ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Berkeley
UNIVERSITY OF CALIFORNIA

Maximilian Brunner

Repetitive Learning MPC and its application in autonomous race driving

Master Thesis

Model Predictive Control Lab
UC Berkeley
-
Automatic Control Institute (IfA)
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Ugo Rosolia
Prof. Dr. Francesco Borrelli
Prof. Dr. Roy Smith

March 2017

Contents

Acknowledgement

This is the final work of my master studies of Mechanical Engineering at ETH Zurich and UC Berkeley. I would like to thank Professor Borrelli who gave me the chance to work on this thesis together with him and his team at UC Berkeley. I would also like to thank my advisors Ugo Rosolia and Jon Gonzales who always supported me throughout the entire work in both theoretical and practical questions.
I would like to thank my supervisors at ETH Zurich, Prof. Thomas Roesgen and Prof. Roy Smith, for guiding me through my master's program, providing valuable advice throughout my studies and supporting me for my master thesis.
I would also like to thank my parents for giving me the opportunity to follow my studies at ETH Zurich and writing my master thesis at UC Berkeley.
Last but not least I would like to thank all my friends who I met during my course of studies and who showed me different paths one could follow as a student of engineering. I would not be where I am without them.

Abstract

A Learning Model Predictive Control (LMPC) strategy for periodic tasks is presented. The proposed control learns from previous iteration data to improve the performance of a closed loop system in each iteration. A sampled safe set and a terminal cost function are used to guarantee recursive feasibility and non-decreasing performance cost in each iteration. The proposed control is tested on an autonomous racing example. Vehicle dynamics are identified online using Linear Regression. The control strategy is implemented on a 1:10 scale RC race car. Sensor data from an ultrasound-based GPS and IMU are used to estimate the system state. Finally, experiments show the real-time feasibility of optimal trajectory generation with online system identification.

Notation and Physical Constants

The following abbreviations will commonly be used throughout this thesis:

- MPC: Model Predictive Control
- LMPC: Learning Model Predictive Control
- RLMPC: Repetitive Learning Model Predictive Control
- BARC: Berkeley Autonomous Race Car
- LR: Linear Regression

The states and inputs of the vehicle are:

- States:
  - x, y: inertial position
  - s: curvilinear abscissa
  - v_x: longitudinal velocity
  - v_y: lateral velocity
  - a_x: longitudinal acceleration
  - a_y: lateral acceleration
  - v: total velocity
  - ψ: heading angle
  - $\dot{\psi}$: yaw rate, angular velocity
  - $\ddot{\psi}$: angular acceleration
  - e_ψ: heading angle error
  - e_y: lateral distance to the center line, lateral position error
- Control Inputs:
  - a: acceleration
  - δ_F: steering angle (of the front wheels)

Mathematical variables:
Note that, in general, vectors are denoted bold lower case letters, matrices are bold capital letters, and scalars are written in light letters. Also, optimal solutions are denoted by a superscript asterisk (*).

- N: number prediction steps (horizon length)
- $\boldsymbol{x}_t^j$: State at time step t in iteration j
- $J^j = J^*_{0\to\infty}(\boldsymbol{x}_0^j)$: Total cost of iteration j

- $J^{\text{LMPC},j}_{t\to t+N}(\boldsymbol{x}^j_t)$: optimal prediction cost at x^j_t
- $h(\boldsymbol{x}, \boldsymbol{u})$: Stage cost
- Q^j: Q-function

The physical constants and system parameters are:

- Gravity: $g = 9.81\,\mathrm{m\,s^{-2}}$
- Overall drag coefficient: $c_D = 0.5$
- Car mass: $m = 2.0\,\mathrm{kg}$
- Car moment of inertia: $I_z = 0.03\,\mathrm{kg\,m^2}$
- Wheel radius: $r_w = 0.036\,\mathrm{m}$
- Distance to rear axles: $L_R = 0.125\,\mathrm{m}$
- Distance to front axles: $L_F = 0.125\,\mathrm{m}$

In simulations:

- Road friction coefficient: $\mu = 0.8$

Chapter 1

Introduction

1.1 Background and Motivation

In almost every country and every industry we can see a steady increase and quest for automation today. Sales of industrial robots have more than doubled over the past six years, according to statistics of the International Federation of Robotics [1]. This is just another evidence of the increasing automation of industry - apart from more public fields such as autonomously driving cars. Today, robots are used for all kinds of tasks, whereas many of these tasks can be characterized as being repetitive, meaning they are designed to repeat the same process over and over again. In order to save costs, time, or other resources, the execution of these tasks often has to be optimized over certain criteria. In complex automation tasks, however, it might be difficult to find a trajectory that optimizes the given process.
This is due to multiple reasons: Finding an optimal trajectory mostly implies minimizing or maximizing an objective function. Solving this optimization problem might prove to be difficult or infeasible for highly complex systems or long time horizons. Additionally, it might not be possible to find an exact mathematical description of the system. In order to tackle this problem of optimizing complex system trajectories, previous research has already focused on iteratively improving trajectories. Some methods like the class of Iterative Learning Controls (ILC, [2], [3]) try to find optimal control inputs based on a given reference trajectory. A novel method, called Learning Model Predictive Control (LMPC, [4]), allows finding optimal trajectories without requiring a given reference trajectory.
The work from [4] can be applied on a large class of so-called batch processes. These are characterized by starting each iteration from the same initial state. In this work, we would like to extend this theory to continuous tasks; these feature smooth transitions between iterations and therefore do not require equal initial conditions.
An example application of improving continuous repetitive tasks is the race driving problem, in which the goal is to minimize the lap time while always starting a new lap from where the previous lap ended. We would like to use this scenario to test the performance of the LMPC on a continuous repetitive system.

1.2 Problem description, previous work, and goal

Based on [4], we would like to develop a control strategy for repetitive systems that learns from previous iterations in order to improve its performance. The performance should be determined by an objective function which is defined according to the given problem. This strategy should be applicable for continuous repetitive systems with smooth transitions, meaning that a new iteration begins at the final state of the previous iteration.
We want to combine this control scheme with an online system identification method. This would allow reacting to unknown or changing system dynamics.

Finally, we want to show the practicability of this method in the race driving scenario in simulation and then apply it on a 1:10 scale remote-controlled race car. For that, we need to develop a technique that estimates all necessary states of the systems, using a low-cost system that is fast and easy to set up.
All these tasks have to be computational efficient so that we can perform experiments in real time, driving the car at high velocities.
There has been research on finding the optimal time trajectory through a given race track, an overview can be found in [5]. Mostly, the goal is to find efficient ways of solving the complex optimization problem by making specific assumptions.
As an example, in [6] two approaches are presented. The first one consists of a two-level MPC controller, in which the high-level MPC computes a path while the low-level MPC provides the path tracking. The second approach uses contouring control to combine both tasks in one MPC.
In [7] the authors transform the general race driving problem into an unconstrained problem which allows efficient computation of optimal solutions.
This thesis is organized as follows: In the second chapter, a short introduction to Model Predictive Control (MPC) and Learning Model Predictive Control (LMPC) is given. Based on that, an extension to repetitive systems (RLMPC) is presented. The third chapter introduces the formulation of car system dynamics, the application of RLMPC on race driving, and simulation results. The fourth chapter illustrates the control implementation on a 1:10 scale race car. Experiments show the real-time feasibility of the proposed control strategy. Finally, the fifth chapter provides conclusions and final remarks.

Chapter 2

Repetitive Learning Model Predictive Control

This chapter gives a general introduction to the concept of Model Predictive Control (MPC). Based on that, the theory of Learning Model Predictive Control (LMPC) for repetitive tasks is presented.

2.1 Model Predictive Control

Model Predictive Control is a concept that has been developed since the late 1970s, starting from Chemical Process Industries [8]. Since then, increasing computational power has helped further research on this method and has extended it to other fields beyond chemical industries. This section gives a brief overview of the mathematical concepts of MPC [9].
Given the discrete time system

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathbb{R}^m$ are the system state and input, subject to the constraints

$$\boldsymbol{x}_t \in \mathcal{X},\ \boldsymbol{u}_t \in \mathcal{U},\ \forall t \in \mathbb{Z}_{0+}. \tag{2.2}$$

The MPC controller tries to minimize the following *objective function*,

$$J_{t\to t+N}(\boldsymbol{x}_t, \boldsymbol{u}_{t\to t+N-1}) = \sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}) + p(\boldsymbol{x}_{t+N|t}), \tag{2.3}$$

where N is the time horizon. The terms $h(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and $p(\boldsymbol{x}_N)$ are referred to as *stage cost* and *terminal cost*, respectively. Then, at each time step t, the objective function is minimized and its first optimal input $\boldsymbol{u}^*_{t|t}$ is applied to the system. This results in a Constrained Finite Time Optimal Control (CFTOC) problem:

$$J^*_{t\to t+N}(\boldsymbol{x}_t, \boldsymbol{u}_{t\to t+N-1}) = \min_{\boldsymbol{u}_{t|t},\dots,\boldsymbol{u}_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}) + p(\boldsymbol{x}_{t+N|t}) \right] \tag{2.4a}$$

s.t.

$$\boldsymbol{x}_{k+1|t} = f(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}),\ \forall k \in [t, ..., t+N-1] \tag{2.4b}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}_t \tag{2.4c}$$

$$\boldsymbol{x}_{k|t} \in \mathcal{X},\ \boldsymbol{u}_{k|t} \in \mathcal{U},\ \forall k \in [t, ..., t+N-1] \tag{2.4d}$$

with the optimal solution for the state $\boldsymbol{x}^*_{t|t\to t+N|t}$ and for the input $\boldsymbol{u}^*_{t|t\to t+N-1|t}$. The first optimal input is applied to the system:

$$\boldsymbol{u}_t = \boldsymbol{u}^*_{t|t}. \tag{2.5}$$

Solving Eq. (2.4) and applying the first optimal input (2.5) at every time step results in a *Receding horizon controller (RHC)*. There are different approaches on how this optimization problem is solved, two common ways are the *Batch Approach* and the *Recursive Approach*. In general, the optimization problem can be non-convex.

2.2 Learning Model Predictive Control

Learning Model Predictive Control (LMPC) is a novel control strategy that allows finding an optimal trajectory in an iterative task by learning from previous iterations. It is based on the concept of Iterative Learning Control (ILC). In general, ILCs are control strategies aiming to minimize the tracking error of an iterative process by improving its trajectory with every iteration, learning from previous trajectories. It is limited to fixed time processes with given tracking references. [2], [3] give overviews of existing methods on ILC.
LMPC extends the framework of ILC by adding an optimization function which allows finding an optimal trajectory of optimal duration. Applying this strategy makes it possible to gradually find an optimal trajectory for complex nonlinear systems.

Definition of iterative processes There are two types of iterative processes that have to be distinguished: *Batch processes* and *Continuous repetitive processes* [10]. Batch processes are intermittently run and are usually modeled starting from the same initial state. As opposed to that, continuous repetitive processes transition directly from one iteration to the next one, meaning that the last state of an iteration matches the initial state of the next iteration.

Theory of LMPC In general, the control problem consists of an optimization problem which can be written as follows:

$$J^*_{0\to\infty}(\boldsymbol{x}_0) = \min_{\boldsymbol{u}_0,\boldsymbol{u}_1,\dots} \sum_{k=0}^{\infty} h(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.6a}$$

$$\text{s.t. } \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\ \forall k \geq 0 \tag{2.6b}$$

$$\boldsymbol{x}_0 = \boldsymbol{x}_S \tag{2.6c}$$

$$\boldsymbol{x}_k \in \mathcal{X},\ \boldsymbol{u}_k \in \mathcal{U},\ \forall k \geq 0 \tag{2.6d}$$

with $\boldsymbol{x}_S$ being the initial state at each iteration. Following theory has been adapted from [4].
First, we define the stage cost of a state $\boldsymbol{x}_t^j$ at time t in iteration j as follows:

$$h(\boldsymbol{x}_F, 0) = 0 \text{ and } h(\boldsymbol{x}_t^j, \boldsymbol{u}_t^j) > 0\ \forall\ \boldsymbol{x}_t^j \in \mathbb{R}^n \setminus \{\boldsymbol{x}_F\}, \boldsymbol{u}_t^j \in \mathbb{R}^m \setminus \{0\} \tag{2.7}$$

where $\boldsymbol{x}_F$ is the final state with $f(\boldsymbol{x}_F, 0) = 0$. Then, the cost of one iteration j is defined as the sum of its stage costs:

$$J^j = J^j_{0\to\infty}(\boldsymbol{x}_0^j) = \sum_{k=0}^{\infty} h(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j). \tag{2.8}$$

Similarly, the cost-to-go of one specific state $\boldsymbol{x}_t^j$ of iteration j is defined as

$$J^j_{t\to\infty}(\boldsymbol{x}_t^j) = \sum_{k=t}^{\infty} h(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j). \tag{2.9}$$

LMPC constructs a sampled safe set of feasible trajectories which is used as a terminal state constraint in the MPC formulation:

$$\mathcal{SS}^j = \left\{ \bigcup_{i\in M^j} \bigcup_{t=0}^{\infty} \boldsymbol{x}_t^i \right\} \tag{2.10}$$

where M^j is the set of indexes k associated with successful iterations k for $k \leq j$, defined as

$$M^j = \left\{ k \in [0, j] : \lim_{t \to \infty} \boldsymbol{x}_t^k = \boldsymbol{x}_F \right\}. \tag{2.11}$$

