpreted as an incentive to the optimizer to keep the original track width if possible, but widen the track if necessary.

2) Trust Region: The linear approximation of the track boundaries can be highly inaccurate for large distances between $\tilde{\boldsymbol{p}}^{(i)}$ and $\boldsymbol{p}^{(i)}$. This can, in extreme cases, cause the SL method to fail. To mitigate this problem, we introduce a trust region constraint that limits the distance between the old and new trajectory as follows:

$$\tilde{p}_x^{(i)} - L \leq p_x^{(i)} \leq \tilde{p}_x^{(i)} + L \quad \forall i = 1,\dots,H_p \tag{31}$$
$$\tilde{p}_y^{(i)} - L \leq p_y^{(i)} \leq \tilde{p}_y^{(i)} + L \quad \forall i = 1,\dots,H_p, \tag{32}$$

where $L \in \mathbb{R}$ is the size of the trust region.

3) Progress Function: Since the progress function from (11) is non-linear, it also needs to be linearly approximated. The progress function is based on the center-line. We start by linearizing the center-line as follows:

$$\boldsymbol{C}(s) \approx \boldsymbol{C}(s_j) + \nabla\boldsymbol{C}(s_j)(s - s_j) \tag{33}$$
$$= \boldsymbol{T}_{C,j} + \boldsymbol{f}_j(s - s_j) \tag{34}$$

Using this, the *argmin* definition of the progress function can be solved explicitly which results in the following:

$$s(\boldsymbol{p}) \approx s_j + \boldsymbol{f}_j^T(\boldsymbol{p} - \boldsymbol{T}_{C,j}). \tag{35}$$

C. Final Trajectory Optimization Problem

Combining (i) the linear kinematic model in (7), (ii) the terminal constraint in (19), (iii) the acceleration constraints in (20), (iv) the track constraints in (29, 30), (v) the trust region in (31, 32), (vi) the progress function in (35), (vii) and a damping penalty on the input $\boldsymbol{u}$ yields the following optimization problem:

$$\underset{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{U}},\xi}{\text{minimize}}$$
$$-\boldsymbol{f}^T_{j(\tilde{\boldsymbol{p}}^{(H_p)})}\boldsymbol{p}^{(H_p)} + q\xi^2 + R\sum_{i=2}^{H_p}\|\boldsymbol{u}^{(i)} - \boldsymbol{u}^{(i-1)}\|_2^2 \tag{36}$$

subject to

$$\boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)} \quad \forall i{=}1,\dots,H_p \tag{37}$$
$$(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{L,j(\tilde{\boldsymbol{p}}^{(i)})})^T(\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}) \leq \xi \quad \forall i{=}1,\dots,H_p \tag{38}$$
$$(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{R,j(\tilde{\boldsymbol{p}}^{(i)})})^T(-\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}) \leq \xi \quad \forall i{=}1,\dots,H_p \tag{39}$$
$$A_{acc}(\tilde{\boldsymbol{v}}^{(i)}, \tfrac{2\pi k}{n})\boldsymbol{u}^{(i)} \leq b_{acc}(\tilde{\boldsymbol{v}}^{(i)}) \quad \forall k = 1,\dots,n \quad \forall i{=}1,\dots,H_p \tag{40}$$
$$\tilde{p}_x^{(i)} - L \leq p_x^{(i)} \leq \tilde{p}_x^{(i)} + L \quad \forall i{=}1,\dots,H_p \tag{41}$$
$$\tilde{p}_y^{(i)} - L \leq p_y^{(i)} \leq \tilde{p}_y^{(i)} + L \quad \forall i{=}1,\dots,H_p \tag{42}$$
$$\boldsymbol{v}^{(H_p)} = 0 \tag{43}$$
$$0 \leq \xi \tag{44}$$

The damping penalty $R \in \mathbb{R}$ on the input changes $\boldsymbol{u}^{(i)} - \boldsymbol{u}^{(i-1)}$ is added to the objective function to reduce aggressive input changes. The constant part of the progress function in (35) is omitted in the objective in (36) as it does not affect the solution. Finally, the equations (36-44) form a QP.

IV. Implementation

The SL method is implemented using the MATLAB[9] programming language. The function `cplexqp()` from the IBM ILOG CPLEX Optimization Studio[10] is used to solve the QPs. The acceleration controllers are implemented using Simulink.

