Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Bachelor Thesis

Bachelorarbeit

Real-time Trajectory Optimization for Autonomous Vehicle Racing

Echtzeitfähige Trajektorienoptimierung für autonome Automobilrennen

Janis Maczijewski
Matrikelnummer: 335828

Aachen, December 5, 2017

Examiners:
Professor Dr.-Ing. Stefan Kowalewski

Advisors:
Dr.-Ing. Bassam Alrifaee

This thesis was submitted to
Lehrstuhl Informatik 11 – Embedded Software

Communicated by Prof. Dr.-Ing. Stefan Kowalewski

Eidesstattliche Versicherung

Maczijewski, Janis	335828
Name, Vorname	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Bachelorarbeit mit dem Titel

Echtzeitfähige Trajektorienoptimierung
für autonome Automobilrennen

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt.

Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

Contents

1 Introduction

This thesis 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 for the near-future and 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 [3].

The trajectory planning task is formulated as an optimization problem using ideas from model predictive control. The resulting optimization problem is non-convex. Non-convex optimization problems in general are NP-hard [4].

To fulfill this challenging combination of requirements, this thesis uses the mathematical formulation of a linearly constrained convex quadratic optimization problem (QP) as building block. Much prior work – both theoretically [8] and in software implementations [7, 14] – has been done to solve QPs. As a result, highly efficient and optimized software libraries for solving QPs are available. Because of this, the methods developed in this thesis can solve the trajectory optimization problem in less than 50ms. The key contribution of the thesis lies in developing an approximation of the trajectory optimization problem that is compatible with a QP. Finally, the developed methods are tested against a high-fidelity vehicle dynamics simulation.

2 Fundamentals

This chapter will formulate the fundamentals of the trajectory optimization problem and the control system that uses its solution.

2.1 Assumptions

This thesis makes a few assumptions about the trajectory optimization task to limit the complexity of the trajectory optimization problem. The assumptions are:

- The race track is horizontal, there are no changes in elevation. This allows us to ignore the vertical vehicle dynamics.
- The vehicle side slip angle[1] is negligible, i.e. the vehicle travels forwards. This allows us to ignore the change in a vehicle's dynamic behavior as it transitions from normal driving to drifting, spinning, sliding, etc.
- There are no changes in vehicle and environmental conditions (e.g. weather, fuel mass, tire temperature, road surface properties).
- At all times the vehicle position, velocity, acceleration and yaw angle are known through sensors and state estimation.

Generalizing this work to remove some assumptions is open for future research.

[1] The angle between the direction of travel and the vehicle's forward direction.

2.2 Coordinate Systems and Symbols

Fig. 2.1 illustrates the coordinate systems and acceleration variables used throughout this thesis.

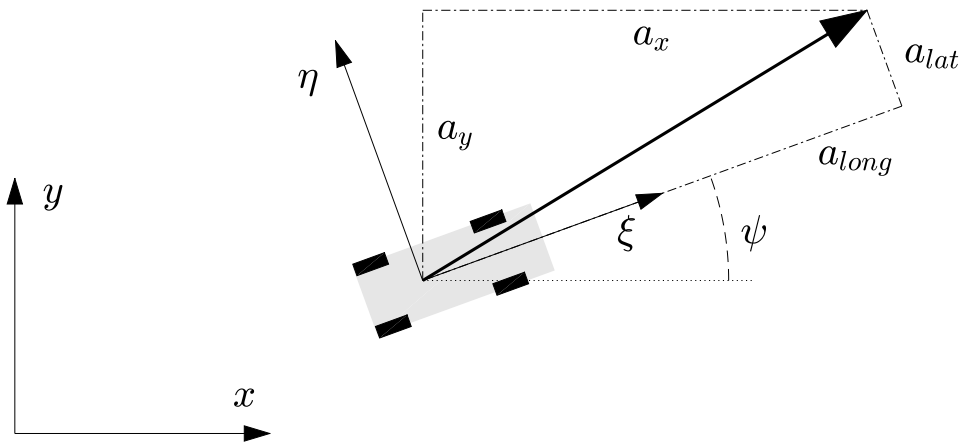

Figure 2.1: Coordinate systems

The following variables are used throughout this thesis:

x, y	Global coordinate system, stationary with respect to the race track
ξ, η	Vehicle coordinate system, with the origin at the center of gravity
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

2.3 System Overview

Fig. 2.2 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. Because a vehicle can not be directly controlled by an acceleration command, a control-loop is required that 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.

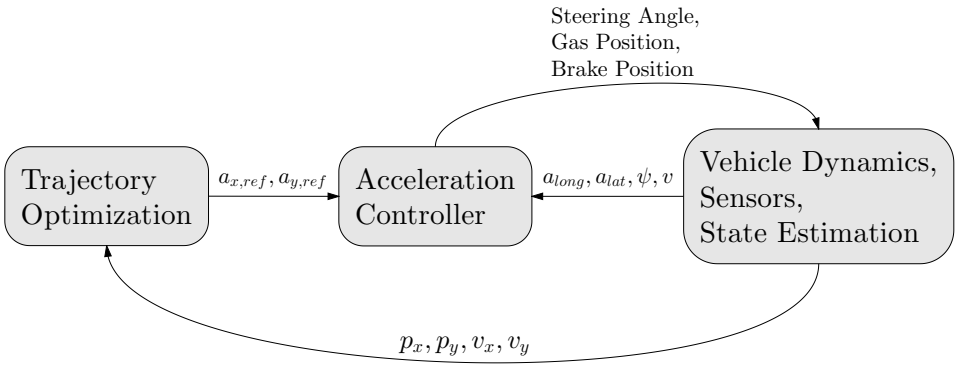

Figure 2.2: System signal flow graph

2.4 Acceleration Controller

Fig. 2.3 illustrates the acceleration controller which consists of three components: The coordinate transformation and the longitudinal and lateral acceleration controllers. While the acceleration controller is not the core subject of this thesis, it is necessary for testing the trajectory optimization in a simulation (see chapter 5).

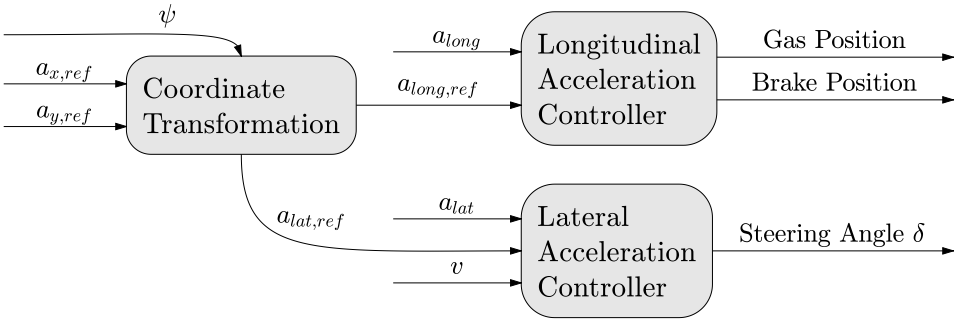

Figure 2.3: Acceleration controller signal flow graph

2.4.1 Coordinate Transformation

In the coordinate transformation the acceleration vector is rotated by the vehicle's yaw angle as illustrated in Fig. 2.1 using the formula:

$$\begin{bmatrix} a_{long,ref} \\ a_{lat,ref} \end{bmatrix} = \begin{bmatrix} cos(\psi) & sin(\psi) \\ -sin(\psi) & cos(\psi) \end{bmatrix} \begin{bmatrix} a_{x,ref} \\ a_{y,ref} \end{bmatrix}. \tag{2.1}$$

2.4.2 Longitudinal Acceleration Controller

Fig. 2.4 shows the Simulink[13] diagram of the longitudinal acceleration controller. A PI controller determines a combined gas and brake signal based on the error in the longitudinal acceleration $a_{long,ref} - a_{long}$. The controller output is split into its positive and negative parts using the functions $max(u, 0)$ and $min(u, 0)$. The positive part is used for the gas position and the negative part is used for the brake position. As a result, the gas and brake are never used concurrently. The negative part is multiplied with the brake gain which has a negative value. This results in a positive brake position signal.

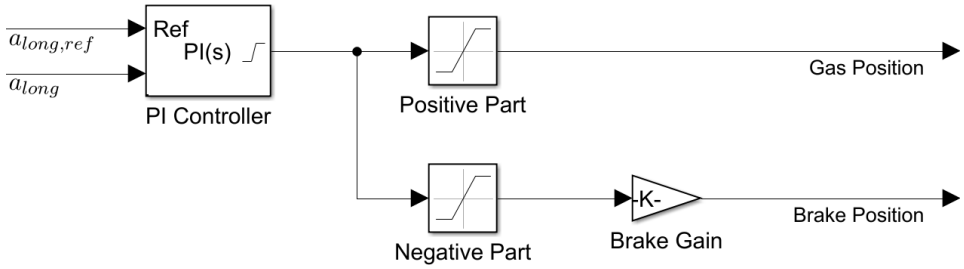

Figure 2.4: Longitudinal Acceleration Controller

2.4.3 Lateral Acceleration Controller

Fig. 2.5 shows the Simulink diagram of the lateral acceleration controller. A vehicle standing still can not create lateral accelerations. To avoid aggressive steering

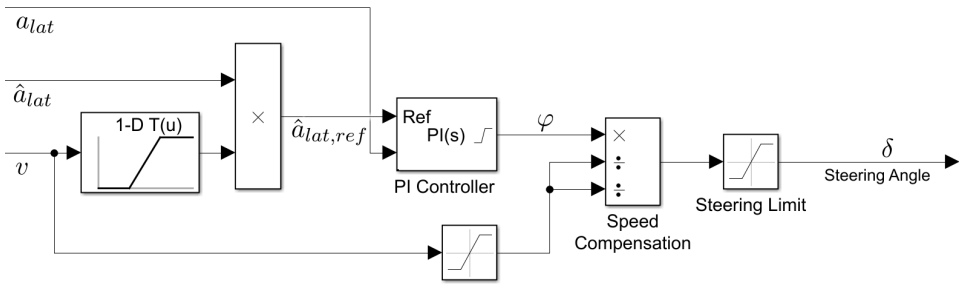

Figure 2.5: Lateral Acceleration Controller

oscillations when starting from a standstill, the reference acceleration is reduced for low speeds as follows:

$$\hat{a}_{lat,ref} = \min(1, \tfrac{v}{v_0})\ a_{lat,ref}, \tag{2.2}$$

where v is the vehicle speed, v_0 is a small[2] speed constant and $\hat{a}_{lat,ref}$ is the reduced lateral reference acceleration.

For the next part, consider the equation

$$a_{lat} = \frac{v^2 \tan(\delta)}{L}, \tag{2.3}$$

where v is the vehicle speed, δ is the steering angle in radians and L is the wheelbase. This equation applies to a bicycle model with front wheel steering [16]. It shows that the response of the lateral acceleration a_{lat} to changes in the steering angle δ depends strongly on the current speed v. To create a controller that can operate at varying speeds an adaptive control approach is used. The variable φ, defined as

$$\varphi = \delta \max(v, v_0)^2, \tag{2.4}$$

[2]Below the lowest speed in the race, excluding the start. Here $v_0 = 10\frac{m}{s}$.

is used as the input variable instead of the steering angle δ. Assuming the small angle approximation $\tan(\delta) \approx \delta$ and $v > v_0$, the lateral acceleration a_{lat} is roughly proportional to φ:

$$a_{lat} \approx \frac{\varphi}{L}. \tag{2.5}$$

The saturation with v_0 is used to avoid division by zero when calculating the steering angle $\delta = \frac{\varphi}{\max(v,v_0)^2}$. Finally, a PI controller is used to determine φ based on the acceleration error $\hat{a}_{lat,ref} - a_{lat}$.

2.4.4 Controller Parameters

The choice of controller parameters depends on the controlled vehicle. For the evaluation in this thesis we use a vehicle dynamics simulation software with a particular vehicle model. Chapter 5 describes the simulation setup. The vehicle dynamics are analyzed using step responses. The lateral acceleration error is recorded for step changes in the steering variable φ. The longitudinal acceleration error is recorded for step changes in the combined gas and brake signal. The acceleration responses for the lateral and longitudinal acceleration are modeled using system identification. The controller parameters are manually tuned using MATLAB's Control System Designer [12]. The transfer functions for the used vehicle are:

$$G_{long}(s) = (0.01\,\mathrm{s^2/m}) + (1.0\,\mathrm{s^3/m})\ \tfrac{1}{s}, \tag{2.6}$$

$$G_{lat}(s) = (20\,\mathrm{rad \cdot m}) + (2000\,\mathrm{rad \cdot m \cdot s})\ \tfrac{1}{s}. \tag{2.7}$$

2.5 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. Thus the maximum acceleration magnitude depends on the speed and the acceleration direction. Any other variables that may affect the maximum acceleration magnitude are assumed 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. 2.6.