Additionally, a terminal cost function is defined over the sampled safe set:

$$Q^j(\boldsymbol{x}) = \begin{cases} \min\limits_{(i,t) \in F^j(\boldsymbol{x})} J^i_{t \to \infty}(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in \mathcal{SS}^j \\ +\infty, & \text{if } \boldsymbol{x} \notin \mathcal{SS}^j \end{cases} \tag{2.12}$$

with

$$F^j(\boldsymbol{x}) = \left\{ (i,t) : i \in [0, j], t \geq 0 \text{ with } \exists\ \boldsymbol{x}_t^i \in \mathcal{SS}^j \text{ and } \boldsymbol{x}_t^i = \boldsymbol{x} \right\}. \tag{2.13}$$

Using the terminal constraints and terminal cost we can write the finite time constrained optimal control problem:

$$J^{\text{LMPC},j}_{t \to t+N}(\boldsymbol{x}_t^j) = \min_{\boldsymbol{u}_{t|t}, \ldots, \boldsymbol{u}_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}) + Q^{j-1}(\boldsymbol{x}_{t+N|t}) \right] \tag{2.14a}$$

s.t.

$$\boldsymbol{x}_{k+1|t} = f(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}), \forall k \in [t, \ldots, t+N], \tag{2.14b}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}_t^j, \tag{2.14c}$$

$$\boldsymbol{x}_{k|t} \in \mathcal{X}, \boldsymbol{u}_{k|t} \in \mathcal{U}, \forall k \in [t, \ldots, t+N], \tag{2.14d}$$

$$\boldsymbol{x}_{t+N|t} \in \mathcal{SS}^{j-1}. \tag{2.14e}$$

Let

$$\boldsymbol{u}^{*,j}_{t:t+N-1|t} = [\boldsymbol{u}^{*,j}_{t|t}, \ldots, \boldsymbol{u}^{*,j}_{t+N-1|t}] \tag{2.15}$$

$$\boldsymbol{x}^{*,j}_{t:t+N|t} = [\boldsymbol{x}^{*,j}_{t|t}, \ldots, \boldsymbol{x}^{*,j}_{t+N|t}] \tag{2.16}$$

be the optimal solution of Eq. (2.14) at time t of the j-th iteration. Then, at time t of iteration j, the first element of $\boldsymbol{u}^{*,j}_{t:t+N|t}$ is applied to the system (2.1)

$$\boldsymbol{u}_t^j = \boldsymbol{u}^{*,j}_{t|t}. \tag{2.17}$$

This is repeated at time $t+1$, based on the new state $\boldsymbol{x}_{t+1|t+1} = \boldsymbol{x}^j_{t+1}$, yielding a *moving* or *receding horizon* control strategy.
It can be shown that this control strategy is recursively feasible and that the equilibrium $\boldsymbol{x}_F$ is asymptotically stable for the closed-loop system (2.1) and (2.14). Additionally, the iteration cost J^j does not increase with the iteration index j and the trajectory converges to a local optimum for $j \to \infty$.
The LMPC strategy presented above applies for batch processes, i.e. the initial state $\boldsymbol{x}_0^j = \boldsymbol{x}_S$ in Eq. (2.6c) has to be the same for each iteration. However, it is not applicable for continuous repetitive systems, for which the final state of an iteration corresponds to the initial state of the next iteration. The next section shows the extension for the repetitive case.

2.3 Repetitive Learning Model Predictive Control

This section presents a learning control strategy for continuous repetitive systems. We prove recursive feasibility, non-increasing iteration cost, and optimality, using the mathematical tools we introduced in section 2.2.

Periodicity and switching condition First, we assume that the state dynamics are periodic with periodicity vector $\boldsymbol{p}$:

$$f(\boldsymbol{x}_k + \boldsymbol{p}, \boldsymbol{u}_k) = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{p} \tag{2.18}$$

If the dynamics are periodic in one state, the periodicity can be expressed using a standard basis vector $\boldsymbol{p} = P\boldsymbol{e}_i$, where P is the periodicity in state i.
The switching from one iteration to the next one happens when the periodic state reaches P, i.e. $\boldsymbol{x}\boldsymbol{e}_i^T \geq P$.
As for repetitive systems the initial state in one iteration j depends on the final state of the previous iteration $j-1$, it can not be constrained to a single state $\boldsymbol{x}_S$ (as in (2.6c)). The problem from Eq. (2.6) therefore needs to be slightly changed:

$$J^*_{0\to\infty}(\boldsymbol{x}_0) = \min_{\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{u}_1, \dots} \sum_{k=0}^{\infty} h(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.19a}$$

$$\text{s.t. } \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\ \forall k \geq 0 \tag{2.19b}$$

$$\boldsymbol{x}_0 \in \mathcal{L} \tag{2.19c}$$

$$\boldsymbol{x}_k \in \mathcal{X},\ \boldsymbol{u}_k \in \mathcal{U},\ \forall k \geq 0 \tag{2.19d}$$

The set $\mathcal{L}$ contains all feasible initial states.
Continuous repetitive systems are defined by smooth transitions between two iterations. In particular, we have that the final state of iteration $j-1$ becomes the initial state of iteration j,

$$\boldsymbol{x}_0^{j+1} = \boldsymbol{x}_{\bar{t}_j+1}^j - \boldsymbol{p}\ \forall j \geq 0, \tag{2.20}$$

with

$$\bar{t}_j = \arg\min_t \{t+1 : \boldsymbol{x}_{t+1}^j \in \mathcal{X}_F\} \tag{2.21}$$

being the time of the transition between iteration j and $j+1$. Eq. (2.20) guarantees that the state remains within the bounds of one period and does not accumulate. It also allows us to extend the sampled safe set beyond the limit of the period $\boldsymbol{p}$.

Assumption 1. We assume that $\mathcal{SS}^0$ is a given non-empty set and the trajectory $\boldsymbol{x}^0 \in \mathcal{SS}^0$ is feasible and convergent to $\mathcal{X}_F$ with $\boldsymbol{x}_0^1 = \boldsymbol{x}_{\bar{t}_0}^0 - \boldsymbol{p}$.

Remark 1. Assumption 1 assumes that we can provide a feasible trajectory that reaches the terminal set $\mathcal{X}_F$.

Assumption 2. At time $t = 0$ of the j-th iteration we assume that $J^{\text{LMPC}}_{0\to N}(\boldsymbol{x}_0^j) \leq J^{\text{LMPC}}_{0\to N}(\boldsymbol{x}_0^{j-1})$, $\forall j \geq 1$.

Remark 2. Assumption 2 implies that the LMPC cost in the initial condition does not increase from iteration to iteration. In general, this can not be assumed for all types of systems and is difficult to verify. However, in our application of autonomous racing we expect the car to cross the finish line at higher velocities in each iteration. This would suggest this assumption to hold, which was also confirmed in experiments.

Theorem 1. Consider system (2.1) controlled by the LMPC (2.14) and (2.17). Let $\mathcal{SS}^j$ be the sampled safe set at the j-th iteration. Let assumption 1 hold, then the LMPC (2.14) and (2.17) is feasible at $\forall j \geq 1$ and $t \in \mathbb{Z}_{0+}$. Moreover, the closed loop system (2.1) and (2.14) converges to the invariant set $\mathcal{X}_F$ at each j-th iteration $\forall y \geq 0$.

Proof. By assumption 1 $\mathcal{SS}^0$ is a non-empty set. From Eq. (2.20) follows that $\boldsymbol{x}_0^0 = \boldsymbol{x}_0^1$. Therefore, at time $t = 0$ of the first iteration the solution of the LMPC

$$[\boldsymbol{x}_0^0, ..., \boldsymbol{x}_N^0] \in \mathcal{SS}^{j-1} = \mathcal{SS}^0 \tag{2.22}$$

$$[\boldsymbol{u}_0^0, ..., \boldsymbol{u}_{N-1}^0] \tag{2.23}$$

satisfies input and state constraints and is a feasible solution to (2.14).
Now assume that the LMPC is feasible at time t and let $\boldsymbol{x}^{*,j}_{t:t+N|t}$ and $\boldsymbol{u}^{*,j}_{t:t+N-1|t}$ be the optimal trajectory and the input sequence. As the state updates in (2.1) and (2.14b) are assumed to be identical we have $\boldsymbol{x}^{j}_{t+1} = \boldsymbol{x}^{*,j}_{t+1|t}$. Also, the terminal constraint enforces $\boldsymbol{x}^{*,j}_{t+N|t} \in \mathcal{SS}^{j-1}$ and, from

$$Q^{j-1}(\boldsymbol{x}^{*,j}_{t+N|t}) = J^{i^*}_{t^*\to\infty}(\boldsymbol{x}^{*,j}_{t+N|t}) = \sum_{k=t^*}^{\infty} h(\boldsymbol{x}^{i^*}_{k}, \boldsymbol{u}^{i^*}_{k}). \tag{2.24}$$

where $\boldsymbol{x}^{i^*}_{t^*} = \boldsymbol{x}^{*,j}_{t+N|t}$, with (i^*, t^*) being the minimizer in (2.12). At time $t+1$ of iteration j the input sequence

$$[\boldsymbol{u}^{*,j}_{t+1|t}, \boldsymbol{u}^{*,j}_{t+2|t}, ...\boldsymbol{u}^{*,j}_{t+N-1|t}, \boldsymbol{u}^{i*}_{t^*}] \tag{2.25}$$

and the related state trajectory

$$[\boldsymbol{x}^{*,j}_{t+1|t}, \boldsymbol{x}^{*,j}_{t+2|t}, ...\boldsymbol{x}^{*,j}_{t+N-1|t}, \boldsymbol{x}^{i^*}_{t^*}, \boldsymbol{x}^{i^*}_{t^*+1}] \tag{2.26}$$

satisfy input and state constraints. Therefore, (2.25)-(2.26) is a feasible solution for the LMPC (2.14) at time $t+1$.
Now consider the final time step $\bar{t}_j$ of an iteration j. Assume that at time $\bar{t}_j$ of the j-th iteration the LMPC is feasible and let

$$\mathbf{u}^{*,j}_{\bar{t}_j:\bar{t}_j+N|\bar{t}_j} = [\boldsymbol{u}^{*,j}_{\bar{t}_j|\bar{t}_j}, \cdots, \boldsymbol{u}^{*,j}_{\bar{t}_j+N-1|\bar{t}_j}] \tag{2.27}$$

$$\mathbf{x}^{*,j}_{\bar{t}_j:\bar{t}_j+N|\bar{t}_j} = [\boldsymbol{x}^{*,j}_{\bar{t}_j|\bar{t}_j}, \cdots, \boldsymbol{x}^{*,j}_{\bar{t}_j+N|\bar{t}_j}] \tag{2.28}$$

be the optimal solution. By definition (2.20) the system reaches the terminal invariant set at time $\bar{t}_j + 1$ and the following iteration $j+1$ starts. We construct a feasible solution to the LMPC at time $t = 0$ in iteration $j+1$ using (2.27). Due to identical state updates we have

$$\boldsymbol{x}^{j+1}_{0} = \boldsymbol{x}^{j}_{\bar{t}_j+1} - \boldsymbol{p} = \boldsymbol{x}^{*,j}_{\bar{t}_j+1|\bar{t}_j} - \boldsymbol{p}. \tag{2.29}$$

By the definition of $\bar{t}_j$ we have $\boldsymbol{x}^{*,j}_{\bar{t}_j+N} \in \mathcal{X}_F \subset \mathcal{SS}^{j-1}$ and therefore it exists $k \geq 0$ such that

$$\boldsymbol{x}^{*,j}_{\bar{t}_j+N|\bar{t}_j} = \boldsymbol{x}^{i^*}_{t^*} = \boldsymbol{x}^{i^*}_{\bar{t}_j+k+1} \tag{2.30}$$

where (i^*, t^*) are the minimizer in (2.12) with $i^* < j$.
Moreover, we notice that

$$\boldsymbol{x}^{*,j}_{\bar{t}_j+N|\bar{t}_j} - \boldsymbol{p} = \boldsymbol{x}^{i^*}_{k+\bar{t}_j+1} - \boldsymbol{p} = \boldsymbol{x}^{i^*+1}_{k} \in \mathcal{SS}^{i^*+1} \subset \mathcal{SS}^{j}. \tag{2.31}$$

Since $\boldsymbol{x}^{i^*+1}_{k+1} \in \mathcal{SS}^{j}$, the trajectory

$$[\boldsymbol{x}^{*,j}_{\bar{t}_j+1|\bar{t}_j} - \boldsymbol{p}, \cdots, \boldsymbol{x}^{*,j}_{\bar{t}_j+N|\bar{t}_j} - \boldsymbol{p} = \boldsymbol{x}^{i^*+1}_{k}, \boldsymbol{x}^{i^*+1}_{k+1}] \tag{2.32}$$

is a feasible trajectory for the LMPC (2.14). Therefore, (2.32) and the related input sequence is a feasible solution to the LMPC at time t in iteration $j+1$. ■

Theorem 2. Consider system (2.1) in closed loop with the LMPC controller (2.14) and (2.17). Let $\mathcal{SS}^j$ be the sampled safe set at iteration j. Let assumptions 1-2 hold, then the iteration cost $J^j_{0\to\infty}(\cdot)$ does not increase with iteration index j.