V. Simulation

To validate the SL method, it is tested in-the-loop with the high-fidelity vehicle dynamics simulation software CarMaker[11]. CarMaker provides an integration with Simulink, through which the acceleration controller commands the simulated vehicle. The CarMaker simulation and the trajectory planning run in separate processes and communicate with each other through the User Datagram Protocol (UDP). This ensures that the simulation and trajectory planning run concurrently, and that the simulation does not wait for the trajectory planning to complete. Thus, the computation time of the trajectory planning does affect the overall performance. This is done to prove the real-time capability of the system. The CarMaker simulation also runs in real-time. All simulation tests were executed on a PC with an Intel i5-4690K processor and 16 GB of RAM running Windows 7.

CarMaker includes some example vehicles and environments. The Formula One vehicle and the Hockenheim race track, shown in Fig. 11, are selected for the evaluation. The trajectory planning assumes the vehicle to be point-like. To account for the width of the vehicle, the race track width is reduced by the vehicle width in the race track model.

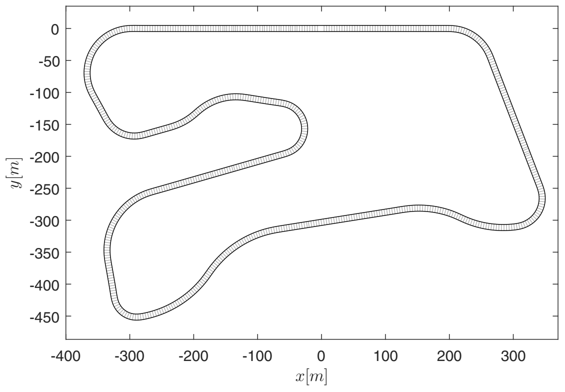

Fig. 11: CarMaker's Hockenheim example race track

CarMaker includes the software module IPGDriver, which can drive a vehicle along a race track. The acceleration curves $a_{forward,max}(v)$, $a_{backward,max}(v)$, $a_{lat,max}(v)$ described in section II-C are determined from the acceleration that the IPGDriver performs with the simulated vehicle. The IPGDriver's performance is also used as a reference for the evaluation of the SL method. Because the race track and the controlled vehicle are identical in both simulations, a direct comparison between the performance of the IPGDriver and the SL method is possible.

There is reason to believe that the IPGDriver racing performance is close to optimal. The IPGDriver has been in development since 1990. The IPGDriver first determines a reference path within the track boundaries that minimizes the curvature. It then determines a speed profile based on the vehicle's acceleration limits. These acceleration limits are speed and direction dependent and thus very similar to the acceleration limits used in this paper (see section II-C) [12]. The primary evaluation criterion is the lowest lap time T_{lap} out of five consecutive laps. Tab. II shows the parameters used for the SL method. These parameters are manually tuned to minimize lap times.

TABLE II: Parameters of the SL method

Parameter	Symbol	Value	Unit
Prediction horizon	H_p	40	–
Iterations	N	1	–
Time-step	Δt	0.15	s
Track width	w	9.4	m
Input change weight	R	0.01	s^4/m^2
Slack weight	q	10	$1/m^2$
Trust region	L	50	m

Fig. 12 compares the simulation results of the IPGDriver and the SL method. It shows data from the best lap in each simulation run. The best lap times are shown in Tab. III. Fig. 12 shows the vehicle's speed v, path curvature κ, longitudinal acceleration and lateral acceleration as functions of the traveled distance s from the start of the lap. The dynamic variables are plotted over distance rather than time to allow comparisons at the same position along the track. The paths are almost identical between all laps and simulation runs. The maximum pairwise relative variation in lap length is less than 0.5%.