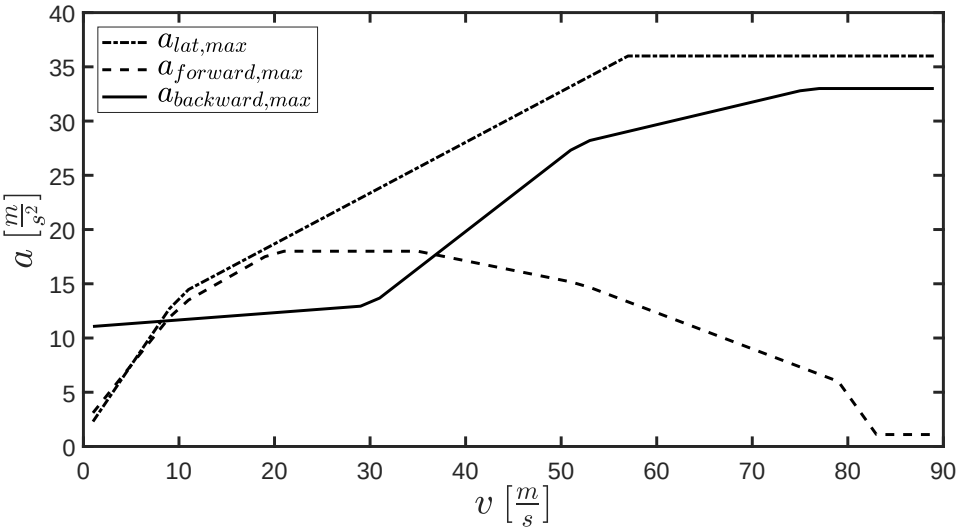

Figure 2.6: Maximum accelerations for varying speeds

To determine the maximum acceleration for arbitrary directions, two half-ellipses are used as illustrated in Fig. 2.7. This is motivated by existing models where the maximum horizontal forces that can be transferred by a tire form an ellipse [9, p. 65, 391]. The lengths of the semi axes for the forward and backward directions are 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:

$$c(a_{long}, a_{lat}, v) = \left(\frac{a_{long}}{a_{long,max}(v)}\right)^2 + \left(\frac{a_{lat}}{a_{lat,max}(v)}\right)^2 - 1 \leq 0 \tag{2.8}$$

$$a_{long,max}(v) = \begin{cases} a_{forward,max}(v) & \text{if } a_{long} > 0 \\ a_{backward,max}(v) & \text{if } a_{long} \leq 0 \end{cases} \tag{2.9}$$

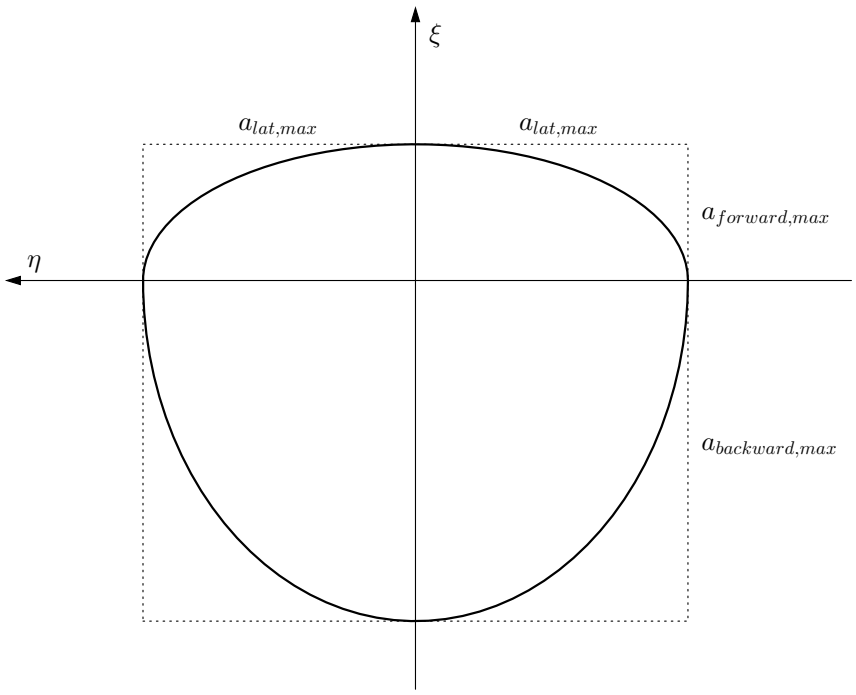

Figure 2.7: Two half-ellipses model the maximum accelerations for varying acceleration directions

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:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \sqrt{v_x^2 + v_y^2} \begin{bmatrix} \cos(\psi) \\ \sin(\psi) \end{bmatrix}. \tag{2.10}$$

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

$$\begin{bmatrix} a_{long} \\ a_{lat} \end{bmatrix} = \frac{1}{\sqrt{v_x^2 + v_y^2 + \varepsilon}} \begin{bmatrix} v_x & v_y \\ -v_y & v_x \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}. \tag{2.11}$$

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 equation (2.11) in equation (2.8) yields the final acceleration constraint

$$c_{acc}(\boldsymbol{v}, \boldsymbol{u}) = \frac{1}{v_y^2 + v_y^2 + \varepsilon} \left(\left(\frac{a_x v_x + a_y v_y}{a_{long,max} \left(\sqrt{v_x^2 + v_y^2} \right)} \right)^2 + \left(\frac{a_y v_x - a_x v_y}{a_{lat,max} \left(\sqrt{v_x^2 + v_y^2} \right)} \right)^2 \right) - 1 \leq 0. \tag{2.12}$$

2.6 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:

$$\dot{p}_x = v_x, \quad \dot{p}_y = v_y, \quad \dot{v}_x = a_x, \quad \dot{v}_y = a_y, \tag{2.13}$$

where all variables are defined in section 2.2. The equations (2.13) 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:

$$\underbrace{\begin{bmatrix} p_x^{(i)} \\ p_y^{(i)} \\ v_x^{(i)} \\ v_y^{(i)} \end{bmatrix}}_{=\boldsymbol{x}^{(i)}} = \underbrace{\begin{bmatrix} 1 & & \Delta t & \\ & 1 & & \Delta t \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{=A} \underbrace{\begin{bmatrix} p_x^{(i-1)} \\ p_y^{(i-1)} \\ v_x^{(i-1)} \\ v_y^{(i-1)} \end{bmatrix}}_{=\boldsymbol{x}^{(i-1)}} + \underbrace{\begin{bmatrix} \frac{\Delta t^2}{2} & \\ & \frac{\Delta t^2}{2} \\ \Delta t & \\ & \Delta t \end{bmatrix}}_{=B} \underbrace{\begin{bmatrix} a_x^{(i)} \\ a_y^{(i)} \end{bmatrix}}_{=\boldsymbol{u}^{(i)}}, \tag{2.14}$$

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. In this thesis a superscript in parentheses – for example i in $\boldsymbol{x}^{(i)}$ – denotes a temporal index while an ordinary superscript – for example 3 in A^3 – still denotes exponentiation. Fig. 2.8 illustrates the index convention for the state and input vectors on a time-line.

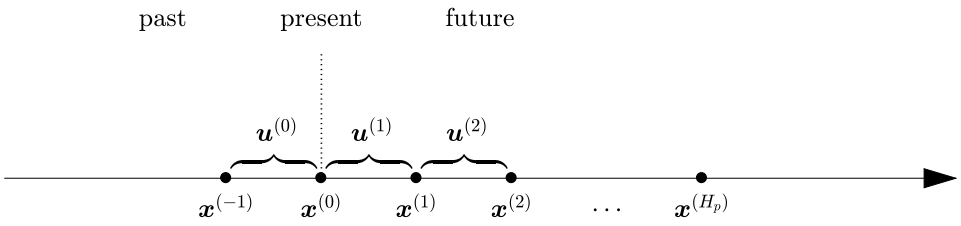

Figure 2.8: MPC time-line and indices

Each state vector applies to a particular point in time. Each input vector applies to the time span between two states. The relation between the continuous and discrete variables is as follows:

$$\hat{\boldsymbol{x}}(t_0 + i\Delta t) = \boldsymbol{x}^{(i)} \tag{2.15}$$

$$\hat{\boldsymbol{u}}(t_0 + \tau) = \boldsymbol{u}^{(i)} \quad \forall \tau \in [(i-1)\Delta t,\ i\Delta t) \tag{2.16}$$

where

- $i \in \{0, 1, \ldots, H_p\}$ is the time step index,
- $H_p \in \mathbb{N}$ is the prediction horizon,
- $\hat{\boldsymbol{x}} : \mathbb{R} \to \mathbb{R}^4$ is the continuous state prediction,
- $\hat{\boldsymbol{u}} : \mathbb{R} \to \mathbb{R}^2$ is the continuous input prediction and
- $t_0 \in \mathbb{R}$ is the present time in the current MPC problem.

The discrete model is then iterated in time to create the predicted trajectory:

$$\boldsymbol{x}^{(1)} = A\boldsymbol{x}^{(0)} + B\boldsymbol{u}^{(1)} \tag{2.17}$$

$$\boldsymbol{x}^{(2)} = A\boldsymbol{x}^{(1)} + B\boldsymbol{u}^{(2)} \tag{2.18}$$

$$\boldsymbol{x}^{(3)} = A\boldsymbol{x}^{(2)} + B\boldsymbol{u}^{(3)} \tag{2.19}$$

$$\vdots$$

$$\boldsymbol{x}^{(H_p)} = A\boldsymbol{x}^{(H_p-1)} + B\boldsymbol{u}^{(H_p)}. \tag{2.20}$$

2.7 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. 2.9 illustrates the center-line and the track area.

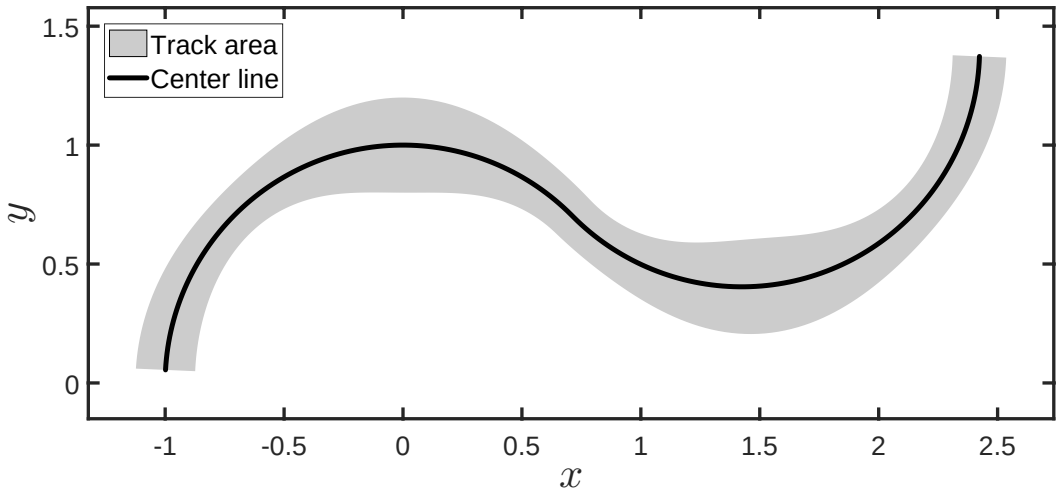

Figure 2.9: Example of a race track model

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

$$\boldsymbol{p}^{(i)} \in T \quad \forall i = 1, \ldots, H_p. \tag{2.21}$$

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

$$\int_0^s \left\| \frac{d\boldsymbol{C}}{ds}(u) \right\|_2 du = s \qquad \forall s \in [0, S] \tag{2.22}$$

$$\implies \qquad \left\| \frac{d\boldsymbol{C}}{ds}(s) \right\|_2 = 1 \qquad \forall s \in [0, S], \tag{2.23}$$

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 2.8). 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* $s : T \to [0, S]$:

$$s(\boldsymbol{p}) = \underset{u \in [0,S]}{\operatorname{argmin}} \|\boldsymbol{C}(u) - \boldsymbol{p}\|_2. \tag{2.24}$$

The progress function finds the center-line point closest to the vehicle's position and returns the arc length from the start point to that point. Fig. 2.10 shows the contour graph of $s(\boldsymbol{p})$ for the following example of a center-line:

$$\boldsymbol{C}(s) = \begin{bmatrix} 2\cos(\sqrt{s}) + 2\sin(\sqrt{s})\sqrt{s} \\ 2\sin(\sqrt{s}) - 2\cos(\sqrt{s})\sqrt{s} \end{bmatrix}. \tag{2.25}$$

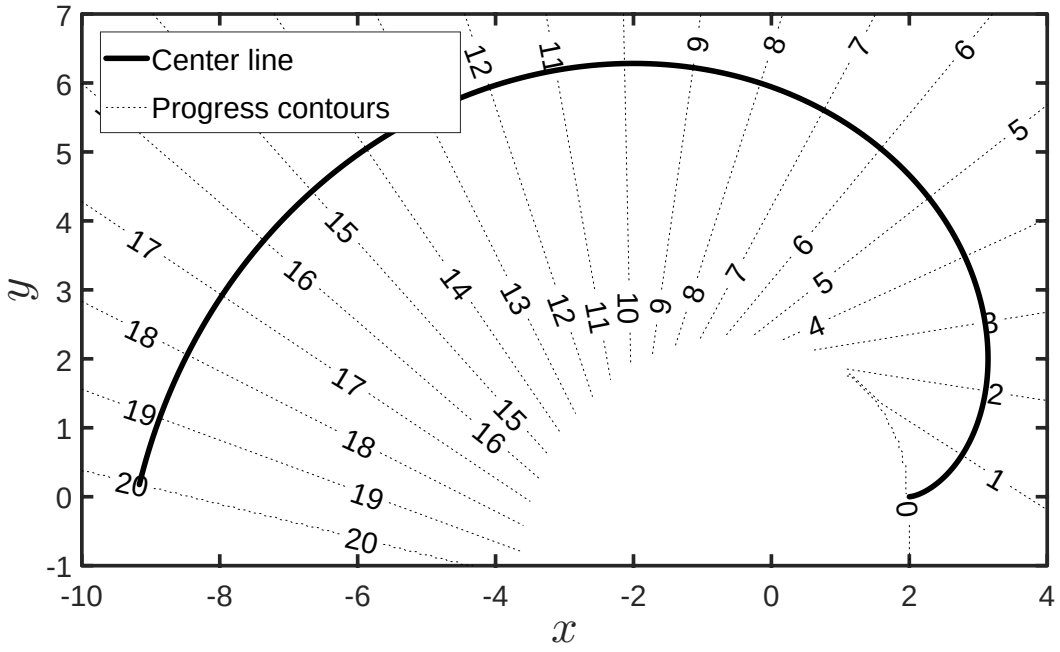

Figure 2.10: Center-line and contours of the progress function

$s(\boldsymbol{p})$ can be undefined at some points in $\mathbb{R}^2$ when the *argmin* in equation (2.24) is not unique. However, the center-line and track area are chosen such that $s(\boldsymbol{p})$ is defined everywhere on T.