Proof. First, we find a lower bound on the j-th iteration cost $J^j_{0\to\infty}(\cdot)$, $\forall\ j > 0$. Consider the realized state and input sequence in iteration j which collects the first element of the optimal state

and input sequence to the LMPC at time t, $\forall t \in \mathbb{Z}_{0+}$. By the definition of the iteration cost in (2.7) and from assumption 2, we have

$$\begin{aligned} J_{0\to\infty}^{j-1}(\boldsymbol{x}_0) &= \sum_{t=0}^{\infty} h(\boldsymbol{x}_t^{j-1}, \boldsymbol{u}_t^{j-1}) = \\ &= \sum_{t=0}^{N-1} h(\boldsymbol{x}_t^{j-1}, \boldsymbol{u}_t^{j-1}) + \sum_{t=N}^{\infty} h(\boldsymbol{x}_t^{j-1}, \boldsymbol{u}_t^{j-1}) \geq \\ &\geq \sum_{t=0}^{N-1} h(\boldsymbol{x}_t^{j-1}, \boldsymbol{u}_t^{j-1}) + Q^{j-1}(\boldsymbol{x}_N^{j-1}) \geq \\ &\geq \min_{\boldsymbol{u}_0,\ldots,\boldsymbol{u}_{N-1}} \left[\sum_{k=0}^{N-1} h(\boldsymbol{x}_k, \boldsymbol{u}_k) + Q^{j-1}(\boldsymbol{x}_N) \right] = \\ &= J_{0\to N}^{\text{LMPC},j}(\boldsymbol{x}_0^{j-1}) \geq J_{0\to N}^{\text{LMPC},j}(\boldsymbol{x}_0^{j}). \end{aligned} \tag{2.33}$$

Given the above expression the proof follows from *Theorem 2* in [4]. ■

Theorem 3. Consider system (2.1) in closed loop with the LMPC controller. Let $\mathcal{SS}^j$ be the sampled safe set in iteration j. Let assumption 1 hold and assume that the closed loop system (2.1) and (2.14) converges to a steady state trajectory $\mathbf{x}^\infty$, for iteration $j \to \infty$. Then, the steady state input $\mathbf{u}^\infty = \lim_{j\to\infty} \mathbf{u}^j$ and the related steady state trajectory $\mathbf{x}^\infty = \lim_{j\to\infty} \mathbf{x}^j$ is a local optimal solution for the infinite horizon optimal control problem (2.6), i.e., $\mathbf{x}^\infty = \mathbf{x}^*$ and $\mathbf{u}^\infty = \mathbf{u}^*$.

Proof. The proof follows as in the LMPC for iterative tasks [4]. ■

2.3.1 Analysis and discussion of the proposed method

The previous section extended the LMPC strategy from [4] for batch processes to repetitive processes. This extension allows smooth transitions between iterations. Additionally, it does not require equal initial states for each iteration. Instead, the initial state of an iteration j matches the final state of the previous iteration $j-1$.
Under the assumption of no model mismatch the proposed method is recursively feasible, stable, and it converges to an optimal solution. In general, the method can be applied to various repetitive systems. However, assumption 2 has to be verified individually in order to guarantee non-increasing cost.

Chapter 3

Application on race driving

This chapter presents the application of the previously developed RLMPC to race driving. First, two car model formulations are presented which approximate real car dynamics with different accuracies. Then, the RLMPC formulation and cost function are introduced and simulations are performed to verify the controller performance.

3.1 Race driving

In general, the goal of race driving is to drive through a given racetrack with the lowest possible lap time. While in real world race driving usually multiple cars are involved, we simplify this problem by assuming only a single car. Mathematically, this can be represented by an optimization problem which minimizes the lap time, given a mathematical model of the car and certain constraints (e.g. lane boundaries, maximum acceleration, and steering angle). Depending on the complexity of the model and the length and shape of the track this can result in a highly complex optimization problem.

3.2 Car model formulations

There are many ways to model the dynamics of a car. There are simple models with few states which approximate real car dynamics for low speeds. More sophisticated models with a great number of states can account for slipping conditions (longitudinal and lateral) and effects of tire suspensions.
This thesis presents two basic models, the *kinematic bicycle model* and the *dynamic bicycle model*. Both assume a two-tire bicycle model in which the two front and rear tires of a real car are combined to one front and one rear tire. Furthermore, the former model assumes that the tires can not slip laterally whereas the latter allows for lateral movements of the tires.
Additionally, two coordinate frames are introduced. The first is an inertial x-y-frame, the second is a track-based frame that uses coordinates relative to the race track.

3.2.1 Kinematic bicycle model

The kinematic bicycle model has been adapted from [11]. It is based on the no-slip condition which means that the velocity vectors at both wheels are directed in the orientation of their respective wheel. Figure 3.1 shows a standard kinematic bicycle. The states in the inertial x-y-frame are the coordinates x and y of the center of the car, the heading angle ψ and the velocity v of the car center. Note that the car center does not necessarily need to coincide with the car's center of gravity since this is a pure geometrical representation of the car model which does not involve masses or forces.

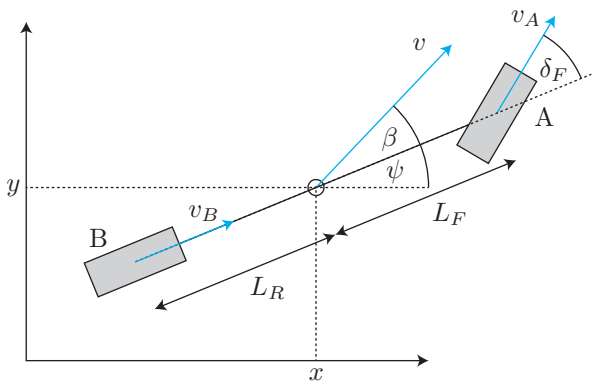

Figure 3.1: Kinematic bicycle model

The state dynamics can be written as

$$\begin{aligned} \dot{x} &= v \cdot \cos(\psi + \beta) \\ \dot{y} &= v \cdot \sin(\psi + \beta) \\ \dot{\psi} &= \frac{v}{L_R} \cdot \sin(\beta) \\ \dot{v} &= a \end{aligned} \tag{3.1}$$

with the car's slip angle

$$\beta = \arctan\left(\frac{L_R}{L_F + L_R} \cdot \tan(\delta_F)\right). \tag{3.2}$$

The control inputs are the longitudinal acceleration a and the steering angle δ_F at the front wheel. The kinematic bicycle approximates real car dynamics well for low velocities and as long as there is no or only very little tire slip. However, especially for slippery road conditions and higher velocities, model mismatch increases to such an extent that it can not be used to reliably predict real car dynamics anymore. This is why the *dynamic bicycle model* is introduced in the next section.

3.2.2 Dynamic bicycle model

While the kinematic bicycle model is derived geometrically, the dynamic bicycle model is based on forces that occur between the tires and the road. Its states are the longitudinal and lateral velocity v_x and v_y in a body fixed frame, and the yaw rate $\dot{\psi}$. The corresponding model is illustrated in Fig. 3.2. The state dynamics presented in eq. (3.3) can be derived by applying Newton's second law of motion [12]:

$$\begin{aligned} \dot{v}_x &= a + \dot{\psi} \cdot v_y \\ \dot{v}_y &= \frac{1}{m} \cdot (F_F \cdot \cos(\delta_F) + F_R) - \dot{\psi} \cdot v_x \\ \ddot{\psi} &= \frac{1}{I_Z} \cdot (L_F \cdot F_F - L_R \cdot F_R) \end{aligned} \tag{3.3}$$

These state dynamics assume that there is only lateral but no longitudinal tire slip which might occur during high accelerations or decelerations. We can avoid longitudinal tire slip by constraining the acceleration input a to small values. This allows us to neglect this effect in the model formulation. The tire forces F_F and F_R depend on the lateral slipping angles α_F and α_R of the front and the rear tire, respectively. One common function that approximates the tire forces is the *Pacejka* function from [13] (often referred to as *Magic formula*):

$$f_P(\alpha) = D \cdot \sin(C \cdot \arctan(B \cdot \alpha)) \tag{3.4}$$

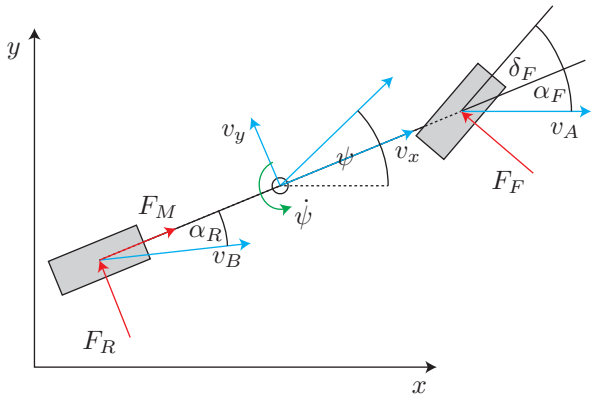

Figure 3.2: Dynamic bicycle model

Note: Eq. (3.4) is a simplified version of the more general Pacejka function.
The Pacejka function is illustrated in Fig. 3.3 for two sets of parameters, one for tire forces on a dry road and one for a snowy road. The chosen values are not parameters of a real tire, they are only meant for illustration purposes. We can see in this figure that the lateral tire force reaches a peak at a specific slip angle before it decreases, leading to the effect commonly known as drifting.

Table 3.1: Pacejka coefficients for different road conditions

Parameter	Meaning	Dry	Snowy
B	Stiffness factor	10	5
C	Shape factor	1.9	2
D	Peak factor	1	0.3

Eq. (3.5a)-(3.5b) show the relations between the Pacejka function and the tire forces, Eq. (3.5c)-(3.5d) are the calculations of slip angles which can be derived geometrically.

$$F_F = -\frac{1}{2} \cdot m \cdot g \cdot \mu \cdot f_P(\alpha_F) \tag{3.5a}$$

$$F_R = -\frac{1}{2} \cdot m \cdot g \cdot \mu \cdot f_P(\alpha_R) \tag{3.5b}$$

$$\alpha_F = \arctan\left(\frac{\dot{y} + L_F\dot{\psi}}{|\dot{x}|}\right) - \delta_F \tag{3.5c}$$

$$\alpha_R = \arctan\left(\frac{\dot{y} - L_R\dot{\psi}}{|\dot{x}|}\right) \tag{3.5d}$$

The terms m, g, and μ are the mass, standard gravity, and road-tire friction, respectively. Their product in Eq. (3.5a)-(3.5b) yields the maximum tire friction force.
The dynamic bicycle model is more complex than the kinematic model but approximates real car dynamics very well even for higher speeds and various road conditions. However, it is important to choose the right set of Pacejka coefficients or measure them properly to obtain reliable results. Also, one might experience numerical problems when using first order approximations of this model. Too long time steps can result in alternating and increasing slip angles and tire forces.

3.2.3 Track reference frame

So far two models were introduced: The kinematic model was constructed in an inertial frame whereas the dynamic model was constructed in a body-fixed frame. However, in race driving,

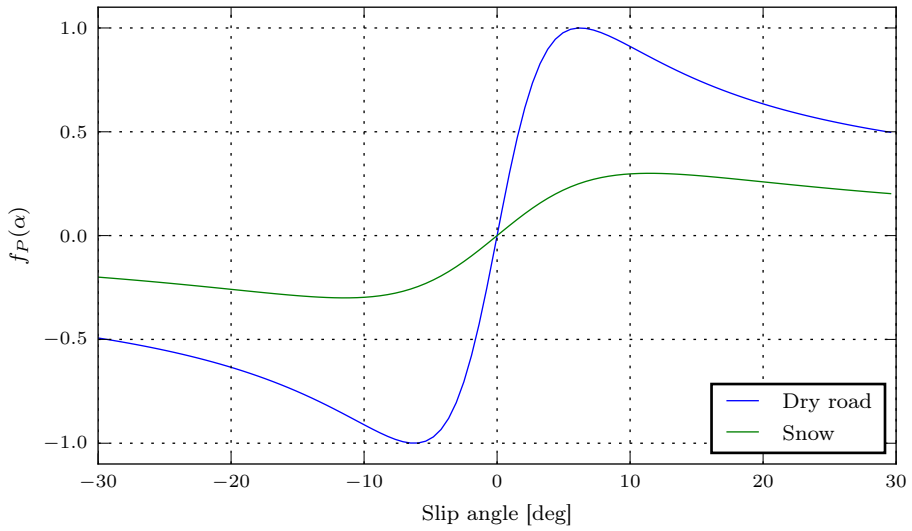

Figure 3.3: Pacejka tire model

we assume that the car follows a given track. In order to simplify calculations, we introduce a reference frame that expresses state dynamics in terms of coordinates relative to the race track. The previous coordinates are mapped to a new set of coordinates which are the *curvilinear abscissa* s, the *lateral position error* from the track's center line e_Y, and the *heading error* e_ψ. This type of frame is also referred to as *Fernet frame* and has been used in previous publications [14]. The coordinate frame is illustrated in Fig. 3.4.