TABLE III: Best lap time out of five consecutive laps

Method	T_{lap}
IPGDriver	44.64s
SL	44.45s

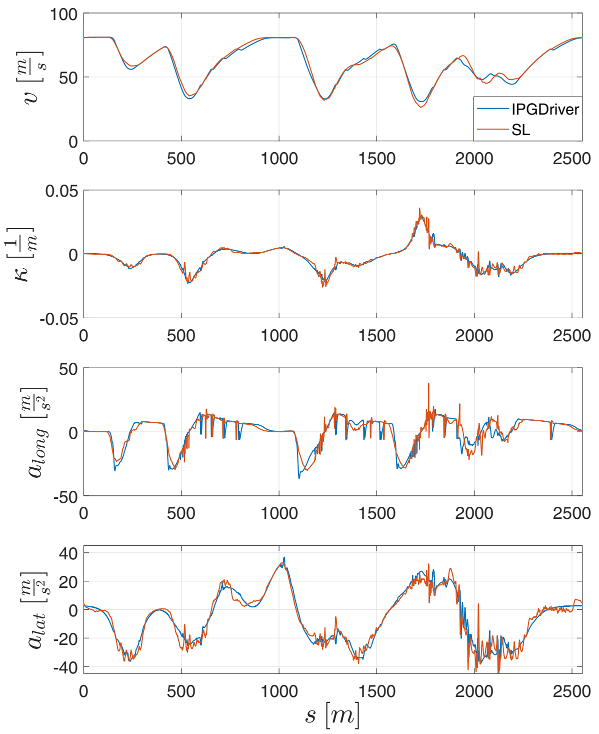

Fig. 12: Comparison of speed, path curvature, longitudinal acceleration and lateral acceleration of the IPGDriver and the SL method

The plot of the path curvature κ shows that both methods have highly similar curvature profiles, which means they agree about on which path is optimal. The lap times and the speed profiles show that the SL method can match and in some track segments exceed the performance of the IPGDriver without losing control of the vehicle.

Tab. IV shows some statistics for the computation time per time-step for the SL method. The data show that the computation time is very consistent. The computation time rarely exceeds the median computation time by 50% and never exceeds it by more than four times.

A video of the CarMaker simulation, with the SL method in control, is available [13]. Another video, which visualizes the computed trajectory of the SL method is also available [14]. In the second video a different race track and no vehicle dynamics simulation is used.

TABLE IV: Computation time statistics for the SL method

Median	99-percentile	Maximum
39.9ms	58.9ms	158.2ms

VI. Conclusion

This paper formulated the trajectory planning problem for autonomous vehicle racing using ideas from model predictive control. The proposed method of Sequential Linearization (SL) finds approximate solutions to the trajectory planning problem in real-time. Using a vehicle dynamics simulation, it was demonstrated that the SL method can achieve a competitive lap time when compared to the IPGDriver method.

Some aspects of this work are open for improvements and future research. This paper neglected some aspects of vehicle racing. For example the three-dimensional shape of the track, changes in track surface, changes in tire condition, changes in fuel mass and moving obstacles such as other vehicles were ignored. These aspects need to be considered to achieve the best performance.

References

[1] J. Canny, B. R. Donald, J. Reif, and P. G. Xavier, "On the complexity of kinodynamic planning," Cornell University, Tech. Rep., 1988.

[2] C. A. Floudas and P. M. Pardalos, *State of the art in global optimization: computational methods and applications*. Springer Science & Business Media, 2013, vol. 7.

[3] Nicholas Gould and Philippe Toint, "A quadratic programming bibliography," *Numerical Analysis Group Internal Report*, vol. 1, p. 32, 2000.

[4] N. Gould and P. Toint, "A quadratic programming page," http://www.numerical.rl.ac.uk/people/nimg/qp/qp.html, accessed: 2017-08-08.

[5] H. Mittelmann, "Decision tree for optimization software," http://plato.asu.edu/sub/nlores.html#QP-problem, accessed: 2017-08-08.

[6] B. Heißing, M. Ersoy, and S. Gies, *Fahrwerkhandbuch: Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik, Perspektiven*, ser. ATZ/MTZ-Fachbuch. Vieweg+Teubner Verlag, 2011.

[7] S. Boyd, "Sequential Convex Programming," Stanford University, Tech. Rep., 2015.

[8] J. M. Maciejowski, *Predictive control: with constraints*. Pearson education, 2002.

[9] MathWorks, "MATLAB Version 9.0 Release 2016a," 2016.

[10] IBM, "IBM ILOG CPLEX Optimization Studio Getting Started with CPLEX for MATLAB manual. Version 12 Release 6," 2015.

[11] IPG Automotive GmbH, "CarMaker User's guide Version 5.1.2," 2016.

[12] IPG Automotive GmbH, "IPGDriver Reference Manual Version 6.5," 2016.

[13] J. Maczijewski. (2017) Real-time Trajectory Optimization for Autonomous Vehicle Racing, CarMaker Test. [Online]. Available: https://youtu.be/_BxhYhIbORk

[14] J. Maczijewski. (2017) Real-time Trajectory Optimization for Autonomous Vehicle Racing. [Online]. Available: https://youtu.be/t4tkZA8yZkg