This formulation of the progress function is problematic for closed loop race tracks. Because then $\boldsymbol{C}(0) = \boldsymbol{C}(S)$ and the progress function is undefined at that point. One approach is to only consider the current section of the track. But this is not necessary because of further simplifications in the actual implementation.

2.8 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 equation (2.14) is non-linear with respect to Δt. This thesis chooses Δt to be constant, which makes equation (2.14) linear. Choosing Δt as an optimization variable is open for future research. 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 equation (2.24). Section (2.8.2) will discuss and compare these two optimization goals.

The optimization problem respects the physical constraints of the vehicle and the track constraints introduced in the sections 2.5, 2.6 and 2.7. The decision variables in the optimization problem are:

$$\boldsymbol{\mathcal{X}} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ \ldots, \ \boldsymbol{x}^{(H_p)}) \tag{2.26}$$

$$\boldsymbol{\mathcal{U}} = (\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ \ldots, \ \boldsymbol{u}^{(H_p)}), \tag{2.27}$$

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:

$$\underset{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{U}}}{\text{maximize}} \quad s(\boldsymbol{p}^{(H_p)}) \tag{2.28}$$

$$\text{subject to} \quad \boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)} \qquad \forall i = 1, \ldots, H_p \tag{2.29}$$

$$\boldsymbol{p}^{(i)} \in T \qquad \forall i = 1, \ldots, H_p \tag{2.30}$$

$$c_{acc}(\boldsymbol{v}^{(i)}, \boldsymbol{u}^{(i)}) \leq 0 \qquad \forall i = 1, \ldots, H_p \tag{2.31}$$

$$\boldsymbol{v}^{(H_p)} = \mathbf{0} \tag{2.32}$$

The equations (2.28), (2.29), (2.30) and (2.31) were introduced in the sections 2.7, 2.6, 2.7 and 2.5 respectively. The constraint in equation (2.32) 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.

2.8.1 Existence and Uniqueness of Solutions

In general, the optimization problem (2.28-2.32) 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 bound imposed by the equations (2.31) and (2.30) then the problem has no solution, because the vehicle is too fast to remain on the track with the available acceleration. Guaranteeing feasible but possibly sub-optimal solutions is the subject of chapter 4. No proofs or counterexamples for the uniqueness of solutions for the optimization problem (2.28-2.32) were found.

2.8.2 Optimization Goal

The optimization problem (2.28-2.32) maximizes the progress function instead of minimizing the time to reach the end of the track. This raises the question whether these two optimization goals are equivalent. Equivalence in this case means that both formulations of the optimization problem lead to the same solution. This equivalence can be demonstrated for a simplified, one-dimensional example. The simplifications are:

- All variables related to the y axis are zero (e.g. $p_y^{(2)} = 0$) and will be omitted.
- $H_p = 2$.
- $p_x^{(0)} = 0$.
- $v_x^{(0)} = 0$.
- $\left|a_x^{(i)}\right| \leq 1,\ i = 1, 2$; this replaces the acceleration constraint (2.31).
- $\boldsymbol{C}(s) = [s \ \ 0], s \in [0, 2]$ which implies $s(\boldsymbol{p}) = p_x$.

The two variants of the optimization problem are:

Problem A	Problem B
minimize Δt	maximize $p_x^{(2)}$
subject to	subject to
$p_x^{(0)} = 0$	$p_x^{(0)} = 0$
$v_x^{(0)} = 0$	$v_x^{(0)} = 0$
$v_x^{(2)} = 0$	$v_x^{(2)} = 0$
$\left\|a_x^{(i)}\right\| \leq 1,\ i = 1, 2$	$\left\|a_x^{(i)}\right\| \leq 1,\ i = 1, 2$
$\boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)},\ i = 1, 2$	$\boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)},\ i = 1, 2$
$p_x^{(2)} = 4$	$\Delta t = 2$
$\Delta t > 0$	

These problems are solved manually. The solution to problem A includes $\Delta t = 2$, the solution to problem B includes $p_x^{(2)} = 4$. The solutions to both problems satisfy the constraints of the other. Thus, they are equivalent.

A proof that this equivalence extends to the optimization problem (2.28-2.32) in general was not found. Even if this equivalence does not hold precisely in the general case, maximizing the progress function still proves to be a useful optimization goal in practice.

2.8.3 Approximate Solutions

The optimization problem (2.28-2.32) is in general non-convex. For example equation (2.30) can create a non-convex constraint. As Fig. 2.11 illustrates, the connecting line between two points on the race track may not be part of the track.

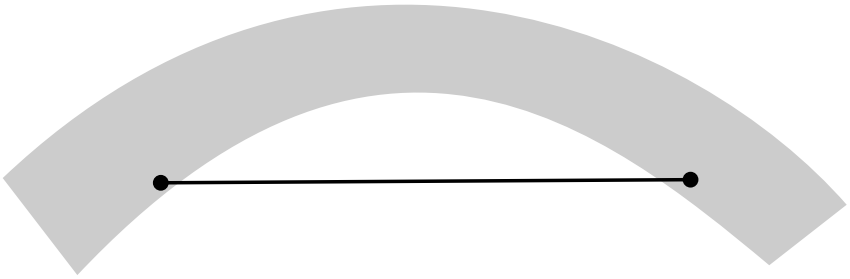

Figure 2.11: Non-convexity at the inside of a turn.

A method to solve the optimization problem (2.28-2.32) reliably in real time[3] is not known to us. The next two chapters will present two methods that find approximate solutions to problem (2.28-2.32). The core idea in both chapters is to:

1. guess a solution,

2. create a local approximation of problem (2.28-2.32) in the form of a convex quadratic program (QP) using the guessed solution,

3. solve this approximation with established methods,

4. repeat the steps 2 and 3 using the solution in step 3 as the new guessed solution in step 2.

This iterative scheme is based on the concept of Sequential Convex Programming [1]. Much prior work – both theoretically [8] and in software implementations [7, 14] – has been done to solve QPs. As a result, highly efficient and optimized software libraries for solving QPs are available.

[3] In our case a response time of roughly 50ms – 500ms is required.

3 Trajectory Optimization using Sequential Linearization

This chapter presents the method termed Sequential Linearization (SL). Its goal is to quickly find an approximate solution to the trajectory optimization problem (2.28-2.32), form here on referred to as the *original program*. To achieve a low computation time, the mathematical form of a convex quadratic program (QP) is utilized. The original program is not a QP. To obtain a QP, the non-linear parts of the original program 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.

```
1: 𝒳 ← [0 0 ... 0]
2: repeat forever
3:         x^(0) ← receiveMeasurements()
4:         for i = 1 to N do
5:                 QP ← QP-Approximation(𝒳, x^(0))
6:                 (𝒳, 𝒰, u^(1)) ← QP-Solver(QP)
7:         end for
8:         sendReferenceAcceleration(u^(1))
9: end
```

Algorithm 1: Pseudocode for the Sequential Linearization Method

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[1], constant number of iterations is sufficient to obtain a good solution. A constant number of iterations is advantageous as it leads to a fairly constant computation

[1] In the tested scenarios $N \leq 5$ was sufficient.

time of the outer 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.

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 equation (2.28), the track constraints in equation (2.30) and the acceleration constraints in equation (2.31) are represented and approximated in the SL method. Because the kinematic model in equation (2.29) and the final velocity constraint in equation (2.32) are linear they can be used in the QP without any changes.

3.1 Acceleration Constraints

The original acceleration constraints are defined as two half ellipses, see section 2.5. A QP can only be subject to linear equations and inequalities. Thus a linear approximation of the acceleration constraints is required. Several tangents are constructed at the boundary of the original acceleration constraint, as illustrated in Fig. 3.1.

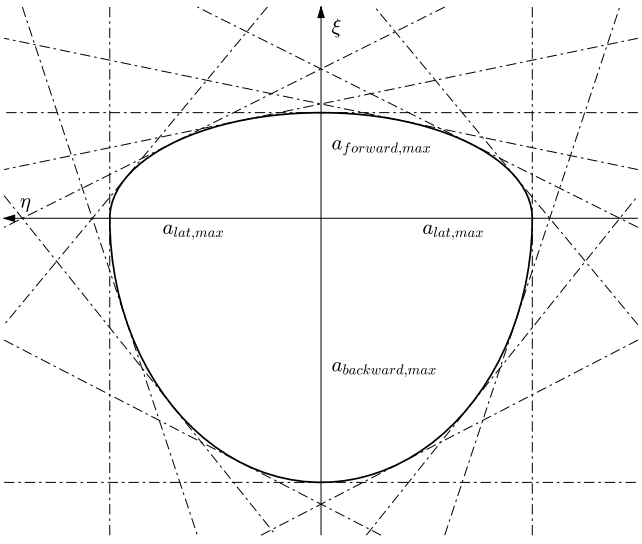

Figure 3.1: Tangent approximation of acceleration constraints

The 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 see this as an acceptable error, since it vanishes as $n \to \infty$. Each tangent is constructed starting from some point on the boundary of the original acceleration constraint. Such a point can be written as

$$\begin{bmatrix} \hat{a}_{long} \\ \hat{a}_{lat} \end{bmatrix} = \begin{bmatrix} a_{long,max}(\tilde{v})\ cos(\phi) \\ a_{lat,max}(\tilde{v})\ sin(\phi) \end{bmatrix} \tag{3.1}$$

$$\text{where} \quad a_{long,max}(\tilde{v}) = \begin{cases} a_{forward,max}(\tilde{v}) & \text{if } cos(\phi) > 0 \\ a_{backward,max}(\tilde{v}) & \text{if } cos(\phi) \leq 0. \end{cases} \tag{3.2}$$

That equation (3.1) actually describes a point on the boundary of the original acceleration constraint can be verified by substituting equation (3.1) into equation (2.8) and using the identity $\cos^2(\phi) + \sin^2(\phi) = 1$. The angle ϕ comes from a regular grid $\phi = \frac{2\pi k}{n}$, $\forall k = 1, \ldots, n$. The direction vector of each tangent is obtained through differentiation as follows:

$$\boldsymbol{t} = \frac{\partial}{\partial \phi} \begin{bmatrix} \hat{a}_{long} \\ \hat{a}_{lat} \end{bmatrix} = \begin{bmatrix} a_{long,max}(\tilde{v})\ (-sin(\phi)) \\ a_{lat,max}(\tilde{v})\ cos(\phi) \end{bmatrix} \tag{3.3}$$

To formulate the linear constraint that corresponds to each tangent, the outer normal vector which points to the infeasible region is required. Because the tangent vector points in the counter-clockwise direction along the boundary of the original acceleration constraint, the outer normal vector is obtained by rotating the tangent vector 90° clockwise as follows:

$$\boldsymbol{n} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \boldsymbol{t} = \begin{bmatrix} a_{lat,max}(\tilde{v})\ cos(\phi) \\ a_{long,max}(\tilde{v})\ sin(\phi) \end{bmatrix} \tag{3.4}$$

The linear constraint is then formulated using the dot product as follows:

$$\begin{bmatrix} a_{long} - \hat{a}_{long} \\ a_{lat} - \hat{a}_{lat} \end{bmatrix}^T \boldsymbol{n} \leq 0. \tag{3.5}$$

This dot product can be interpreted as the deviation from the point $(\hat{a}_{long}, \hat{a}_{lat})$ projected onto the outer normal vector. This quantity must be negative because the outer normal vector points to the infeasible region.

Substitution of the equations (2.11), (3.4) and (3.1) into inequality (3.5) results in the following inequality:

$$\left(\frac{1}{\sqrt{\tilde{v}_x^2 + \tilde{v}_y^2 + \varepsilon}} \begin{bmatrix} \tilde{v}_x & \tilde{v}_y \\ -\tilde{v}_y & \tilde{v}_x \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} - \begin{bmatrix} a_{long,max}(\tilde{v})\ cos(\phi) \\ a_{lat,max}(\tilde{v})\ sin(\phi) \end{bmatrix} \right)^T \begin{bmatrix} a_{lat,max}(\tilde{v})\ cos(\phi) \\ a_{long,max}(\tilde{v})\ sin(\phi) \end{bmatrix} \leq 0 \tag{3.6}$$

Simplification of inequality (3.6) leads to the final acceleration constraint:

$$\underbrace{\frac{1}{\sqrt{\tilde{v}_x^2 + \tilde{v}_y^2 + \varepsilon}} \begin{bmatrix} a_{lat,max}(\tilde{v})\ cos(\phi) \\ a_{long,max}(\tilde{v})\ sin(\phi) \end{bmatrix}^T \begin{bmatrix} \tilde{v}_x & \tilde{v}_y \\ -\tilde{v}_y & \tilde{v}_x \end{bmatrix}}_{A_{acc}(\tilde{\boldsymbol{v}}, \phi)} \boldsymbol{u} \leq \underbrace{a_{long,max}(\tilde{v})\ a_{lat,max}(\tilde{v})}_{b_{acc}(\tilde{\boldsymbol{v}})} \tag{3.7}$$

This constraint is applied to all prediction time-steps and for multiple values of ϕ as follows:

$$A_{acc}(\tilde{\boldsymbol{v}}^{(i)}, \tfrac{2\pi k}{n})\ \boldsymbol{u}^{(i)} \leq b_{acc}(\tilde{\boldsymbol{v}}^{(i)}) \quad \forall i = 1, \ldots, H_p \tag{3.8}$$

$$\forall k = 1, \ldots, n \tag{3.9}$$

3.2 Race Track Model

3.2.1 Linear Approximation of the Track Boundaries

Because a quadratic program 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. 3.2.