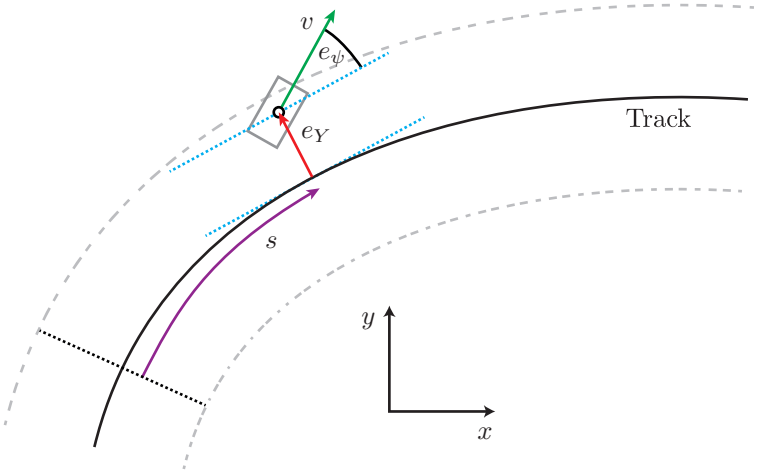

Figure 3.4: Dynamic bicycle model

The new state dynamics use the curvature $c(s)$ to describe the shape of the race track. The curvature is defined as the inverse of the curve radius

$$c(s) = \frac{1}{r(s)}. \tag{3.6}$$

For a race track given in global coordinates $x(s)$ and $y(s)$, the curvature can be calculated by

$$c = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}}. \tag{3.7}$$

The state dynamics of the *kinematic model* in the track frame are derived as follows (refer to [11]):

$$\dot{s} = v \cdot \frac{\cos(e_\psi + \beta)}{1 - e_Y \cdot c(s)} \tag{3.8a}$$

$$\dot{e}_Y = v \cdot \sin(e_\psi + \beta) \tag{3.8b}$$

$$\dot{e}_\psi = \frac{v}{L_F} \cdot \sin(\beta) - v \cdot c(s) \cdot \frac{\cos(e_\psi + \beta)}{1 - e_Y \cdot c(s)} \tag{3.8c}$$

$$\dot{v} = a \tag{3.8d}$$

The *dynamic bicycle model* in the track frame is described by following equations:

$$\dot{e}_Y = v_x \cdot \sin(e_\psi) + v_y \cdot \cos(e_\psi) \tag{3.9a}$$

$$\dot{e}_\psi = \dot{\psi} - c(s) \cdot \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_Y \cdot c(s)} \tag{3.9b}$$

We are going to use the models of Eq. (3.8) and Eq. (3.9) for all further simulations in combination with a given racetrack, i.e. a given curvature function $c(s)$.

3.3 LMPC formulation

This section presents the LMPC formulation of the race driving problem. In general, race driving is a minimum time problem: The variable that is to be optimized is the time the car takes to drive from the start to the finish line.
We use the dynamic bicycle model with following states and inputs:

$$\boldsymbol{x}_t^j = \begin{pmatrix} v_{x,t}^j & v_{y,t}^j & \dot{\psi}_t^j & e_{\psi,t}^j & e_{y,t}^j & s_t^j \end{pmatrix}^T \tag{3.10}$$

$$\boldsymbol{u}_t^j = \begin{pmatrix} a_t^j & \delta_t^j \end{pmatrix}^T, \tag{3.11}$$

and Eq. (2.14) as the optimization function. The racing problem is formulated by using a constant cost function whenever the car is located between the start and finish line and zero cost when the car has crossed the finish line [4]. Thus the stage cost is defined as follows:

$$h(\boldsymbol{x}_t, \boldsymbol{u}_t) = \begin{cases} 1 & \text{if } \boldsymbol{x}_t \notin \mathcal{X}_F, \\ 0 & \text{if } \boldsymbol{x}_t \in \mathcal{X}_F, \end{cases} \tag{3.12}$$

where $\mathcal{X}_F$ is the set of points beyond the finish line at s_{target}:

$$\mathcal{X}_F = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \boldsymbol{x}\boldsymbol{e}_6^T = s > s_{target} \right\}, \tag{3.13}$$

where e_6 is the 6-th standard basis in $\mathbb{R}^6$.
Lane boundaries are modeled as state constraints on the lateral position error e_Y:

$$-\frac{w_{Lane}}{2} \leq e_Y \leq \frac{w_{Lane}}{2}, \tag{3.14}$$

with lane width w_{Lane}.

3.4 Approximation of the safe set and LP relaxation

The LMPC problem from (2.14) minimizes its objective function over the safe set $\mathcal{SS}$. Since the safe set is a set of discrete states of previous iterations, this formulation is a Mixed Integer Nonlinear Programming (MINLP) problem. As this type of problem is generally computationally challenging,

we introduce the concept of safe set relaxation to transform the MINLP to a continuous problem. For this purpose, we approximate the safe set and its cost function using a convex combination of two polynomials as in [15].
We define the time varying approximated safe set $\tilde{\mathcal{SS}}_t^{j,j-1}$ as follows:

$$\tilde{\mathcal{SS}}_t^{j,j-1} = \left\{ \boldsymbol{x} \in \mathbb{R}^5, \lambda \in [0,1] : \boldsymbol{x} \in \lambda \tilde{\boldsymbol{x}}_t^{j,j-1} + (1-\lambda)\tilde{\boldsymbol{x}}_t^{j,j-2} \right\} \tag{3.15}$$

where $\tilde{\boldsymbol{x}}_t^{j,k}$ is an n-th degree polynomial that approximates the k-th trajectory locally:

$$\tilde{\boldsymbol{x}}_t^{j,k} = \left\{ \boldsymbol{x} \in \mathbb{R}^5 : \forall i \in \{v_x, v_y, \dot{\psi}, e_\psi, e_y\}, i = \begin{pmatrix} s^n & s^{n-1} & ... & s & 1 \end{pmatrix} \boldsymbol{\Gamma}_{t,i}^{j,k} \right\}, \tag{3.16}$$

where $\boldsymbol{\Gamma}_{t,i}^{j,k}$ is a vector containing the coefficients of the polynomial at time t.
We also define the time varying function $C_t^{j,k}(\cdot)$ to approximate the cost-to-go function $J_{t\to\infty}^k(\cdot)$:

$$C_t^{j,k}(\boldsymbol{x}) = \begin{cases} \begin{pmatrix} s^n & s^{n-1} & ... & s & 1 \end{pmatrix} \boldsymbol{\Delta}_t^{j,k}, & \text{if } \boldsymbol{x} \in \tilde{\boldsymbol{x}}_t^{j,k}, \\ +\infty, & \text{if } \boldsymbol{x} \notin \tilde{\boldsymbol{x}}_t^{j,k}, \end{cases} \tag{3.17}$$

where $\boldsymbol{\Delta}_t^{j,k}$ contains the coefficients of the polynomial approximating the cost-to-go in iteration k.
We define the continuous time varying approximation of $Q^{j-1}(\cdot)$:

$$\tilde{Q}_t^{j,j-1}(\boldsymbol{x}, \lambda) = \begin{cases} \lambda C_t^{j,j-1}(\boldsymbol{x}) + (1-\lambda) C_t^{j,j-2}(\boldsymbol{x}), & \text{if } (\boldsymbol{x}, \lambda) \in \tilde{\mathcal{SS}}_t^{j,j-1}, \\ +\infty, & \text{if } (\boldsymbol{x}, \lambda) \notin \tilde{\mathcal{SS}}_t^{j,j-1}. \end{cases} \tag{3.18}$$

Having defined the relaxation of the safe set and approximation of the terminal cost function we can rewrite the LMPC formulation from Eq. (2.14):

$$\tilde{J}_{t\to t+N}^{\text{LMPC},j}(\boldsymbol{x}_t^j) = \min_{\lambda, \boldsymbol{u}_{t|t}, ..., \boldsymbol{u}_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, u_{k|t}) + \tilde{Q}^{j,j-1}(\boldsymbol{x}_{t+N|t}, \lambda) \right] \tag{3.19a}$$

s.t.

$$\boldsymbol{x}_{k+1|t} = f(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}), \forall k \in [t, ..., t+N], \tag{3.19b}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}_t^j, \tag{3.19c}$$

$$\boldsymbol{x}_{k|t} \in \mathcal{X}, \boldsymbol{u}_k \in \mathcal{U}, \forall k \in [t, ..., t+N], \tag{3.19d}$$

$$(\boldsymbol{x}_{t+N|t}, \lambda) \in \tilde{\mathcal{SS}}^{j,j-1}. \tag{3.19e}$$

In practice, the polynomial degree n as well as the number of points that are used for the approximation are design parameters. The polynomial degree has to be chosen large enough to approximate the trajectory in the chosen area well enough. However, a high degree polynomial function can lead to numerical difficulties for the solver due to large derivative values and multiple local minima. Similarly, the number of points has to be chosen in such a way that the predicted terminal state $\boldsymbol{x}_N$ lies inside the approximated area. This is why this method works especially well if the approximated region is not too large (i.e. if the horizon N is not too long) and if the trajectories can be well approximated by polynomials of the chosen degree.
For simulations and experiments in this thesis, a polynomial degree of $n = 4...6$ proved to work well.
In the case of no model mismatch it would be sufficient to use only the previous iteration in the safe set. However, since we expect some model mismatch in experiments, we always select the previous iteration as well as a second iteration with lowest iteration cost for the relaxation of the safe set and the approximation and the Q function.
In general, this approach could be extended to using more than two trajectories in the safe set. However, this would require more than one coefficient λ and make the optimization problem more complex.

Table 3.2: Simulation parameters

Parameter	Value
Mass	$m = 2.0\,\mathrm{kg}$
Axle distances	$L_F = L_R = 0.125\,\mathrm{m}$
Moment of inertia	$I_z = 0.03\,\mathrm{kg\,m^2}$

3.5 Simulations

The LMPC strategy is applied on the kinematic and the dynamic model in the track frame. The results are then mapped to the inertial $x - y$ frame to visualize the trajectory.
The physical parameters of the model are taken from a 1:10 scale RC car that is also used for real experiments. They are listed in Tab. 3.2.
The curvature $c(s)$ is given and constructed in such a way that the track is closed (finish line = start line). It is modeled as a piecewise linear and continuous function. At each time step, the curvature is locally approximated by a polynomial. This approximation is needed in the control formulation to allow a smooth prediction of the curvature at every s. The curvature used for simulations is shown in Fig. 3.5. Negative curvatures are interpreted as right turns, positive curvatures as left turns. The track used for simulations has a length of $s_{Total} = 50.49\,\mathrm{m}$.
Note: Constructing the curvature $c(s)$ by a piecewise polynomial function (polynomial degree $n > 1$) and thus making it differentiable could improve the approximation by polynomials even more. However, using a piecewise linear function proved to yield good results for our application.

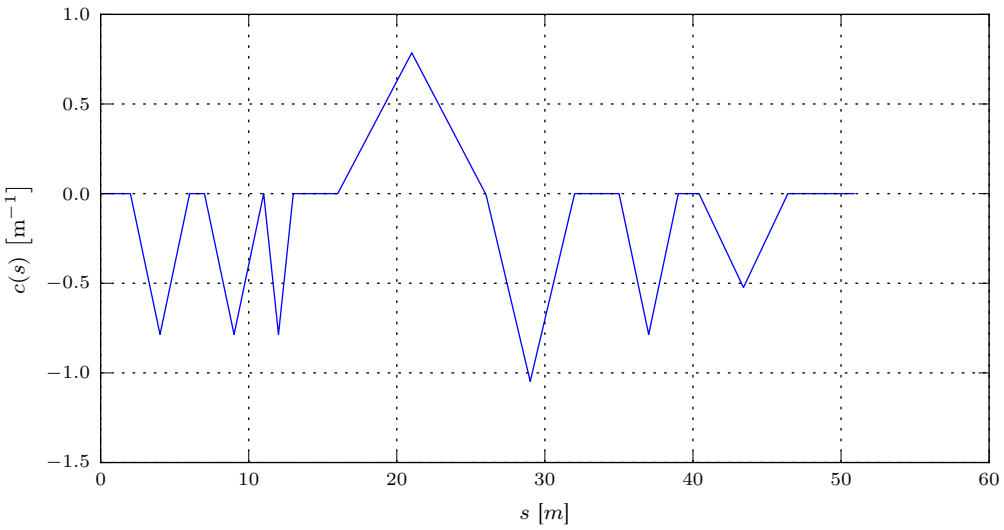

Figure 3.5: Curvature $c(s)$

Lane constraints Lane constraints are implemented as soft constraints in the cost function [9, p. 114]:

$$J_{Lane} = \sum_{i=1}^{N} [c_1 \epsilon_i^2 + c_2 \epsilon_i], \text{ with } \epsilon_i = |e_{Y,i}| - \frac{w_{Lane}}{2} \tag{3.20}$$

with lane width $w_{Lane} = 0.8\,\mathrm{m}$. This is done so that the solver is able to solve the LMPC problem even if model mismatch leads to the car leaving the track boundaries. At the same time it facilitates

the solving process even when the trajectory is within the track boundaries and thus allows real time feasibility.

Soft terminal constraint The terminal state constraint (2.14e) is implemented as soft constraint. This has two main advantages: First, soft constraints can be handled easier and faster by numerical solvers. Second, the soft constraint can be tuned in order to allow small constraint violations. This leads to more freedom in finding better trajectories. On the other hand, violating the constraint can also lead to non-feasible trajectories.

3.5.1 Simulation using the kinematic model

The kinematic bicycle model is a purely geometrical model which can, theoretically, accelerate infinitely fast and take infinitely small turns. This is why the unconstrained lap time minimization problem eventually would result in a geometrical shortest path problem with infinite velocity.
In order to avoid this issue and to approximate a realistic car, following constraints and costs are added to the standard LMPC formulation:

Constraints on input: Both acceleration and steering angle are limited:

$$\begin{aligned} -1.0\,\mathrm{m\,s^{-2}} \leq a \leq 1.0\,\mathrm{m\,s^{-2}}, \\ -0.3\,\mathrm{rad} \leq \delta_F \leq 0.3\,\mathrm{rad}. \end{aligned}$$

Derivative cost: A cost term penalizing the derivatives of the input and the state is added to the cost function:

$$\begin{aligned} J_{deriv} = &\sum_{k=t}^{t+N-1} (\boldsymbol{x}_{k+1|t} - \boldsymbol{x}_{k|t})^T \boldsymbol{Q}_x (\boldsymbol{x}_{k+1|t} - \boldsymbol{x}_{k|t}) \\ &+ \sum_{k=t-1}^{t+N-2} (\boldsymbol{u}_{k+1|t} - \boldsymbol{u}_{k|t})^T \boldsymbol{Q}_u (\boldsymbol{u}_{k+1|t} - \boldsymbol{u}_{k|t}) \end{aligned} \tag{3.21}$$

with diagonal derivative cost coefficients $\boldsymbol{Q}_x$ and $\boldsymbol{Q}_u$. This penalizes fast changes in control inputs which would physically not be possible.

Figures 3.7 to 3.6 show the results of the race problem with the kinematic model. The simulation was performed at a constant sampling rate of $dt = 0.1\,\mathrm{s}$ with a horizon length of $N = 20$.
Both the simulation and the LMPC formulation use the same kinematic model. However, since the track's curvature is approximated by a polynomial it cannot fit the given curvature exactly throughout the entire track. This leads to slight model mismatches.

Evaluation of the results The first lap is performed using a pure path following strategy at a constant reference speed of $1.0\,\mathrm{m\,s^{-1}}$. This is done to construct a safe set of one iteration.
Figure 3.7 shows a 2D-view of the race track and the comparison of three laps. We can see that the trajectory converges to a shortest path through the given racetrack. There are still slight deviations from the ideal shortest path (visible in Fig. 3.8 in lap 20 at $s = 0$ and $s = 33\,\mathrm{m}$). These result from a slow learning process in long straight sections of the track and would converge completely to the lane boundary (at $e_Y = -0.4\,\mathrm{m}$) after more iterations.
Figure 3.9 shows the velocity during three selected laps. The car reaches its maximum velocity (which is limited by constrained acceleration) in the first learning iteration and keeps it throughout the entire lap. Figure 3.6 shows the iteration cost (i.e. lap time) as a function of the iteration number. As the controller accelerates to its maximum speed in the first learning iteration, the lap time decreases quickly over the first two iterations. Afterwards, all performance improvements result from shortening the path by cutting corners and driving along lane boundaries. However, this does not lead to significant improvements. The lap time converges to approx. 45.3 s, slight oscillations result from imprecise approximations of the curvature.

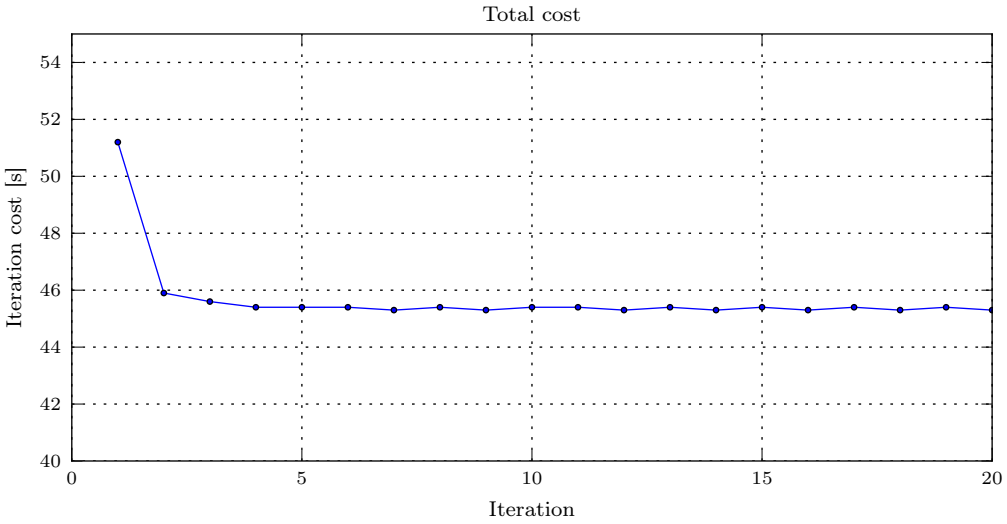

Figure 3.6: Iteration Cost, Kinematic Model

3.5.2 Simulation using the dynamic model

Similar to the kinematic model, simulations using the dynamic bicycle model from Eq. (3.3) were performed. The sampling time was $dt = 0.04\,\mathrm{s}$. In order to avoid unrealistic velocities and numerical issues, the acceleration input was constrained to $-1.8\,\mathrm{m\,s^{-2}} \leq a \leq 1.8\,\mathrm{m\,s^{-2}}$. We used the physical parameters of the BARC and a Pacejka model with coefficients $B = 2.0, C = 2.0, D = 0.5$. Figures 3.11 to 3.13 show the simulation results.

Evaluation of the results The first two laps are controlled by a path following controller at a reference speed of $1.5\,\mathrm{m\,s^{-1}}$. After that, the LMPC strategy is able to reduce the lap time to 20.5 s at an average speed of $2.4\,\mathrm{m\,s^{-1}}$.
Figure 3.10 presents the iteration cost as a function of the iteration number. A constant lap time is reached after approximately 10 iterations. The learning rate (i.e. the convergence speed) depends mainly on the horizon length N. In this simulation we chose a horizon length of $N = 25$ which corresponds to a prediction time of 1.0 s. Additionally, the learning rate is strongly influenced by the soft terminal state constraint. Tightening this soft constraint not only increases computation time but also slows down the learning rate.
As opposed to the kinematic model, the velocity changes through one iteration as it needs to slow down in sharp curves and accelerate in straight sections. This variation of velocities can be seen in Fig. 3.13. Fig. 3.12 compares longitudinal and lateral velocities of multiple laps during the learning process. While the velocities increase quickly over the first 5 iterations, they eventually converge to a steady state trajectory.
Figure 3.14 illustrates a plot of longitudinal and lateral acceleration which is also known as the *friction circle*. It can be seen that the maximum lateral accelerations display maximal values of $1.4\,\mathrm{m\,s^{-2}}$. Longitudinal accelerations are smaller, because we constrained and used derivative cost on the acceleration input in order to simulate low tire forces in longitudinal direction.