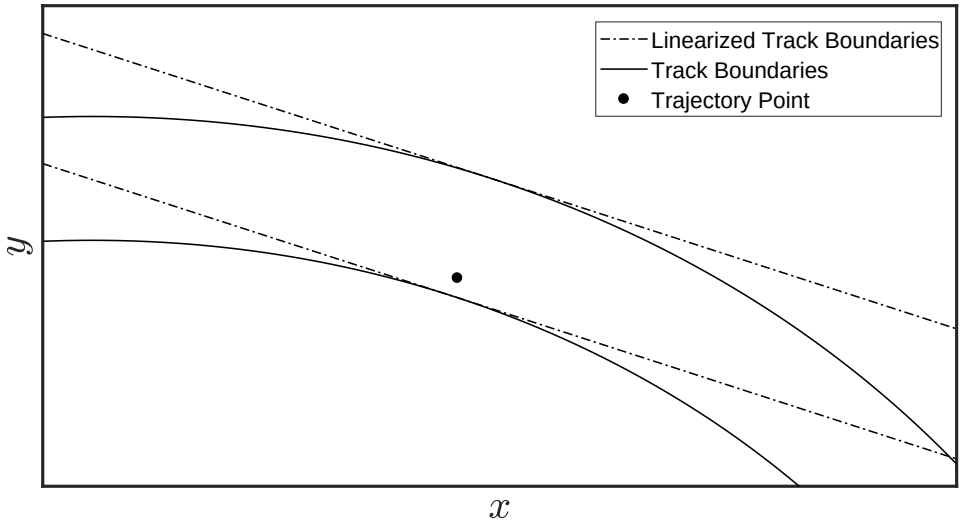

Figure 3.2: Linear approximation of the track boundaries for one trajectory point

The linear 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 poor if the solution for $\boldsymbol{p}^{(i)}$ is far away from $\tilde{\boldsymbol{p}}^{(i)}$. However, as the SL iteration approaches a stable solution, the distance $\|\boldsymbol{p}^{(i)} - \tilde{\boldsymbol{p}}^{(i)}\|$ decreases and the linear approximation becomes more accurate.

To simplify the calculation of the linear approximation of the track, the track center-line is represented with a set of discrete points rather that a continuous curve. The general concept of the track area T in section 2.7 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 thesis. Fig. 3.3 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.

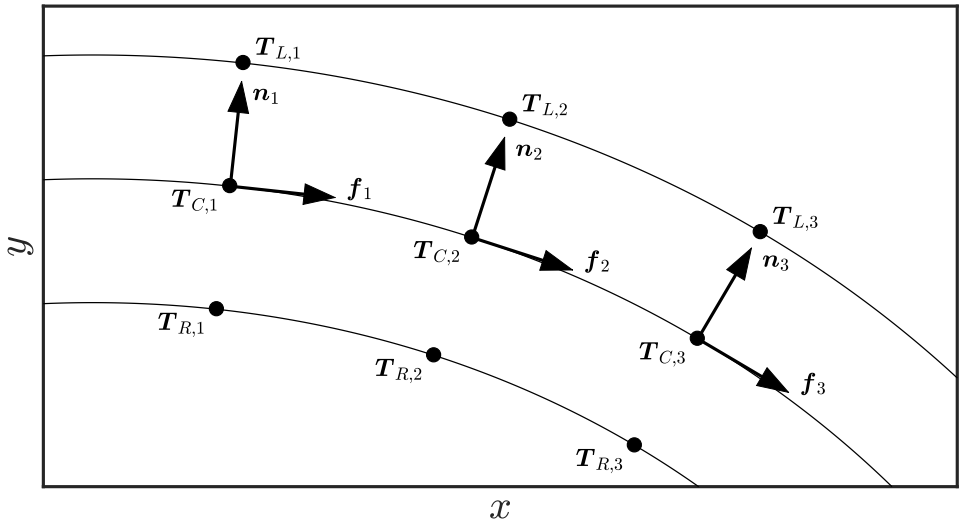

Figure 3.3: Track representation in the Sequential Linearization method

The center-line is discretized on a regular grid $s_j = j\,\Delta s$ where $\Delta s \in \mathbb{R}^+$, $j \in \{0, 1, \ldots, N\}$ and $N = \lfloor \frac{S}{\Delta s} \rfloor$. The discretization shown in Fig. 3.3 is expressed as follows:

$$\boldsymbol{f}_j = \nabla \boldsymbol{C}(s_j) \tag{3.10}$$
$$\boldsymbol{n}_j = \left[\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right] \boldsymbol{f}_j \tag{3.11}$$
$$\boldsymbol{T}_{C,j} = \boldsymbol{C}(s_j) \tag{3.12}$$
$$\boldsymbol{T}_{L,j} = \boldsymbol{C}(s_j) + \tfrac{w}{2}\boldsymbol{n}_j \tag{3.13}$$
$$\boldsymbol{T}_{R,j} = \boldsymbol{C}(s_j) - \tfrac{w}{2}\boldsymbol{n}_j \tag{3.14}$$

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

To formulate the linear track constraints as shown in Fig. 3.2 the index j of the center-line point closest to $\tilde{\boldsymbol{p}}^{(i)}$ is required. This is done using a simple brute-force approach as follows:

$$j(\boldsymbol{p}) = \underset{k\in\{0,\dots,N\}}{\operatorname{argmin}} \left\|\boldsymbol{T}_{C,k} - \boldsymbol{p}\right\|_2 . \tag{3.15}$$

The linear track constraints are then formulated as follows:

$$\left(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{L,j(\tilde{\boldsymbol{p}}^{(i)})}\right)^T \left(\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}\right) \leq 0 \quad \forall i = 1,\dots,H_p \tag{3.16}$$

$$\left(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{R,j(\tilde{\boldsymbol{p}}^{(i)})}\right)^T \left(-\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}\right) \leq 0 \quad \forall i = 1,\dots,H_p \tag{3.17}$$

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.

In some cases the linear track constraints in the inequalities (3.16) and (3.17) create a problem if they make the optimization problem infeasible. This can for example happen because of a bad initialization. To alleviate this problem, the track constraints are softened using a slack variable [11, p. 98]. 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:

$$\left(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{L,j(\tilde{\boldsymbol{p}}^{(i)})}\right)^T \left(\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}\right) \leq \xi \quad \forall i = 1,\dots,H_p \tag{3.18}$$

$$\left(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{R,j(\tilde{\boldsymbol{p}}^{(i)})}\right)^T \left(-\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}\right) \leq \xi \quad \forall i = 1,\dots,H_p \tag{3.19}$$

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.

3.2.2 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:

$$\tilde{p}_x^{(i)} - L \leq p_x^{(i)} \leq \tilde{p}_x^{(i)} + L \quad \forall i = 1, \ldots, H_p, \tag{3.20}$$

$$\tilde{p}_y^{(i)} - L \leq p_y^{(i)} \leq \tilde{p}_y^{(i)} + L \quad \forall i = 1, \ldots, H_p, \tag{3.21}$$

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

3.2.3 Progress Function

Since the progress function from equation (2.24) 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:

$$\boldsymbol{C}(s) \approx \boldsymbol{C}(s_j) + \nabla \boldsymbol{C}(s_j)(s - s_j) \tag{3.22}$$

$$= \boldsymbol{T}_{C,j} + \boldsymbol{f}_j(s - s_j) \tag{3.23}$$

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

$$s(\boldsymbol{p}) \approx s_j + \boldsymbol{f}_j^T(\boldsymbol{p} - \boldsymbol{T}_{C,j}) \tag{3.24}$$

3.3 Trajectory Optimization Problem

Combining

- the acceleration constraints in section 3.1,
- the track constraints, trust region and the progress function in section 3.2,
- the linear kinematic model in section 2.14,
- the terminal constraint in equation (2.32)
- and a damping penalty on the input $\boldsymbol{u}$

yields the following optimization problem:

$$\begin{aligned}
\underset{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{U}},\xi}{\text{minimize}} \quad & -\boldsymbol{f}_{j(\tilde{\boldsymbol{p}}^{(H_p)})}^T \boldsymbol{p}^{(H_p)} + q\xi^2 + R\sum_{i=2}^{H_p} \|\boldsymbol{u}^{(i)} - \boldsymbol{u}^{(i-1)}\|_2^2 && && (3.25)\\
\text{subject to} \quad & \boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)} && \forall i = 1,\dots,H_p && (3.26)\\
& \left(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{L,j(\tilde{\boldsymbol{p}}^{(i)})}\right)^T \left(\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}\right) \le \xi && \forall i = 1,\dots,H_p && (3.27)\\
& \left(\boldsymbol{p}^{(i)} - \boldsymbol{T}_{R,j(\tilde{\boldsymbol{p}}^{(i)})}\right)^T \left(-\boldsymbol{n}_{j(\tilde{\boldsymbol{p}}^{(i)})}\right) \le \xi && \forall i = 1,\dots,H_p && (3.28)\\
& A_{acc}(\tilde{\boldsymbol{v}}^{(i)}, \tfrac{2\pi k}{n})\boldsymbol{u}^{(i)} \le b_{acc}(\tilde{\boldsymbol{v}}^{(i)}) && \forall k = 1,\dots,n && \\
& && \forall i = 1,\dots,H_p && (3.29)\\
& \tilde{p}_x^{(i)} - L \le p_x^{(i)} \le \tilde{p}_x^{(i)} + L && \forall i = 1,\dots,H_p && (3.30)\\
& \tilde{p}_y^{(i)} - L \le p_y^{(i)} \le \tilde{p}_y^{(i)} + L && \forall i = 1,\dots,H_p && (3.31)\\
& \boldsymbol{v}^{(H_p)} = 0 && && (3.32)\\
& 0 \le \xi && && (3.33)
\end{aligned}$$

The damping penalty $R \in \mathbb{R}$ on the input changes $\boldsymbol{u}^{(i)} - \boldsymbol{u}^{(i-1)}$ is added to the minimization objective to reduce aggressive input changes. The constant part of the progress function in equation (3.24) is omitted in the objective in equation (3.25) as

it does not affect the solution. The equations (3.25-3.33) form a convex quadratic program (QP).

4 Trajectory Optimization using Sequential Convex Restriction

This chapter presents the method termed Sequential Convex Restriction (SCR) which solves the trajectory optimization problem. SCR is in many ways similar to the Sequential Linearization (SL) method presented in chapter 3. SCR is also an iterative method and uses a QP solver as a subroutine to achieve a low computation time. However, the key improvement of SCR over SL is the provided guarantee of a feasible trajectory prediction in the original program (2.28-2.32). A feasible trajectory prediction may in general not exist as discussed in section 2.8.1. Thus, a feasible trajectory prediction can only be guaranteed with a particular assumption. The SCR method assumes that a feasible trajectory prediction from the previous time-step is available. Given a feasible trajectory prediction, the SCR method can determine an improved feasible trajectory prediction for the next time-step. Algorithm 2 shows the pseudocode for the SCR method. Except for the initialization in line 4 and the QP formulation in line 6 it is identical to the SL method.

```
1:  𝓧 ← [0 0 ... 0]
2:  repeat forever
3:      x^(0) ← receiveMeasurements()
4:      𝓧 ← Initialization(𝓧)
5:      for i = 1 to N do
6:          QP ← QP-Restriction(𝓧, x^(0))
7:          (𝓧, 𝓤, u^(1)) ← QP-Solver(QP)
8:      end for
9:      sendReferenceAcceleration(u^(1))
10: end
```

Algorithm 2: Pseudocode for the SCR method

4.1 Initialization

Since the SCR method uses a known feasible trajectory prediction to determine an improved feasible trajectory prediction, we need to specify the initial feasible trajectory prediction.

In the first time-step, the vehicle is assumed to stand still at the origin. The origin is assumed to be part of the track. Thus, $\mathcal{X} = [0\ 0\ \dots\ 0], \mathcal{U} = [0\ 0\ \dots\ 0]$ is a feasible trajectory prediction. In the subsequent time-steps, the final trajectory prediction of the previous time-step – denoted with a tilde (e.g. $\tilde{\mathcal{X}}$, $\tilde{\boldsymbol{x}}^{(1)}$) – is shifted by one time-step Δt as follows:

$$\boldsymbol{x}^{(i-1)} = \tilde{\boldsymbol{x}}^{(i)} \qquad \forall i = 2, \dots, H_p, \tag{4.1}$$

$$\boldsymbol{u}^{(i-1)} = \tilde{\boldsymbol{u}}^{(i)} \qquad \forall i = 2, \dots, H_p, \tag{4.2}$$

$$\boldsymbol{x}^{(H_p)} = \tilde{\boldsymbol{x}}^{(H_p)}, \tag{4.3}$$

$$\boldsymbol{u}^{(H_p)} = \boldsymbol{0}. \tag{4.4}$$

The last predictions $\boldsymbol{x}^{(H_p)}$ and $\boldsymbol{u}^{(H_p)}$ can not be taken from the previous time-step because $\tilde{\boldsymbol{x}}^{(H_p+1)}$ and $\tilde{\boldsymbol{u}}^{(H_p+1)}$ do not exist. Instead the trajectory prediction must be explicitly extended by one time-step as done in the equations (4.3, 4.4). This is possible because of the final constraint $\boldsymbol{v}^{(H_p)} = 0$. We need to prove that the trajectory prediction (4.1-4.4) is indeed feasible in the original program (2.28-2.32). We do this by checking every constraint in the original program.

Equation (2.29) for $i = 1$:

$$\boldsymbol{x}^{(1)} = A\boldsymbol{x}^{(0)} + B\boldsymbol{u}^{(1)} \tag{4.5}$$

Subtracting $\tilde{\boldsymbol{x}}^{(2)} = A\tilde{\boldsymbol{x}}^{(1)} + B\tilde{\boldsymbol{u}}^{(2)}$ yields

$$\iff \qquad \underbrace{\boldsymbol{x}^{(1)} - \tilde{\boldsymbol{x}}^{(2)}}_{=0 \text{ (eq. 4.1)}} = A(\boldsymbol{x}^{(0)} - \tilde{\boldsymbol{x}}^{(1)}) + B\underbrace{(\boldsymbol{u}^{(1)} - \tilde{\boldsymbol{u}}^{(2)})}_{=0 \text{ (eq. 4.2)}} \tag{4.6}$$

$$\iff \qquad 0 = \boldsymbol{x}^{(0)} - \tilde{\boldsymbol{x}}^{(1)} \tag{4.7}$$

$$\iff \qquad \boldsymbol{x}^{(0)} = \tilde{\boldsymbol{x}}^{(1)}. \tag{4.8}$$

Equation (4.8) states that the measured state $\boldsymbol{x}^{(0)}$ must match the previous predicted state $\tilde{\boldsymbol{x}}^{(1)}$. In other words, the SCR method assumes that the vehicle follows the predicted trajectory without disturbances. Extending the SCR method to explicitly account for disturbances is open for future research.

Equation (2.29) for $i = 2, \ldots, (H_p - 1)$:

$$\boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)} \tag{4.9}$$

Subtracting $\tilde{\boldsymbol{x}}^{(i+1)} = A\tilde{\boldsymbol{x}}^{(i)} + B\tilde{\boldsymbol{u}}^{(i+1)}$ yields

$$\iff \quad \underbrace{\boldsymbol{x}^{(i)} - \tilde{\boldsymbol{x}}^{(i+1)}}_{=0 \text{ (eq. 4.1)}} = A\underbrace{(\boldsymbol{x}^{(i-1)} - \tilde{\boldsymbol{x}}^{(i)})}_{=0 \text{ (eq. 4.1)}} + B\underbrace{(\boldsymbol{u}^{(i)} - \tilde{\boldsymbol{u}}^{(i+1)})}_{=0 \text{ (eq. 4.2)}} \tag{4.10}$$

$$\iff \quad \mathbf{0} = \mathbf{0} + \mathbf{0}. \tag{4.11}$$

Equation (2.29) for $i = H_p$:

$$\boldsymbol{x}^{(H_p)} = A\boldsymbol{x}^{(H_p-1)} + B\boldsymbol{u}^{(H_p)} \tag{4.12}$$

$$\overset{\text{(eq. 4.1-4.4)}}{\iff} \quad \tilde{\boldsymbol{x}}^{(H_p)} = A\tilde{\boldsymbol{x}}^{(H_p)} \tag{4.13}$$

$$\iff \quad \mathbf{0} = (A - I)\tilde{\boldsymbol{x}}^{(H_p)} \tag{4.14}$$

$$\overset{\text{(eq. 2.14)}}{\iff} \quad \mathbf{0} = \begin{bmatrix} 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{p}_x^{(H_p)} \\ \tilde{p}_y^{(H_p)} \\ \tilde{v}_x^{(H_p)} \\ \tilde{v}_y^{(H_p)} \end{bmatrix} \tag{4.15}$$

$$\overset{\text{(eq. 2.32)}}{\iff} \quad \mathbf{0} = \begin{bmatrix} 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{p}_x^{(H_p)} \\ \tilde{p}_y^{(H_p)} \\ 0 \\ 0 \end{bmatrix} \tag{4.16}$$

$$\iff \quad \mathbf{0} = \mathbf{0}. \tag{4.17}$$

Here we use the constraint $\tilde{\boldsymbol{v}}^{(H_p)} = 0$ which requires that the vehicle comes to a standstill at the end of the predicted trajectory. This allows us to extend the prediction into the future where the vehicle remains at a standstill.

Equation (2.30) for $i = 1, \ldots, (H_p - 1)$:

$$\boldsymbol{p}^{(i)} \in T \tag{4.18}$$

$$\overset{\text{(eq. 4.1)}}{\iff} \quad \tilde{\boldsymbol{p}}^{(i+1)} \in T. \tag{4.19}$$

Equation (2.30) for $i = H_p$:

$$\boldsymbol{p}^{(H_p)} \in T \tag{4.20}$$

$$\overset{\text{(eq. 4.3)}}{\iff} \quad \tilde{\boldsymbol{p}}^{(H_p)} \in T. \tag{4.21}$$

Equation (2.31) for $i = 1, \ldots, (H_p - 1)$:

$$c_{acc}(\boldsymbol{v}^{(i)}, \boldsymbol{u}^{(i)}) \leq 0 \tag{4.22}$$

$$\iff \quad c_{acc}(\tilde{\boldsymbol{v}}^{(i+1)}, \tilde{\boldsymbol{u}}^{(i+1)}) \leq 0. \tag{4.23}$$

Equation (2.31) for $i = H_p$:

$$c_{acc}(\boldsymbol{v}^{(H_p)}, \boldsymbol{u}^{(H_p)}) \leq 0 \tag{4.24}$$

$$\iff \quad c_{acc}(\tilde{\boldsymbol{v}}^{(H_p)}, \boldsymbol{0}) \leq 0 \tag{4.25}$$

$$\iff \quad -1 \leq 0. \tag{4.26}$$

Equation (2.32):

$$\boldsymbol{v}^{(H_p)} = \boldsymbol{0} \tag{4.27}$$

$$\iff \quad \tilde{\boldsymbol{v}}^{(H_p)} = \boldsymbol{0}. \tag{4.28}$$

Thus, we have proven that the initial trajectory prediction (4.1-4.4) is feasible in the original program (2.28-2.32).

4.2 Sequential Convex Restriction

This section discusses the core idea of SCR, which allows a new feasible trajectory prediction to be computed based on an existing feasible trajectory prediction. We use a generalized form of the original program (2.28-2.32) to simplify the argument. The following sections 4.3-4.6 apply the idea of a sequential convex restriction to the original program. The original program (2.28-2.32) can be written in the following form:

$$\begin{aligned} \underset{x}{\text{minimze}} \quad & f(x) && (4.29)\\ \text{subject to} \quad & Ax = b && (4.30)\\ & x \in C, && (4.31) \end{aligned}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $m \leq n$, $f : \mathbb{R}^n \to \mathbb{R}$, $C \subseteq \mathbb{R}^n$. Program (4.29-4.31) is in general non-convex. Let $\tilde{x}$ be a known feasible point in the program (4.29-4.31). We define a convex, restricted version of program (4.29-4.31) based on $\tilde{x}$ as follows:

$$\begin{aligned} \underset{x}{\text{minimze}} \quad & g_{\tilde{x}}(x) && (4.32)\\ \text{subject to} \quad & Ax = b && (4.33)\\ & x \in R(\tilde{x}), && (4.34) \end{aligned}$$

where $g_{\tilde{x}} : \mathbb{R}^n \to \mathbb{R}$ is a convex approximation of the original objective f and $R : C \to \mathcal{P}(C)$ is the *restriction function* which must satisfy the following conditions:

$$R(\tilde{x}) \subseteq C \qquad \forall \tilde{x} \in C \tag{4.35}$$

$$R(\tilde{x}) \text{ is convex} \qquad \forall \tilde{x} \in C \tag{4.36}$$

$$\tilde{x} \in R(\tilde{x}) \qquad \forall \tilde{x} \in C \tag{4.37}$$

Fig. 4.1 illustrates the restriction function and the feasible sets of the programs (4.29-4.31) and (4.32-4.34) for a two-dimensional case. Program (4.32-4.34) has by definition at least one feasible point, which is $\tilde{x}$. Program (4.32-4.34) is also convex because it minimizes a convex function on a convex set. Thus, it has a unique solution. Let $\mathcal{F}(\tilde{x})$ denote this solution.

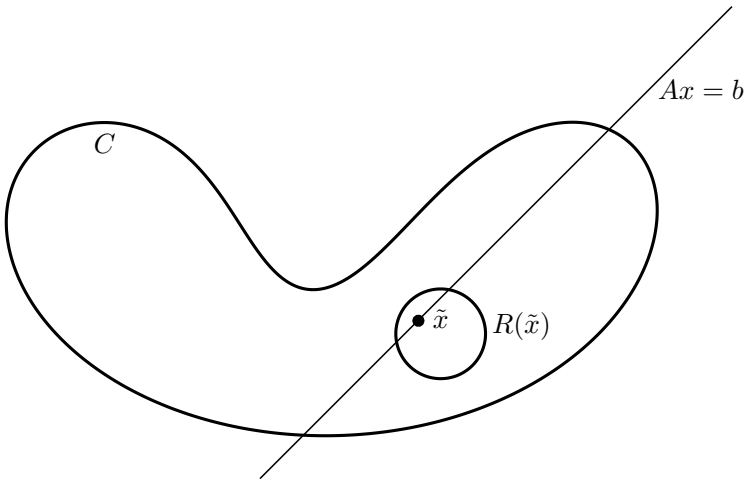

Figure 4.1: Illustration of the restriction function and the linear constraints

Note that $\mathcal{F}(\tilde{x})$ is also a feasible point in the original program (4.29-4.31) because $\tilde{x} \in R(\tilde{x}) \implies \tilde{x} \in C$. Thus $\mathcal{F}(\tilde{x})$ can be used as a new starting point for program (4.32-4.34) which results in $\mathcal{F}(\mathcal{F}(\tilde{x}))$. We can generate a sequence of points $\tilde{x}_{i+1} = \mathcal{F}(\tilde{x}_i)$, each of which is feasible in program (4.29-4.31) by iterating on this process. Whether this sequence approaches an optimal – or at least useful – solution of program (4.29-4.31) depends strongly on the formulation of $g_{\tilde{x}}$ and $R(\tilde{x})$. For example $R(\tilde{x}) = \{\tilde{x}\}$ is a valid but useless restriction function, because it results in $\mathcal{F}(\tilde{x}) = \tilde{x}$.

4.3 Application of Sequential Convex Restriction

To apply SCR to the original program (2.28-2.32) we start by defining how the original program can be rewritten in the form of the generalized program (4.29-4.31). We combine the decision variables from the original program as follows:

$$x = (\boldsymbol{p}^{(1)}, \dots, \boldsymbol{p}^{(H_p)}; \boldsymbol{v}^{(1)}, \boldsymbol{u}^{(1)}, \dots, \boldsymbol{v}^{(H_p)}, \boldsymbol{u}^{(H_p)}) \tag{4.38}$$

where $x \in \mathbb{R}^{6H_p}$ is the decision variable in the generalized program (4.29-4.31). The equations (2.29) and (2.32) are affine and can be combined into the form $Ax = b$ of equation (4.30). The track constraints (2.30) and acceleration constraints (2.31) are non-linear and need to be combined into the form $x \in C$ in equation (4.31). We formulate the acceleration constraint (2.31) as a set as follows:

$$\mathcal{A} = \{(v_x, v_y, a_x, a_y) \in \mathbb{R}^4 \mid c_{acc}((v_x\ v_y), (a_x\ a_y)) \leq 0\} \tag{4.39}$$

$$= \{(\boldsymbol{v}, \boldsymbol{u}) \in \mathbb{R}^2 \times \mathbb{R}^2 \mid c_{acc}(\boldsymbol{v}, \boldsymbol{u}) \leq 0\}, \tag{4.40}$$

where c_{acc} is the acceleration constraint function introduced in section 2.5. Now we can combine the track and acceleration constraints using the Cartesian product as follows:

$$(\boldsymbol{p}^{(i)} \in T) \ \wedge \ ((\boldsymbol{v}^{(i)}, \boldsymbol{u}^{(i)}) \in \mathcal{A}) \quad \forall i = 1, \dots, H_p \tag{4.41}$$

$$\iff \quad x \in \underbrace{(T^{H_p} \times \mathcal{A}^{H_p})}_{=C}, \tag{4.42}$$

where $C \subseteq \mathbb{R}^{6H_p}$ corresponds to the constraint set in equation (4.31) and T is the track area introduced in section 2.7. The original program is a maximization, the generalized program is a minimization. Thus, the objective function is negated as follows:

$$f(x) = -s(\boldsymbol{p}^{(H_p)}) = -s(x_{2H_p-1}, x_{2H_p}), \tag{4.43}$$

where s is the progress function in equation (2.32).

We have defined the original program (2.28-2.32) in terms of the generalized program (4.29-4.31). As the next step in applying SCR to the original program, we need to construct a convex objective $g_{\tilde{x}}$ and a restriction function $R(\tilde{x})$.

Sections 4.4 and 4.5 will formulate the restriction functions for the track area $R_T : T \to \mathcal{P}(T)$ and for the acceleration constraints $R_\mathcal{A} : \mathcal{A} \to \mathcal{P}(\mathcal{A})$ respectively. R_T and $R_\mathcal{A}$ fulfill the conditions for a restriction function (4.35-4.37) on their respective sets T and $\mathcal{A}$. The overall restriction function $R(x)$ is then composed using the Cartesian product as follows:

$$R(x) = R_T(\boldsymbol{p}^{(1)}) \times \ldots \times R_T(\boldsymbol{p}^{(H_p)}) \times R_\mathcal{A}(\boldsymbol{v}^{(1)}, \boldsymbol{u}^{(1)}) \times \ldots \times R_\mathcal{A}(\boldsymbol{v}^{(H_p)}, \boldsymbol{u}^{(H_p)}). \tag{4.44}$$

Note that this definition corresponds directly to the definition of the original constraint set C in equation (4.42) and to the definition of the decision variable in equation (4.38), i.e. the order of the variables is matched.

We need to justify that the Cartesian product of two restriction functions also is a restriction function. We do this by checking that the conditions (4.35-4.37) are preserved by the Cartesian product as follows:

Condition (4.35): $((A_1 \subseteq B_1) \wedge (A_2 \subseteq B_2)) \implies (A_1 \times A_2) \subseteq (B_1 \times B_2)$ [17, 15].

Condition (4.36): (A is convex) $\wedge$ (B is convex) $\implies$ $(A \times B)$ is convex. The Cartesian product of convex sets is convex [2, p. 38].

Condition (4.37): $(a \in A) \wedge (b \in B) \implies (a, b) \in (A \times B)$. This follows directly form the definition of the Cartesian product.