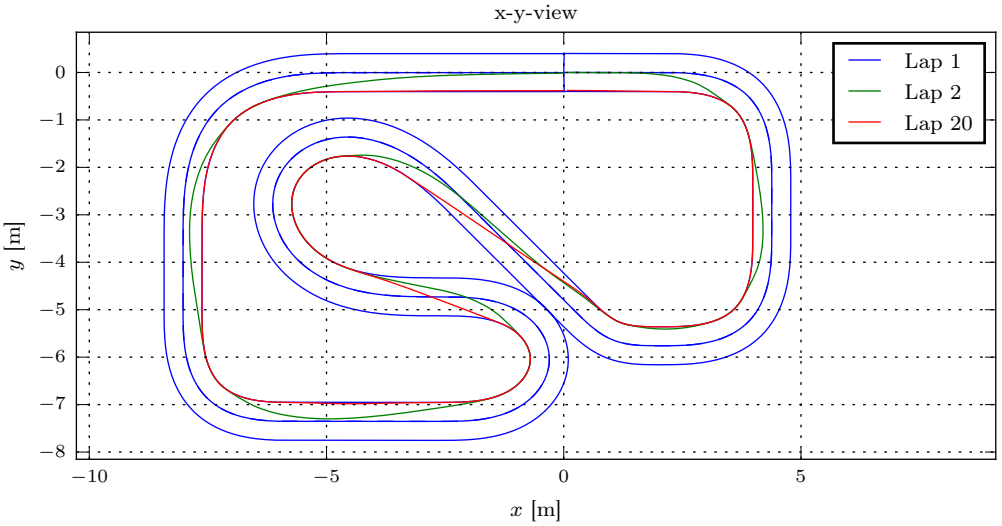

Figure 3.7: Top view, Kinematic Model

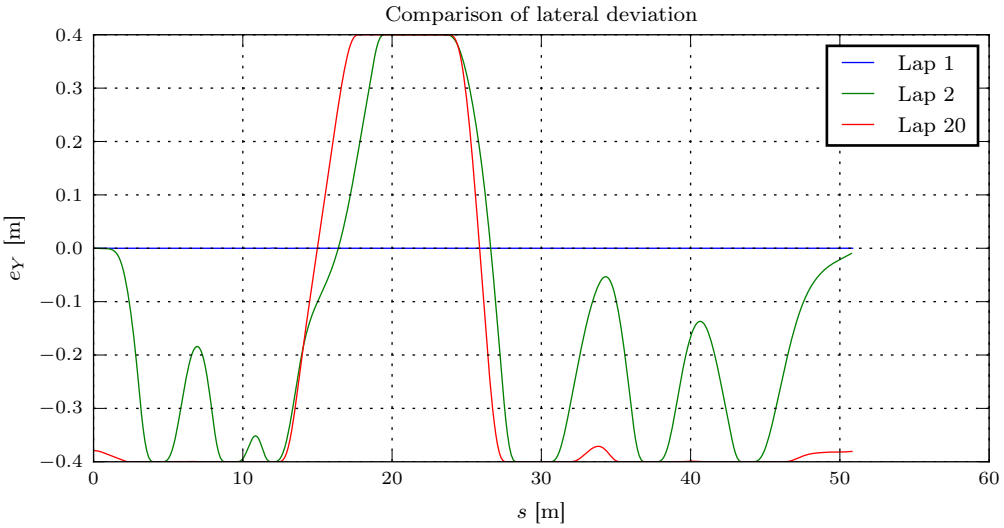

Figure 3.8: Lateral distance error e_Y, Kinematic Model

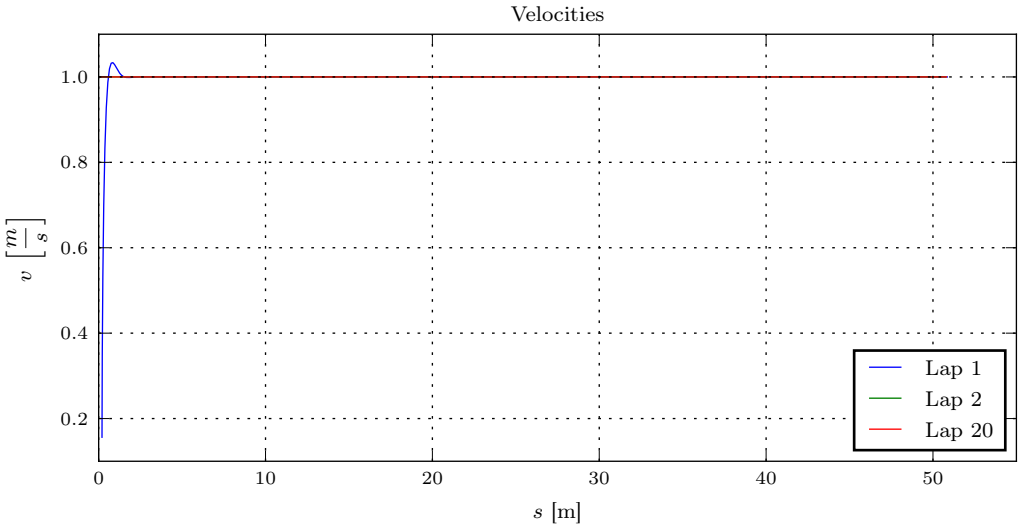

Figure 3.9: Velocity v, Kinematic Model

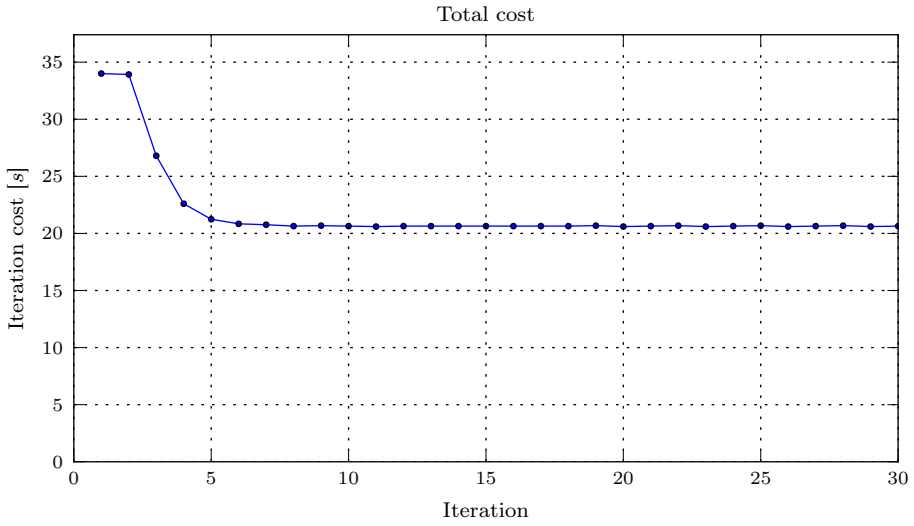

Figure 3.10: Iteration Cost, Dynamic Model

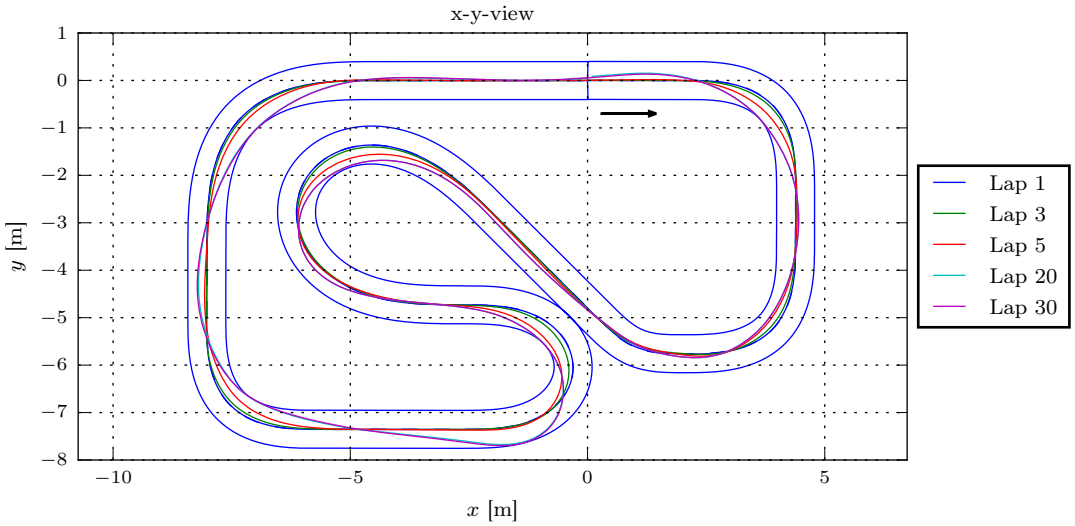

Figure 3.11: Top view, Dynamic Model

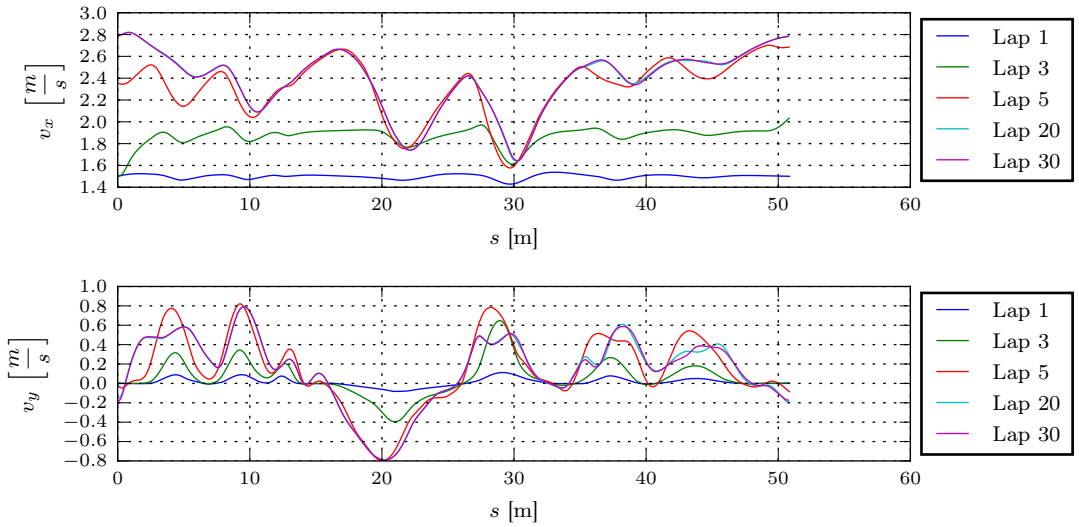

Figure 3.12: Velocities, Dynamic Model

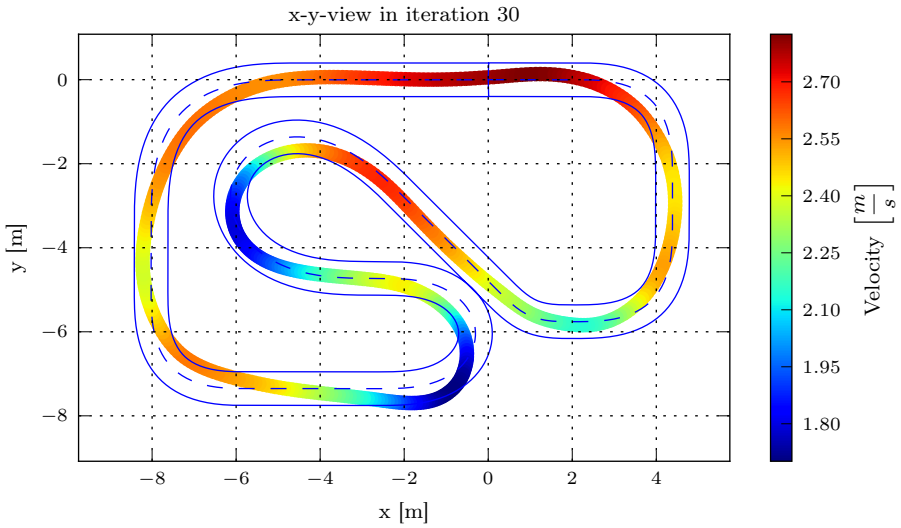

Figure 3.13: Velocity over x and y, Dynamic Model

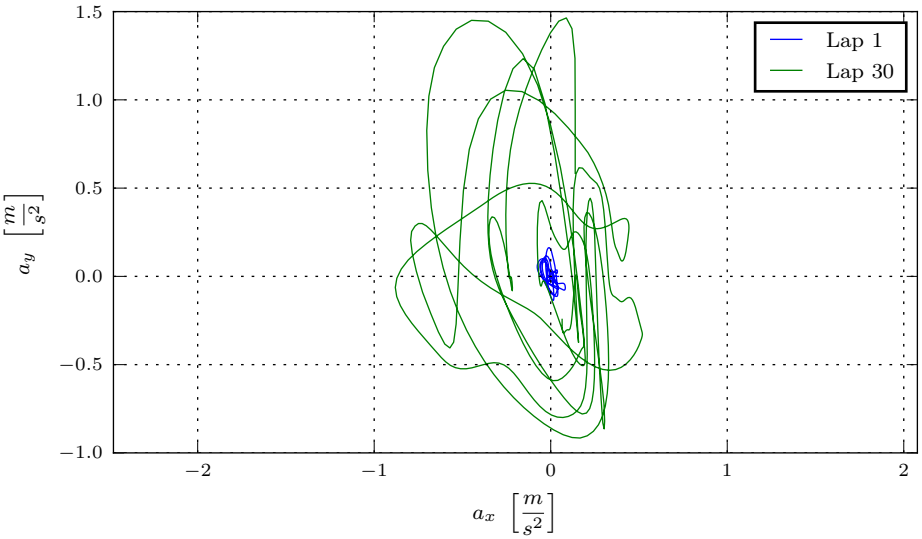

Figure 3.14: Friction circle, Dynamic Model

Chapter 4

Implementation

In this chapter the presented control strategy is implemented on a 1:10 scale remote-controlled car. The car is a commercially available hobby race car which has been modified to include sensors and onboard CPUs.
First, the car and its available sensors and actuators are introduced. We present the methods used to find a mapping from the optimal calculated commands to the actuator inputs. In section 4.2 a state estimation strategy based on Extended Kalman Filters and the projection onto a given race track are presented. Section 4.3 shows how we use Linear Regression to calculate system dynamics online. This allows adapting to varying dynamics (e.g. different tire behaviour in different parts of the track). Finally, section 4.4 presents the technical implementation and test results.

4.1 Introduction to the BARC

The car used for our experiments is a remote controlled race car of the model "Basher RZ-4 1/10 Rally Racer" that has been modified by the MPC Lab at UC Berkeley to easily test new control techniques. This car is called Berkeley Autonomous Race Car (BARC, [16]) and it has been used previously in student projects. The basic setup is shown in Figure 4.1.
Aside from standard actuators (brushless DC motor and steering servo), it features two onboard CPUs. The first CPU is an Arduino Nano which allows low-level control of the actuators as well as receiving and simple processing of sensor data. The second CPU is a Samsung Exynos 5422, provided by an Odroid XU4 single board computer. The Exynos CPU family has been used since 2010 on various smartphones like the Samsung Galaxy S. This processor is used for high-level computing (i.e. communication with USB devices and an external computer).

4.1.1 Sensors

For the purpose of race driving the estimation of absolute position, velocity and orientation are needed. The goal of the BARC project is to provide an affordable solution that can be used for testing autonomous driving algorithms. To accomplish this goal the following low-cost sensors were chosen:

IMU The Inertial Measurement Unit measures linear and angular accelerations as well as the car's current orientation in space. The IMU used on the BARC is a myAHRS+ which runs at a frequency of 50 Hertz. It is directly connected to the Odroid through a USB port.

Encoders Each wheel contains two magnetically operating encoders which are used to measure the car's velocity. These measurements are only reliable as long as the car is not drifting and as long as the wheels are spinning. The frequency of these measurements is directly related to the rotating speed of the wheels. The encoders are connected to the Arduino and send a

Figure 4.1: Basic BARC setup

signal at each half spin of the wheel. The Arduino sends the average speed resulting from the signals to the Odroid.

GPS To determine the absolute position of the car, an indoor positioning system from *Marvelmind robotics* [17] is used. Similar to the Global Positioning System (GPS), it determines the position of the car by triangulating the distances between stationary beacons around the racetrack and a mobile beacon which is fixed to the car. Distances are measured by the time of flight of ultrasound signals between all beacons.
The system measures the position with deviations of about 2 cm at a varying frequency between 10 and 16 Hertz.

4.1.2 Input mapping

While the MPC formulation calculates inputs of acceleration and steering in SI units, the actuators are controlled by the Arduino using pulse-width-modulation (PWM) signals. The PWM signals are processed by a servo library, which allows values in the range of 0...180 (i.e. turning angles of a servo). This section describes the identification procedure to find the steering mapping and acceleration mapping.

Steering mapping In order to identify the steering mapping, different steering PWM signals are sent to the steering servo while driving at a constant acceleration signal. Due to electromagnetic motor drag, this leads to a constant velocity while performing circles of different radii. A low velocity is used so that an ideal kinematic bicycle model can be assumed. Using the equations of the kinematic bicycle model, the steering angle δ_F can be inferred as follows:

$$\delta_F = \arctan\left(\frac{\dot{\psi} \cdot (L_F + L_R)}{v_x}\right) \tag{4.1}$$

It can be seen that only $\dot{\psi}$ and v_x need to be measured to calculate δ_F. Both quantities are measured with good accuracy by the IMU and encoders. Running this open loop test and approximating

the measurements with an affine function, following results were obtained:

$$u_{PWM} = 89.8 \cdot \delta_F + 90.8 \tag{4.2}$$

We could observe a maximum steering angle of 0.3 rad. This leads, in combination with the

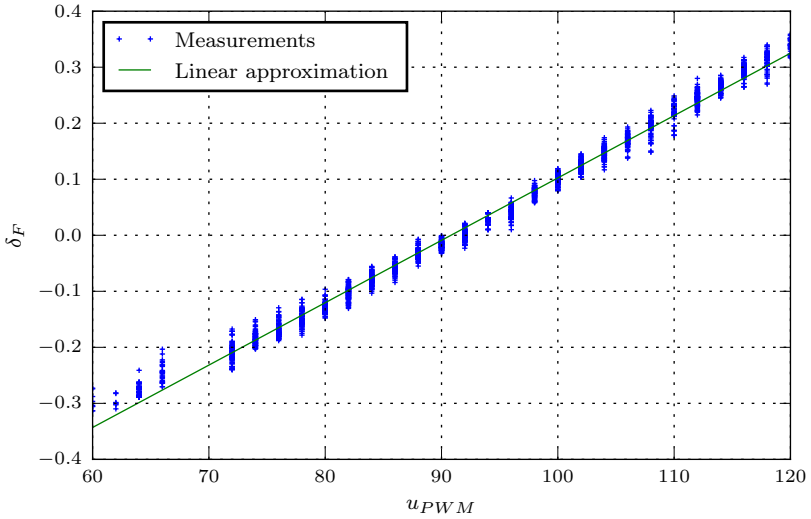

Figure 4.2: Steering mapping

wheelbase $L_R + L_F$, to a minimum possible turning radius of

$$R_{min} = \frac{L_R + L_F}{\tan(\delta_F)} = 0.81\,\text{m}, \tag{4.3}$$

which yields a maximum curvature of $c_{max} = 1.24\,\text{m}^{-1}$.