4.4 Race Track Model

4.4.1 Structure of the Track Restriction Function

This section defines the structure of the track area $T \subseteq \mathbb{R}^2$ and the restriction function for the track area $R_T : T \to \mathcal{P}(T)$. Because we want to use R_T in the constraints of a QP, it must not just be convex but also consist of affine constraints. Thus, we can write R_T as follows:

$$R_T(\tilde{\boldsymbol{p}}) = P_{j(\tilde{\boldsymbol{p}})}, \tag{4.45}$$

$$P_j = \{\boldsymbol{p} \in \mathbb{R}^2 \mid F_j \boldsymbol{p} \leq \boldsymbol{g}_j\}, \tag{4.46}$$

where $P_j \subseteq \mathbb{R}^2$, $F_j \in \mathbb{R}^{n \times 2}$, $\boldsymbol{g}_j \in \mathbb{R}^n$ and $j : \mathbb{R}^2 \to \mathbb{N}$ is an index function which selects the correct set of affine constraints. Equation (4.46) can be interpreted geometrically: A bounded set of linear constraints in two dimensions P_j forms a convex polygon. Since the track area is non-convex, multiple polygons P_j are necessary to define it. Thus, we define the track area as a union of convex polygons as follows:

$$T = \bigcup_{j=1}^{N} P_j. \tag{4.47}$$

The index function $j(\boldsymbol{p})$ in the track restriction function $R_T(\boldsymbol{p})$ selects the polygon which includes $\boldsymbol{p}$. If multiple overlapping polygons include $\boldsymbol{p}$, then the polygon whose boundary has the largest distance to $\boldsymbol{p}$ is selected. We assume that the affine constraints that make up the polygons are normalized, i.e. $\|\boldsymbol{f}_{k,j}\| = 1$ where $\boldsymbol{f}_{k,j} \in \mathbb{R}^2$ is a row vector in F_j. Thus, the signed distances to the polygon boundaries are $F_j \boldsymbol{p} - \boldsymbol{g}_j$. The index function can be written as follows:

$$j(\boldsymbol{p}) = \underset{j=1,\ldots,N}{\operatorname{argmin}} \; d_j(\boldsymbol{p}) \tag{4.48}$$

$$d_j(\boldsymbol{p}) = \max_i \; (F_j \boldsymbol{p} - \boldsymbol{g}_j)_i \tag{4.49}$$

where $d_j(\boldsymbol{p}) : \mathbb{R}^2 \to \mathbb{R}$ is the signed distance to P_j and the subscript i selects the i-th entry from the signed distances vector $F_j\boldsymbol{p} - \boldsymbol{g}_j$.

With the above definitions we can prove that the track restriction function $R_T(\tilde{\boldsymbol{p}})$ in equation (4.45) fulfills the conditions (4.35-4.37) for a restriction function.

Condition (4.35):

$$R_T(\tilde{\boldsymbol{p}}) \subseteq T \qquad \forall \tilde{\boldsymbol{p}} \in T \tag{4.50}$$

$$\iff \quad P_{j(\tilde{\boldsymbol{p}})} \subseteq \bigcup_{j=1}^{N} P_j \qquad \forall \tilde{\boldsymbol{p}} \in T \tag{4.51}$$

$$\iff \quad P_k \subseteq \bigcup_{j=1}^{N} P_j \qquad \forall k = 1, \ldots, N. \tag{4.52}$$

Condition (4.36):

$$R_T(\tilde{\boldsymbol{p}}) \text{ is convex} \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.53}$$

$$\iff \quad P_{j(\tilde{\boldsymbol{p}})} \text{ is convex} \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.54}$$

$$\iff \quad P_k \text{ is convex} \quad \forall k = 1, \ldots, N \tag{4.55}$$

$$\iff \quad \{\boldsymbol{p} \in \mathbb{R}^2 \mid F_k\boldsymbol{p} \leq \boldsymbol{g}_k\} \text{ is convex} \quad \forall k = 1, \ldots, N. \tag{4.56}$$

Condition (4.37):

$$\tilde{\boldsymbol{p}} \in R_T(\tilde{\boldsymbol{p}}) \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.57}$$

$$\iff \quad \tilde{\boldsymbol{p}} \in P_{j(\tilde{\boldsymbol{p}})} \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.58}$$

$$\iff \quad \tilde{\boldsymbol{p}} \in \{\boldsymbol{p} \in \mathbb{R}^2 \mid F_{j(\tilde{\boldsymbol{p}})}\boldsymbol{p} \leq \boldsymbol{g}_{j(\tilde{\boldsymbol{p}})}\} \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.59}$$

$$\iff \quad F_{j(\tilde{\boldsymbol{p}})}\tilde{\boldsymbol{p}} \leq \boldsymbol{g}_{j(\tilde{\boldsymbol{p}})} \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.60}$$

$$\iff \quad d_{j(\tilde{\boldsymbol{p}})}(\tilde{\boldsymbol{p}}) \leq 0 \quad \forall \tilde{\boldsymbol{p}} \in T \tag{4.61}$$

$$\iff \quad \left(\min_{k=1,\ldots,N} d_k(\tilde{\boldsymbol{p}})\right) \leq 0 \quad \forall \tilde{\boldsymbol{p}} \in T. \tag{4.62}$$

From $\tilde{\boldsymbol{p}} \in T$ it follows that $\exists m : \tilde{\boldsymbol{p}} \in P_m$ and $d_m(\tilde{\boldsymbol{p}}) \leq 0$. Thus

$$\left(\min_{k=1,\dots,N} d_k(\tilde{\boldsymbol{p}})\right) \leq d_m(\tilde{\boldsymbol{p}}) \leq 0 \quad \forall \tilde{\boldsymbol{p}} \in T. \tag{4.63}$$

We have proven that the track restriction function $R_T(\boldsymbol{p})$ is a valid restriction function by checking the conditions (4.35-4.37).

4.4.2 Construction of the Track Restriction Function

Section 4.4.1 demonstrated that a union of convex polygons can be used to define the track restriction function $R_T(\boldsymbol{p})$. Next, we need an algorithm to calculate that set of convex polygons. The algorithm presented below is motivated by the following requirements:

- The polygons should be as large as possible. With larger polygons fewer iterations are required for a good solution.
- The polygons should overlap. This is necessary to allow predicted trajectory points $\boldsymbol{p}^{(i)}$ to switch from one polygon to another between iterations.
- The resulting track should be equivalent to the track in the SL method (see section 3.2) to allow a comparison between the two methods.

Algorithm 3 shows the pseudocode for the steps to calculate the track polygons.

1: $\mathcal{T} \leftarrow$ tessellateTrack()
2: $\mathcal{T} \leftarrow$ mergePolygons($\mathcal{T}$)
3: $\mathcal{T} \leftarrow$ addOverlaps($\mathcal{T}$)
4: **return** $\mathcal{T}$

Algorithm 3: Pseudocode for the steps to calculate the track polygons

4.4.2.1 Track Tessellation

The track is constructed from the points $\boldsymbol{T}_{L,j}$ and $\boldsymbol{T}_{R,j}$ on the left and right boundary of the track, which were defined in the SL method, see section 3.2. Let *hull* : $\mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}^n)$ be the convex hull function. The convex hull is the smallest convex

set which is a superset of the input set. Then we can define a list of polygons $\mathcal{T}$ that tessellate the track as follows:

$$P_j = hull(\{\boldsymbol{T}_{L,j}, \boldsymbol{T}_{R,j}, \boldsymbol{T}_{L,j+1}, \boldsymbol{T}_{R,j+1}\}) \quad \forall j = 1, \ldots, (N-1), \tag{4.64}$$

$$P_N = hull(\{\boldsymbol{T}_{L,N}, \boldsymbol{T}_{R,N}, \boldsymbol{T}_{L,1}, \boldsymbol{T}_{R,1}\}), \tag{4.65}$$

$$\mathcal{T} = (P_1, P_2, \ldots, P_N), \tag{4.66}$$

where N is the number of points on each track boundary. We assume that we are working with a closed-loop track. Thus, there is a polygon connecting the first and last points of the track. Fig. 4.2 shows $\mathcal{T}$ for an example track.

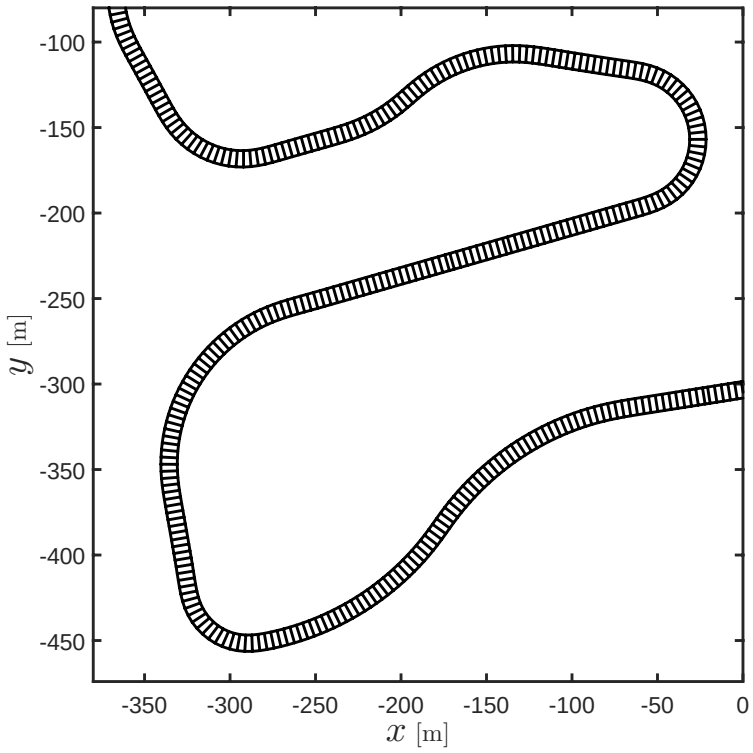

Figure 4.2: Tessellation of an example track

4.4.2.2 Merging Polygons

If the union of two convex polygons P_j and P_{j+1} is also convex, then they can be replaced by their union $P_j \cup P_{j+1}$. To reduce the number of polygons further, we also want to merge polygons, if their union is *almost* convex. We can use the following metric to quantify how close $P_j \cup P_{j+1}$ is to being convex:

$$d_C(P_A, P_B) = |hull(P_A \cup P_B)| - |P_A \cup P_B|. \tag{4.67}$$

From the definition of $hull()$ it follows that $d_C(P_A, P_B) \geq 0$. $d_C(P_A, P_B)$ is the extra area that needs to be added to make $P_A \cup P_B$ convex. In the list of polygons form section 4.4.2.1 we replace P_j and P_{j+1} with $hull(P_j \cup P_{j+1})$ if $d_C(P_j, P_{j+1}) \leq \varepsilon_A$ where $\varepsilon_A > 0$ is some small constant. Fig. 4.3 shows the example track from Fig. 4.2 after the polygons are merged.

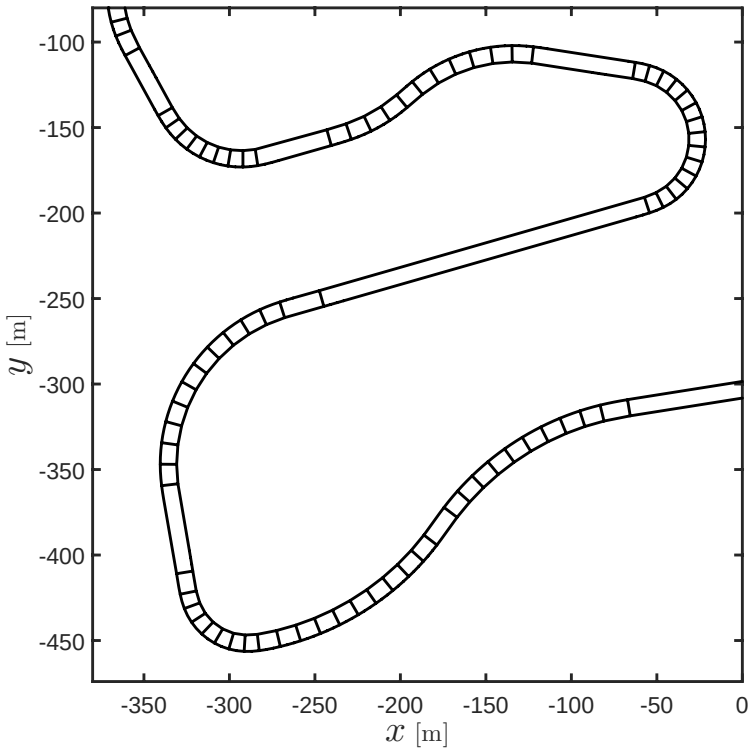

Figure 4.3: Track tessellation with merged polygons

4.4.2.3 Overlaps

If two polygons share an edge, it may be possible to extend each to overlap part of the other one, as illustrated in Fig. 4.4.

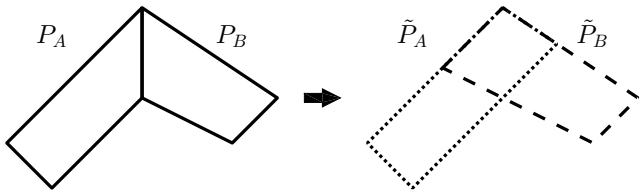

Figure 4.4: Polygon overlaps

We want to find a polygon $\tilde{P}_A$ such that:

$$\tilde{P}_A \subseteq (P_A \cup P_B), \tag{4.68}$$
$$\tilde{P}_A \supseteq P_A, \tag{4.69}$$
$$\tilde{P}_A \text{ is convex.} \tag{4.70}$$

P_A and P_B are convex and share an edge. Thus, they can be written as follows:

$$P_A = \{\boldsymbol{p} \in \mathbb{R}^2 \mid (F_A \boldsymbol{p} \leq \boldsymbol{g}_A) \wedge (\boldsymbol{a}^T \boldsymbol{p} \leq b)\}, \tag{4.71}$$
$$P_B = \{\boldsymbol{p} \in \mathbb{R}^2 \mid (F_B \boldsymbol{p} \leq \boldsymbol{g}_B) \wedge (\boldsymbol{a}^T \boldsymbol{p} \geq b)\}, \tag{4.72}$$

where $\boldsymbol{p}, \boldsymbol{a} \in \mathbb{R}^2$, $b \in \mathbb{R}$, $\boldsymbol{g}_A \in \mathbb{R}^n$, $\boldsymbol{g}_B \in \mathbb{R}^m$, $F_A \in \mathbb{R}^{n \times 2}$, $F_B \in \mathbb{R}^{m \times 2}$. The shared edge is represented by $\boldsymbol{a}^T \boldsymbol{p} = b$. The remaining edges of P_A and P_B are represented by $F_A \boldsymbol{p} \leq \boldsymbol{g}_A$ and $F_B \boldsymbol{p} \leq \boldsymbol{g}_B$ respectively. We define the sets P_A^+ and P_B^+ where the shared edge is removed as follows:

$$P_A^+ = \{\boldsymbol{p} \in \mathbb{R}^2 \mid F_A \boldsymbol{p} \leq \boldsymbol{g}_A\}, \tag{4.73}$$
$$P_B^+ = \{\boldsymbol{p} \in \mathbb{R}^2 \mid F_B \boldsymbol{p} \leq \boldsymbol{g}_B\}, \tag{4.74}$$

Now we define our extended polygon $\tilde{P}_A$ as follows:

$$\tilde{P}_A = hull(P_A \cup (P_A^+ \cap P_B^+)). \tag{4.75}$$

The conditions (4.69) and (4.70) follow immediately from the definition of $\tilde{P}_A$. Although condition (4.68) appears to be true, no proof was found. In the software implementation the condition (4.68) is checked automatically for every instance. No counterexamples were found in the experiments. For every pair of polygons P_A and P_B that share an edge in the track, P_A is replaced by $\tilde{P}_A$. Fig. 4.5 shows the example track from Fig. 4.3 after these overlaps were added. Each polygon is only partially visible because another polygon overlaps it.