Steering delay Open loop experiments with fast changing steering inputs showed that the servo exhibits a short time delay of about 0.2 seconds. This delay can be seen in Fig. 4.3. Since the LMPC formulation uses a sample time of $dt = 0.1\,\text{s}$, we model this delay as a 2-step delay:

$$\begin{aligned} x_{k+1} &= f(x_k, a_k, \delta_{F,-2,k}) \\ \delta_{F,-2,k+1} &= \delta_{F,-1,k} \\ \delta_{F,-1,k+1} &= \delta_{F,k} \end{aligned} \tag{4.4}$$

Acceleration mapping The acceleration mapping consists of two different regimes: (positive) acceleration and breaking (negative acceleration). We found out that these two regimes can be described by two separate affine functions. Additionally, we have to find the motor drag which we model as a first order damping system in the longitudinal dynamics:

$$\dot{v} = a - c_M \cdot v \tag{4.5}$$

$$\text{with } a = \begin{cases} c_1^+ \cdot u_{PWM} + c_2^+ & \text{if } a > 0, \\ c_1^- \cdot u_{PWM} + c_2^- & \text{if } a < 0. \end{cases} \tag{4.6}$$

Adding the two linear mapping functions for acceleration and breaking we need to determine 5 parameters:

$$\dot{v} = c_1^+ \cdot u_{PWM} + c_2^+ - c_M \cdot v \;\forall \dot{v} > 0 \tag{4.7a}$$

$$\dot{v} = c_1^- \cdot u_{PWM} + c_2^- - c_M \cdot v \;\forall \dot{v} < 0 \tag{4.7b}$$

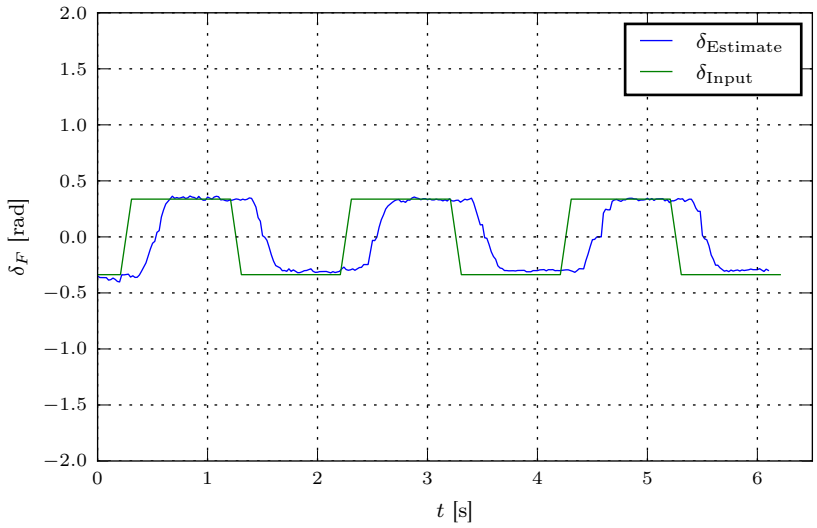

Figure 4.3: Steering delay

To identify the coefficients of these two regimes, two groups of open loop tests were performed: Firstly, different (positive) accelerations were sent to the motor for a fixed interval of 4 seconds before the car was stopped. Secondly, the car was accelerated and then decelerated by sending different (negative) accelerations.
The resulting correlations can be approximated by two affine functions, where one describes the positive and the other one describes the negative acceleration region.
The results of the coefficients are listed in Table 4.1.

Table 4.1: Pacejka coefficients for different road conditions

Coefficient	Value
c_1^+	0.162
c_2^+	-14.7
c_1^-	0.0149
c_2^-	-1.47
c_M	0.5

Both the linear functions and the measured accelerations are illustrated in Fig. 4.4.

4.2 State estimation

This section presents the sensor fusion and state estimation. A good state estimation should filter out the sensor noise and drifts. A common approach would use a Kalman filter with the system's model and a covariance matrix to predict and determine the most probable current state estimate. However, since the previously mentioned system identification assumes no prior model knowledge, we cannot use a given model for state estimation.
Therefore, integrating and smoothing functions which do not account for system dynamics would be another approach to measure current state estimates. However, this approach would not be able to reliably predict the heading ψ since its sensor measurement is distorted by its drift.
Various different techniques have been explored in the course of this thesis and following technique proved to be working best. It combines two Extended Kalman filters, one of which assumes a

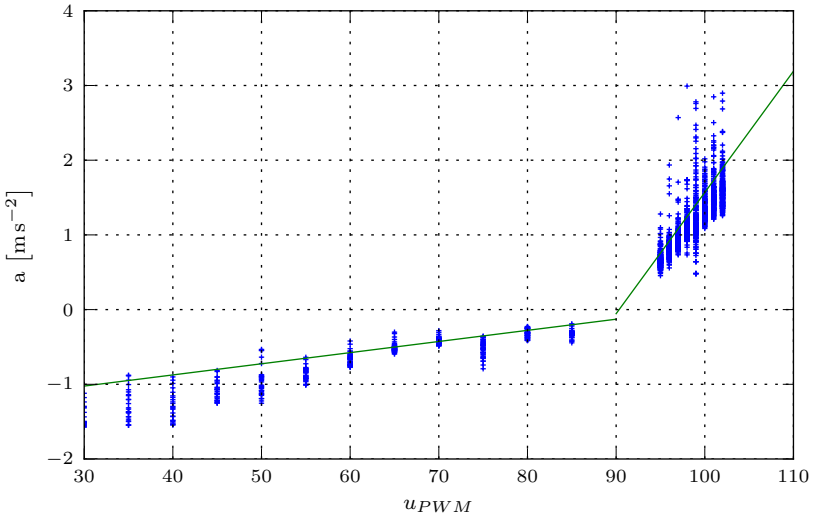

Figure 4.4: Acceleration mapping

kinematic bicycle model whereas the other one follows a pure kinematic point mass approach (denoted as *kinematic model*). The filter model is illustrated in Fig 4.5.

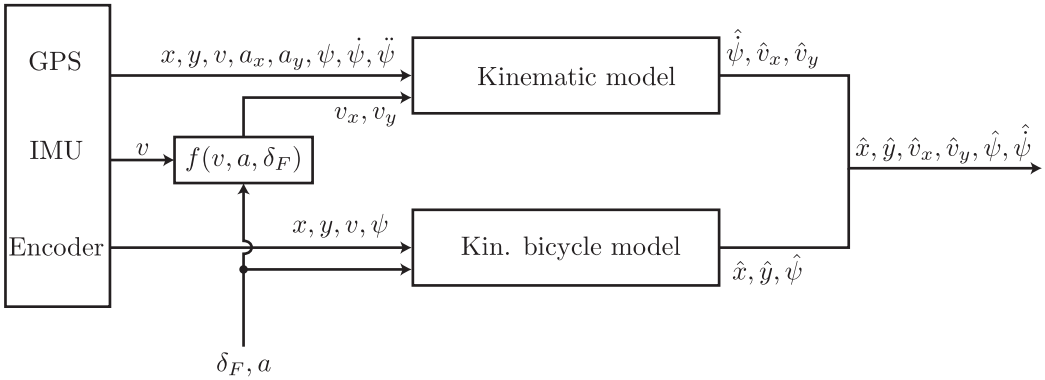

Figure 4.5: Filter model

The kinematic model combines measurements of acceleration, velocity, and position by integrating (see sec. 4.2.2).
This filtering approach is based on the assumption that only the kinematic bicycle model is able to estimate the current heading angle $\hat{\psi}$ using measurements of coordinates and velocity, while the point mass model would not be able to account for the drift in the ψ measurement. This is why the kinematic bicycle model is used to calculate the heading angle as well as the inertial coordinates. On the other hand, the kinematic model is well suited to combine measurements of accelerations, velocities, and position. This is why we use this model for the estimation of the yaw rate and velocities.

4.2.1 Extended Kalman Filter

Kalman filters provide a general tool to calculate a state estimate using noisy and/or biased sensor measurements assuming a gaussian model for the noise. The standard Kalman filter uses a linear model of the system in order to calculate the most probable state from a set of different state

measurements.
Following general procedure has been taken from [18]. The Extended Kalman filter is based on the same principles as the standard Kalman filter but it allows to use a nonlinear model by linearizing the model at each time step.
The state estimation always consists of two steps: The *Prediction step* and the *Update step*. The first step calculates a prediction of the next state $\hat{\boldsymbol{x}}_{k|k-1}$ based on the previous state estimate $\hat{\boldsymbol{x}}_{k-1|k-1}$ and the control input $\boldsymbol{u}_{k-1}$. The update step corrects this prediction with the new measurement $\boldsymbol{z}_k$. The final state estimate is calculated using the Kalman gain $\boldsymbol{K}_k$. Both the model and measurement equations are linearized in each step.
Prediction:

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}) \tag{4.8a}$$
$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1} \tag{4.8b}$$

Update:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{4.9a}$$
$$\boldsymbol{S}_k = \boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^T + \boldsymbol{R}_k \tag{4.9b}$$
$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^T\boldsymbol{S}_k^{-1} \tag{4.9c}$$
$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k \tag{4.9d}$$
$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1} \tag{4.9e}$$

The predicted and updated estimate covariance $\boldsymbol{P}_{k|k-1}$ and $\boldsymbol{P}_{k|k}$, respectively, is a measure for the uncertainty of the state estimate. The process noise matrix $\boldsymbol{Q}$ is a measure for the uncertainty of the model, it is usually determined in experiments. The measurement noise matrix $\boldsymbol{R}$ depends on noise values of the sensors. They are often provided in sensor data sheets or can otherwise be measured in experiments. The state transition and observation matrices $\boldsymbol{F}_k$ and $\boldsymbol{H}_k$ are the functions $f(\boldsymbol{x}, \boldsymbol{u})$ and $h(\boldsymbol{x})$, linearized at the current state estimate $\hat{\boldsymbol{x}}_{(\cdot)}$ and input $\boldsymbol{u}_k$:

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k}, \boldsymbol{u}_k} \tag{4.10a}$$
$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{4.10b}$$

4.2.2 Filter model

Kinematic bicycle model The first filter features a kinematic bicycle model, Eq. (4.11) shows its model equations:

$$\begin{aligned}
\dot{x} &= v \cdot \cos(\psi + \beta) \\
\dot{y} &= v \cdot \sin(\psi + \beta) \\
\dot{\psi} &= \frac{v}{L_R} \cdot \sin(\beta) \\
\dot{v} &= a - c_M \cdot v \\
\dot{d}_\psi &= 0
\end{aligned} \tag{4.11}$$

with heading drift d_ψ. There are two physical parameters, the first one is the distance L_R between the rear axle and the GPS sensor which is measured as $L_R = 0.125\,\text{m}$. The second is the motor drag c_M which was estimated in section 4.1.2 as $c_M = 0.5$.
The measurement vector contains GPS, Encoder and heading measurements:

$$\boldsymbol{z} = \begin{pmatrix} x_m & y_m & \psi_m & v_m \end{pmatrix}^T, \tag{4.12}$$

where the subscript m denotes measured values. This leads to following measurement model:

$$h(\hat{\boldsymbol{x}}_{k|k-1}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \hat{\boldsymbol{x}}_{k|k-1} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{\psi} + \hat{d}_\psi \\ \hat{v} \end{pmatrix} \tag{4.13}$$

Kinematic model The second filter features a pure kinematic motion model from [19] which integrates accelerations to find velocities and positions. Eq. (4.14) shows its model equations:

$$\begin{aligned} \dot{x} &= \cos(\psi) v_x - \sin(\psi) v_y \\ \dot{y} &= \sin(\psi) v_x + \cos(\psi) v_y \\ \dot{v}_x &= a_x + \dot{\psi} v_y \\ \dot{v}_y &= a_y - \dot{\psi} v_x \\ \dot{\psi} &= \dot{\psi} \\ \ddot{\psi} &= 0 \\ \dot{a}_x &= 0 \\ \dot{a}_y &= 0 \\ \dot{d}_\psi &= 0 \end{aligned} \tag{4.14}$$

It uses the longitudinal and angular acceleration measurements of the IMU and position measurements of the GPS. Since the car's suspension leads to strong tilting motions in curves, the measurements of a_y are distorted significantly. In order to support a correct estimation of v_y and to reduce the effect of drift errors in a_y, the system additionally receives artificial measurements of v_x and v_y. These are constructed by following relations:

$$\begin{aligned} v_{x,m} &= v_m \cos(\delta_F) \\ v_{y,m} &= v_m \sin(\delta_F) \end{aligned}$$

where v_m is the encoder measurement and δ_F the control input. Thus, the measurement vector of the second filter uses following measurements:

$$\boldsymbol{z} = \begin{pmatrix} x_m & y_m & v_{x,m} & v_{y,m} & \psi_m & \dot{\psi}_m & a_{x,m} & a_{y,m} \end{pmatrix}^T \tag{4.15}$$

The measurement function $h(\hat{\boldsymbol{x}})$ is derived as follows:

$$h(\hat{\boldsymbol{x}}_{k|k-1}) = \begin{pmatrix} \hat{x} & \hat{y} & \hat{v}_x & \hat{v}_y & \hat{\psi} + \hat{d}_\psi & \hat{\dot{\psi}} & \hat{a}_x & \hat{a}_y \end{pmatrix}^T \tag{4.16}$$

Tuning of Q and R matrices The measurement noise matrices were constructed assuming diagonal matrices (independent sensors) and measuring the sensor noises during operation.
The process noise matrix $\boldsymbol{Q}$ was tuned manually in post-processing. The values were tuned with the objective to provide a smooth fit to the raw measurements.