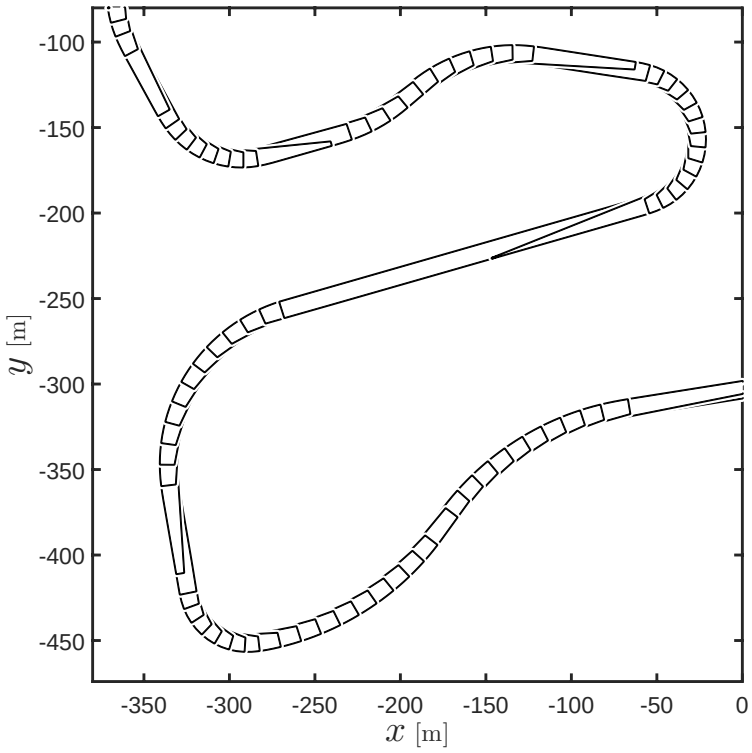

Figure 4.5: Overlapping track polygons, only partially visible due to the overlaps

4.4.3 Objective Function

To apply the SCR method to the trajectory optimization problem we need to define the objective function $g_{\tilde{x}}$ (see equation 4.32). Because $g_{\tilde{x}}$ is used in a QP, it must be convex and quadratic. We define $g_{\tilde{x}}$ as follows:

$$g_{\tilde{x}}(x) = -\boldsymbol{q}^T_{j(\tilde{\boldsymbol{p}}^{(H_p)})}\boldsymbol{p}^{(H_p)}, \tag{4.76}$$

where $\boldsymbol{q}_{j(\tilde{\boldsymbol{p}}^{(H_p)})} \in \mathbb{R}^2$ is the forward direction vector for the current track polygon. Thus, the projection of the final trajectory position $\boldsymbol{p}^{(H_p)}$ onto the forward direction is maximized. We define the forward direction vector for each track polygon as the mean of all forward direction vectors from the SL method that are inside the polygon. $\boldsymbol{q}_j$ can be written as follows:

$$\boldsymbol{q}_j = \frac{1}{|I_j|}\sum_{i \in I_j} \boldsymbol{f}_i \qquad \forall j = 1, \ldots, N, \tag{4.77}$$

$$I_j = \{i \in \{1, \ldots, M\} \mid \boldsymbol{T}_{C,i} \in P_j\} \qquad \forall j = 1, \ldots, N, \tag{4.78}$$

where

- $P_j \in \mathcal{T}$ is a polygon in the SCR track model (see section 4.4.2),
- $N \in \mathbb{N}$ is the number of polygons in the SCR track model,
- $M \in \mathbb{N}$ is the number of discretization points in the SL track model (see section 3.2),
- $\boldsymbol{T}_{C,i} \in \mathbb{R}^2$ is a center-line point in the SL track model,
- $\boldsymbol{f}_i \in \mathbb{R}^2$ is a forward vector in the SL track model,
- $I_j \subseteq \mathbb{N}$ is the index set of center-line points in the polygon P_j.

4.5 Acceleration Constraints

To apply the SCR method to the trajectory optimization problem we need to find an acceleration restriction function $R_{\mathcal{A}}(\boldsymbol{v}, \boldsymbol{u})$ for the acceleration constraint $(\boldsymbol{v}, \boldsymbol{u}) \in \mathcal{A}$

(see equation 4.39). A restriction function for the general form of $\mathcal{A}$ was not found. This is due to the time constraints of this thesis and due to a lack of intuition for four-dimensional geometry. Instead we use the following simplified acceleration constraints:

$$\mathcal{A} = \{(\boldsymbol{v}, \boldsymbol{u}) \in \mathbb{R}^4 \mid (\cos(\tfrac{2\pi i}{n}), \sin(\tfrac{2\pi i}{n}))^T \boldsymbol{u} \leq a_{max} \ \forall i = 1, \ldots, n\}, \tag{4.79}$$

$$R_{\mathcal{A}}(\boldsymbol{v}, \boldsymbol{u}) \equiv \mathcal{A}, \tag{4.80}$$

where a_{max} is the maximum acceleration. These simplified constraints are consistent with the original acceleration constraints in section 2.5 if the original acceleration constraints are simplified as follows:

$$a_{forward,max}(v) = a_{max}, \tag{4.81}$$

$$a_{backward,max}(v) = a_{max}, \tag{4.82}$$

$$a_{lat,max}(v) = a_{max}. \tag{4.83}$$

4.6 Trajectory Optimization Problem

Combining

- the acceleration constraints in section 4.5,
- the track constraints and the objective function in section 4.4,
- the linear kinematic model in section 2.14,
- the terminal constraint in equation (2.32),
- a damping penalty on the input $\boldsymbol{u}$
- and a slack variable ξ to soften the track constraints

yields the following optimization problem:

$$\underset{\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{U}},\xi}{\text{minimize}} \quad -\boldsymbol{q}_{j(\tilde{\boldsymbol{p}}^{(H_p)})}^T \boldsymbol{p}^{(H_p)} + q\xi + R\sum_{i=2}^{H_p} \|\boldsymbol{u}^{(i)} - \boldsymbol{u}^{(i-1)}\|_2^2 \tag{4.84}$$

$$\text{subject to} \quad \boldsymbol{x}^{(i)} = A\boldsymbol{x}^{(i-1)} + B\boldsymbol{u}^{(i)} \qquad \forall i = 1,\dots,H_p \tag{4.85}$$

$$F_{j(\tilde{\boldsymbol{p}}^{(i)})}\boldsymbol{p}^{(i)} \leq \boldsymbol{g}_{j(\tilde{\boldsymbol{p}}^{(i)})} + \xi \qquad \forall i = 1,\dots,H_p \tag{4.86}$$

$$(\cos(\tfrac{2\pi k}{n}), \sin(\tfrac{2\pi k}{n}))^T \boldsymbol{u}^{(i)} \leq a_{max} \qquad \forall i = 1,\dots,H_p$$

$$\forall k = 1,\dots,n \tag{4.87}$$

$$\boldsymbol{v}^{(H_p)} = 0 \tag{4.88}$$

$$0 \leq \xi \tag{4.89}$$

The equations (4.84-4.89) form a convex quadratic optimization program (QP).

The slack variable ξ is added to the track constraints to allow for disturbances from the controlled vehicle. This is necessary for the simulation test in chapter 5. If the vehicle follows the predicted trajectory without disturbances as assumed, the slack variable is not necessary and will be zero[1].

[1] Assuming a sufficiently large slack weight q

5 Implementation and Results

5.1 Implementation

The Sequential Linearization (SL) and Sequential Convex Restriction (SCR) methods, presented in chapters 3 and 4 respectively, are implemented using the MATLAB[12] programming language. Both methods rely on a QP solver. The function `cplexqp()` from the IBM ILOG CPLEX Optimization Studio[10] is used to solve the QPs. The acceleration controllers are implemented using Simulink[13] as shown in section 2.4.

5.2 Simulation

To validate the SL and SCR methods, they are tested in-the-loop with a high-fidelity vehicle dynamics simulation. The software CarMaker[5] is used for the simulation. CarMaker provides 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 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. 5.1, are selected for the evaluation.

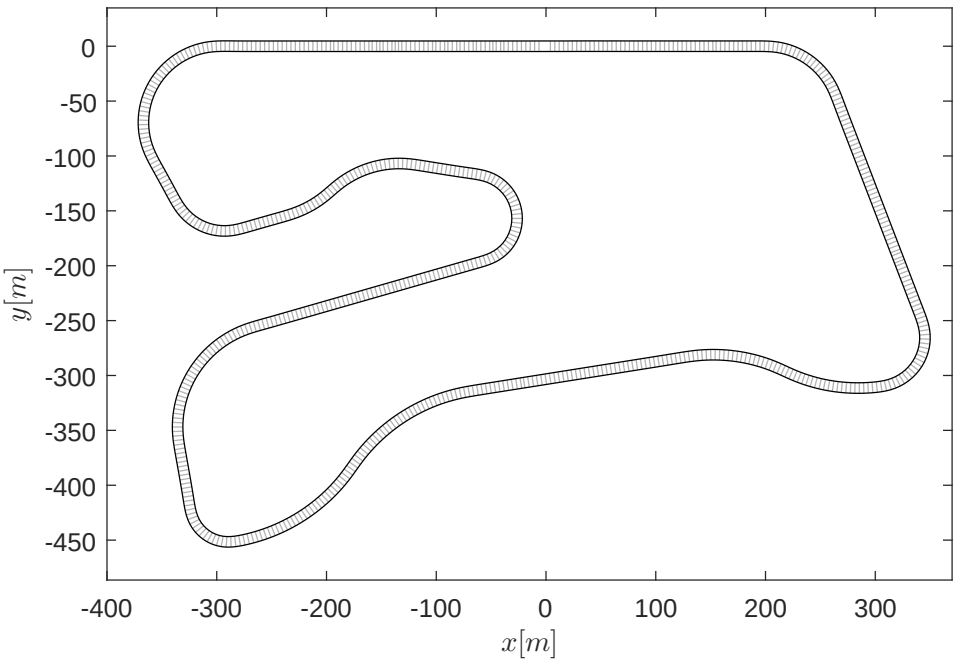

Figure 5.1: CarMaker's Hockenheim example race track

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.

CarMaker includes the software module IPGDriver[6], 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 2.5 are determined from the acceleration that the IPGDriver performs with the simulated vehicle. The IPGDriver's performance is also used as a point of reference for the evaluation of the SL and SCR methods. Because the race track and controlled vehicle are identical in all simulations, a direct comparison between the performance of the IPGDriver and of the SL and SCR methods is possible.

Although a rigorous analysis that shows that the IPGDriver chooses the optimal racing trajectory is not available, there is reason to believe that the IPGDriver racing performance is close to optimal. 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 thesis (see section 2.5). [6]

The primary evaluation criterion is the lowest lap time T_{lap} out of five consecutive laps. Table 5.1 shows the parameters used for the SL method and table 5.2 shows the parameters used for the SCR method. These parameters are manually tuned to minimize lap times.

Name	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

Table 5.1: SL method parameters

Name	Symbol	Value	Unit
Prediction horizon	H_p	20	
Iterations	N	2	
Time-step	Δt	0.5	s
Track width	w	9.4	m
Input change weight	R	0.01	s^4/m^2
Slack weight	q	10^5	$1/m$
Maximum acceleration	a_{max}	22	m/s^2

Table 5.2: SCR method parameters

Fig. 5.2 compares the simulation results of the IPGDriver and the SL and SCR methods. It shows data from the best lap in each simulation run. The best lap times are shown in table 5.3. Fig. 5.2 shows the vehicle's speed v, path curvature κ, longitudinal acceleration and lateral acceleration as a function of the traveled distance s form 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 the simulation runs. The total distance traveled in a lap varies by less than 0.5%.

Method	T_{lap}
IPGDriver	44.64s
SL	44.45s
SCR	48.42s

Table 5.3: Best lap time out of five consecutive laps

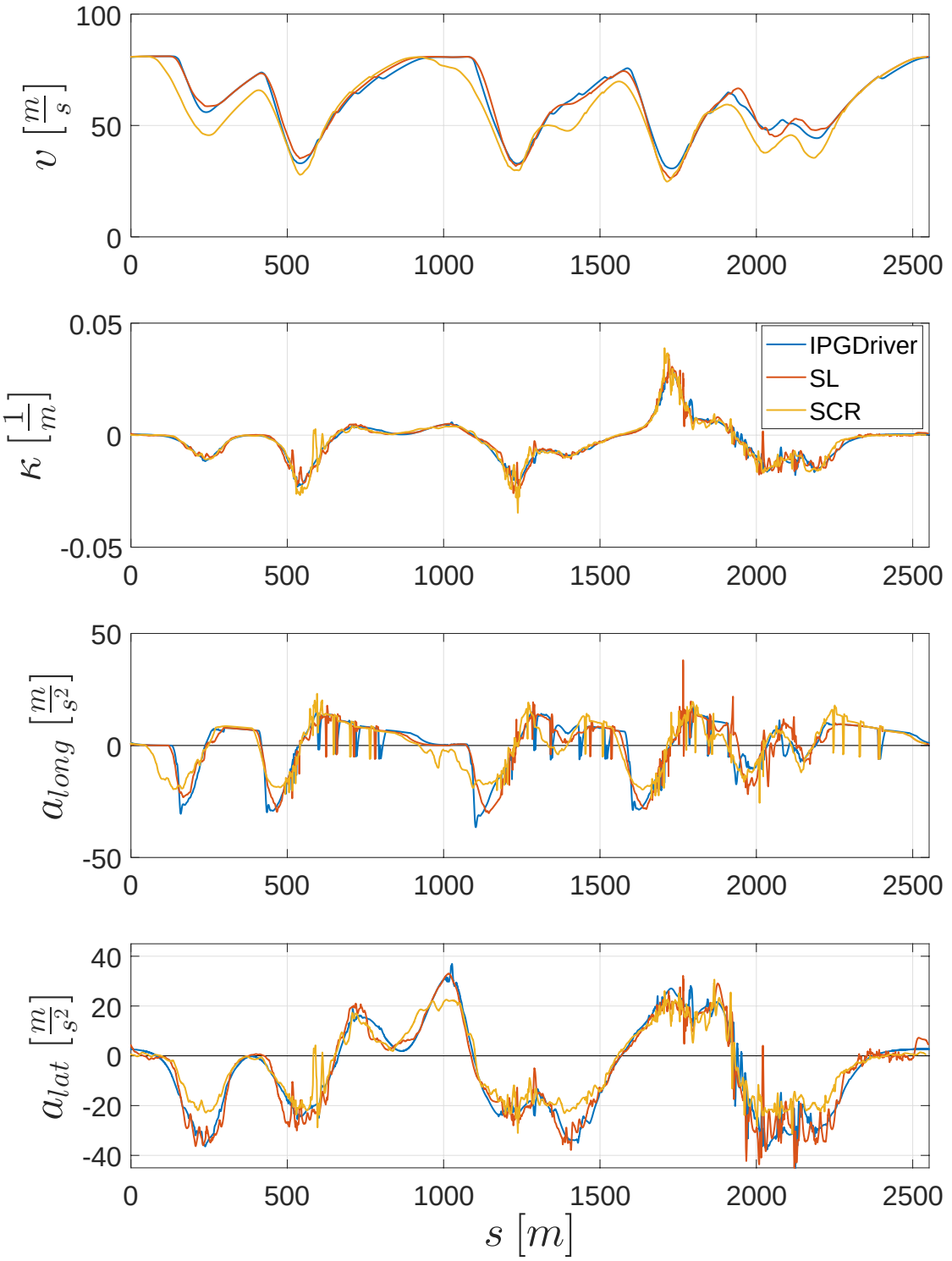

Figure 5.2: Comparison of speed, path curvature, longitudinal acceleration and lateral acceleration of the IPGDriver and the SL and SCR methods

The plot of path curvature κ shows that all three 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. Due to the simplified acceleration model in the SCR method, it can not optimally use the vehicle's capabilities. This results in lower speeds and higher lap times.

Table 5.4 shows some statistics for the computation time per time-step for the SL and SCR methods. 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.

Method	Median	99-percentile	Maximum
SL	39.9ms	58.9ms	158.2ms
SCR	47.8ms	55.2ms	112.9ms

Table 5.4: Computation time statistics for the SL and SCR methods

5.3 Parameter Study

This section analyzes the effects of three key parameters in the trajectory optimization problem. The parameters are the prediction horizon H_p, the prediction time-step Δt and the number of iterations N in the SL and SCR methods. Each parameter is varied individually and its effect on the best lap time T_{lap} and the median computation time T_{comp} is observed. Table 5.5 shows the baseline set of parameters, i.e. the parameters which are not varied. These parameters apply to both the SL and SCR methods, unless otherwise stated.

Name	Symbol	Value	Unit
Prediction horizon	H_p	40	
Iterations	N	3	
Time-step	Δt	0.5	s
Track width	w	9.4	m
Input change weight	R	0.01	s^4/m^2
Slack weight (only SL)	q	10	$1/m^2$
Trust region (only SL)	L	50	m
Slack weight (only SCR)	q	10^5	$1/m$
Maximum acceleration (only SCR)	a_{max}	22	m/s^2

Table 5.5: Baseline parameters

For this parameter study the simulation setup is simplified. The acceleration controller and the vehicle dynamics simulation are removed. Instead the predicted state $\boldsymbol{x}^{(1)}$ is used as the measured state $\boldsymbol{x}^{(0)}$ in the next time-step. This removes disturbances from the vehicle dynamics and allows us to study the SL and SCR methods in isolation.

Fig. 5.3 shows the effect of varying the number of iterations N. Unsurprisingly, the computation time grows linearly with the number of iterations. The lap times show that only a very small number of iterations are necessary for good performance. For the SL method even a single iteration is sufficient.

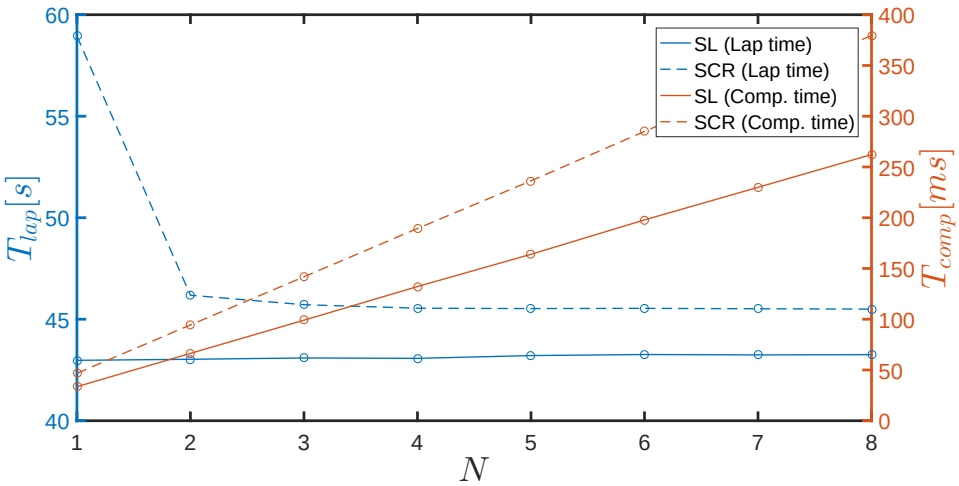

Figure 5.3: Best lap time and median computation time for a varying number of iterations N

Fig. 5.4 shows the effect of varying the prediction horizon H_p. The number of variables in the optimization problem is proportional to H_p. Thus, the computation time grows with H_p. A small prediction horizon results in a short prediction time $T_p = H_p \Delta t$. In the last prediction step the vehicle must come to a standstill. If the prediction time is too short, then the speed is limited, because the planned trajectory only contains the final deceleration maneuver. This increases the lap time. H_p should be sufficiently large, such that the planned trajectory does not need to immediately begin with the final deceleration. A lower bound for H_p can be obtained as follows:

$$\frac{v_{max}}{\bar{a}} \leq T_p = H_p \Delta t, \tag{5.1}$$

where v_{max} is the vehicle's top speed and $\bar{a}$ is the mean deceleration.

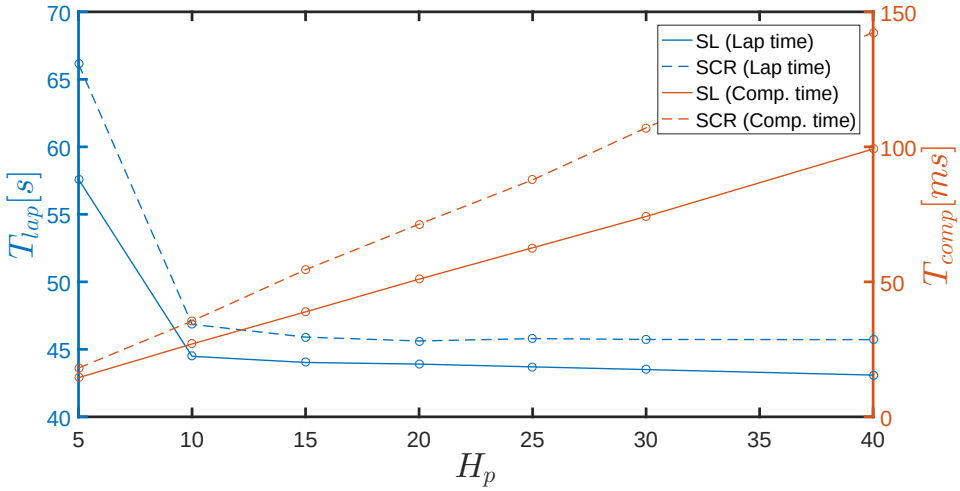

Figure 5.4: Best lap time and median computation time for a varying prediction horizon H_p

Fig. 5.5 shows the effect of varying the prediction time-step Δt. The time-step has no significant effect on the computation time. A small time-step also results in a short prediction time $T_p = H_p \Delta t$, and thus, an increased lap time. Given a required prediction time T_p, there is a trade-off between the prediction horizon and the time-step. A large prediction horizon results in a long computation time. A large time-step reduces the flexibility of the planned trajectory, e.g. maneuvers that require multiple quick input changes may not be possible with a large time-step. A large time-step may also cause the vehicle to clip the track boundary, since only the discrete trajectory points are constrained to the track, not the path between them.

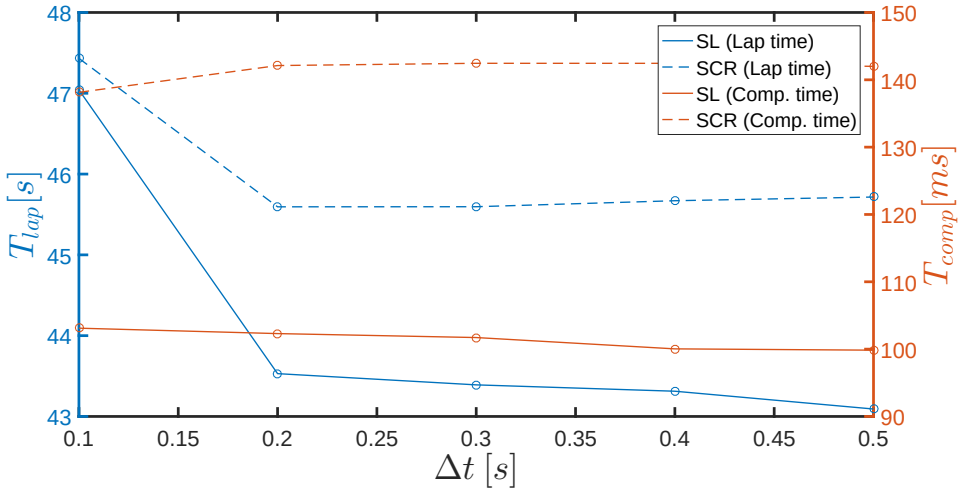

Figure 5.5: Best lap time and median computation time for a varying prediction time-step Δt

6 Conclusion and Outlook

6.1 Conclusion

Chapter 2 formulated the trajectory planning problem for autonomous vehicle racing, using ideas from model predictive control. The chapters 3 and 4 developed the methods Sequential Linearization (SL) and Sequential Convex Restriction (SCR) that find approximate solutions to the trajectory planning problem. It was proven that the SCR method produces feasible trajectories under the assumption of disturbance-free control of the vehicle. Chapter 5 demonstrated that both the SL and SCR methods are capable of guiding a vehicle along a race track in real time. It was also demonstrated that the SL method can achieve a competitive lap time when compared to the IPGDriver method.

6.2 Outlook

Several aspects of this thesis are open for improvements and future research:

Problem formulation: This thesis 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 may need to be considered to achieve the best performance.

Feasibility with disturbances: The SCR method guarantees feasible trajectories under the assumption of disturbance-free control of the vehicle. Extending the SCR method to guarantee feasibility with bounded disturbances is open for future research.

Acceleration model: The SCR method uses a simplified acceleration model (see section 4.5). Developing a speed and direction dependent acceleration model is open for future research.

Bibliography

[1] Stephen Boyd. Sequential convex programming. https://stanford.edu/class/ee364b/lectures/seq_slides.pdf. Accessed: 2017-08-08.

[2] Stephen Boyd and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.

[3] John Canny, Bruce Randall Donald, John Reif, and Patrick G Xavier. On the complexity of kinodynamic planning. Technical report, Cornell University, 1988.

[4] Christodoulos A Floudas and Panos M Pardalos. State of the art in global optimization: computational methods and applications, volume 7. Springer Science & Business Media, 2013.

[5] IPG Automotive GmbH. CarMaker User's guide Version 5.1.2. 2016.

[6] IPG Automotive GmbH. IPGDriver Reference Manual Version 6.5. 2016.

[7] Nicholas Gould and Philippe Toint. A quadratic programming page. http://www.numerical.rl.ac.uk/people/nimg/qp/qp.html. Accessed: 2017-08-08.

[8] Nicholas Gould and Philippe Toint. A quadratic programming bibliography. Numerical Analysis Group Internal Report, 1:32, 2000.

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

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

[11] Jan Marian Maciejowski. Predictive control: with constraints. Pearson education, 2002.

[12] MathWorks. MATLAB Version 9.0 Release 2016a. 2016.

[13] MathWorks. Simulink Version 8.7 Release 2016a. 2016.

[14] Hans Mittelmann. Decision tree for optimization software. `http://plato.asu.edu/sub/nlores.html#QP-problem`. Accessed: 2017-08-08.

[15] ProofWiki. Cartesian product of subsets. `https://proofwiki.org/w/index.php?title=Cartesian_Product_of_Subsets`. Accessed: 2017-09-02.

[16] Rajesh Rajamani. Vehicle dynamics and control. Springer Science & Business Media, 2011.

[17] Seth Warner. Modern algebra. Courier Corporation, 1990.