Frequency of the filter The extended Kalman filter runs at a frequency of 50 Hertz and assumes that all measurements are received at the same time and with equal frequencies. However, since we are using different sensors at different frequencies, we are using following strategy for each sensor:

IMU The IMU provides data at the frequency of the Kalman filter itself, 50 Hz. This is why we can use this data directly.

GPS The indoor GPS provides measurements at varying frequencies between 10 and 16 Hertz, whereas there are times when no measurement are received for up to one second. These interruptions might result from hardware issues (e.g. interfering reflections of ultrasound, timing issues) and could not be resolved. In order to account for this uncertainty, we always extrapolate the previously received data by polynomials to the timestamp of the Kalman filter update. This extrapolation makes sure that even longer measurement breaks don't pose any difficulties to the estimator.

Encoder Three of the car's wheels are equipped with encoders, using 2 magnets each. The frequency of signals from these encoders can be calculated as follows:

$$f = \frac{1}{T} = \frac{v}{\pi r} \tag{4.17}$$

with the velocity v and the wheel radius r. This means that at a velocity of $1.0\,\mathrm{m\,s^{-1}}$ and for a wheel radius of 3.6 cm, we receive a velocity measurement from each wheel at a rate of 9 Hz. These measurements are not synchronized as the wheels start at different angles. All measurements are received by the Arduino, averaged, and then sent to the filter at a constant rate of 50 Hz. Since even with three encoders we can't reach an update rate of 50 Hz, we assume a rather high encoder measurement noise in the measurement noise matrix $\boldsymbol{R}$ to account for equal consecutive measurements.

4.2.3 Mapping to the track reference frame

The estimator discussed above returns a state estimate in the inertial frame which then needs to be transformed into the track reference frame. The coordinates in the track reference frame are the curvilinear abscissa s, the lateral position error e_Y, and the heading error e_ψ. The procedure for this transformation is described below.
We assume that the track is given by n equidistant vertices with coordinates $\boldsymbol{x}_i = \begin{pmatrix} x_i & y_i \end{pmatrix}, i = 1...n$.

1. Find the closest vertex $\boldsymbol{x}_i$ of the track to the current position of the car.
2. Select a set of vertices around vertex i.
3. Use this set to approximate the track locally using a polynomial, returning functions for the coordinates $(x, y) = f(s)$ and curvature $c = g(s)$.
4. Use these functions to calculate the current curvilinear abscissa s, lateral position e_Y and heading error e_ψ.

The number of points used for the approximation and the polynomial degree have to be chosen in such a way to provide an accurate fit throughout the entire prediction horizon.
In practice, we used a polynomial of 8^{th} degree. This polynomial proved to approximate the piecewise linear curvature well enough while keeping the complexity of the optimization problem at a reasonable level.

4.3 System Identification

In order to provide an accurate model to the MPC, the system's dynamics are identified at each time step. The dynamic bicycle model from Eq. (3.3) is used to derive a Linear Regression model that uses functions of the system's states as features. State estimates from previous laps are then used to calculate current parameters $\theta_{i,j}$.

Linear Regression model of the dynamic bicycle model:

$$v_{x,k+1} - v_{x,k} = \theta_{x,1} \cdot v_{y,k}\dot{\psi}_k + \theta_{x,2} \cdot v_{x,k} + \theta_{x,3} \cdot a_k \tag{4.18a}$$

$$v_{y,k+1} - v_{y,k} = \theta_{y,1} \cdot \frac{v_{y,k}}{v_{x,k}} + \theta_{y,2} \cdot v_{x,k}\dot{\psi}_k + \theta_{y,3} \cdot \frac{\dot{\psi}_k}{v_{x,k}} + \theta_{y,4} \cdot \delta_k \tag{4.18b}$$

$$\dot{\psi}_{k+1} - \dot{\psi}_k = \theta_{\psi,1} \cdot \frac{\dot{\psi}_k}{v_{x,k}} + \theta_{\psi,2} \cdot \frac{v_{y,k}}{v_{x,k}} + \theta_{\psi,3} \cdot \delta_k \tag{4.18c}$$

with states v_x, v_y, $\dot{\psi}$ and parameters $\theta_{i,j}$. Note that we added a term in (4.18a) to account for motor drag and to match it with Eq. (4.5). The features $\boldsymbol{X}$ of this Linear Regression problem are nonlinear functions of the states and inputs. Using multiple samples n, each equation can be rewritten as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}. \tag{4.19}$$

Equation (4.20) shows the equations for the LR of v_x:

$$\boldsymbol{y} = \begin{pmatrix} v_{x,2} - v_{x,1} \\ v_{x,3} - v_{x,2} \\ \vdots \\ v_{x,n} - v_{x,n-1} \end{pmatrix} \qquad \boldsymbol{X} = \begin{pmatrix} v_{y,1}\dot{\psi}_1 & v_{x,1} & a_1 \\ v_{y,2}\dot{\psi}_2 & v_{x,2} & a_2 \\ \vdots & \vdots & \vdots \\ v_{y,n-1}\dot{\psi}_{n-1} & v_{x,n-1} & a_{n-1} \end{pmatrix} \qquad \boldsymbol{\theta} = \begin{pmatrix} \theta_{x,1} \\ \theta_{x,2} \\ \theta_{x,3} \end{pmatrix} \tag{4.20}$$

The LR equations for v_y and $\dot{\psi}$ can be written accordingly.
To determine the optimal coefficients $\boldsymbol{\theta}^*$, following minimization has to be solved:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2 \tag{4.21}$$

There are different approaches to solve this problem, we use the *normal equation* method. This leads to following analytic solution:

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{4.22}$$

Since we are using about $n = 100...1000$ samples, the calculation of the inverse of $\boldsymbol{X}^T\boldsymbol{X}$ is still sufficiently fast. However, if we were to use even more samples, other methods such as *gradient descent* might work faster.

4.4 Implementation

The estimator and MPC solver are implemented in the ROS framework (Robot Operating System, [20]). The sensor data is sent from the BARC to a more powerful CPU (Macbook Pro 2010, 2.4 GHz) which solves the MPC problem at a rate of 10 Hertz and sends the optimal input to the BARC. The setup is illustrated in Fig. 4.6. The ROS framework manages the execution of all scripts and makes sure that they are running at their specified frequencies. It also manages the wireless communication between the Odroid and the computer.
The estimator is a python script that runs at a constant frequency of 50 Hz.
The MPC optimization problem is solved by Ipopt [21] which is executed by a script written in Julia 0.4.6 [22]. It solves the LMPC problem at a constant frequency of 10 Hz and sends the optimal control input to the BARC. At the same time, it receives and saves all state estimates from the estimator.
Since we need to maintain a constant input rate of 10 Hz while the optimization process can last different times (at most 0.1 s), the commands are always sent at the beginning of one time step. The MPC solver uses an initial state that is predicted by 0.1 seconds from the most recent estimated state. Figure 4.7 shows the timing diagram of ROS nodes and messages.
Below all steps are described that are performed during one time step (asynchronously). Additionally, Figure 4.8 illustrates the closed-loop system.

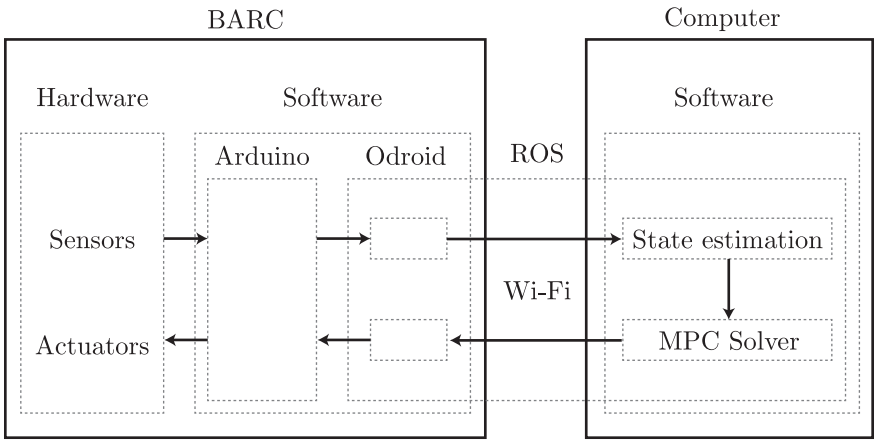

Figure 4.6: Software/Hardware setup

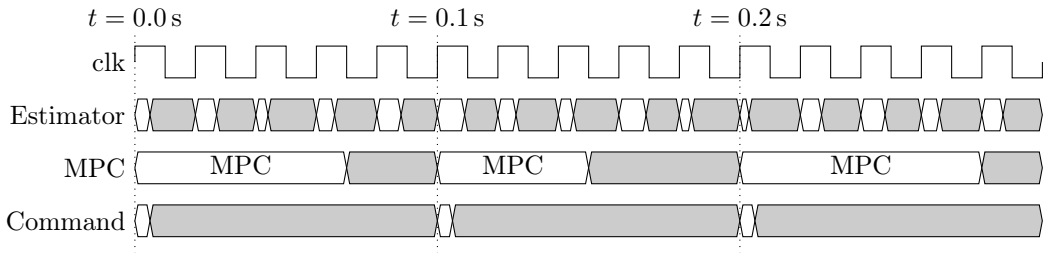

Figure 4.7: Timing diagram

1. State estimation
   (a) Read sensor data
   (b) Calculate the current state in the inertial frame
   (c) Transform the state to the track frame
   (d) Calculate polynomial coefficients that approximate the curvature locally
2. System Identification
   (a) Select data from previous and current lap
   (b) Calculate Matrices of features
   (c) Calculate parameters of system dynamics using Linear Regression
3. Approximate safe set and cost function
   (a) Select locally near data from previous laps
   (b) Calculate polynomials that approximate this data to construct the safe set
4. Solve the LMPC problem
   (a) Use the state estimate as initial state for the MPC formulation
   (b) Minimize the cost function, using state dynamics from the Linear Regression and curvature as well as the safe set and terminal cost function
   (c) Send the optimal input to the actuators

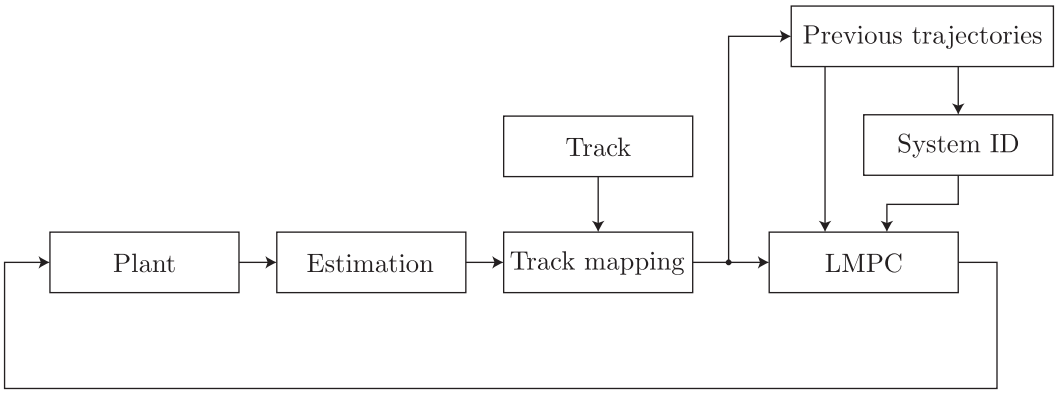

Figure 4.8: Control structure

4.5 Experimental results

The experiments were performed in a closed space at UC Berkeley. The available area had a size of approx. 5 by 8 meters and provided enough space for a simple race track.
We chose a race track that fits the available area while still providing curves in different directions with the maximum possible curvature of $1.24\,\mathrm{m}^{-1}$. The curvature is shown in Fig. 4.9. The track length is 19 m. We used a constant sample size for the system identification of 120 points per lap which corresponds to 2.4 s since the state estimation runs at 50 Hz. We also used an LMPC horizon length of $N = 12$ which corresponds to a prediction time of 1.2 s. This horizon is a trade-off between fast learning and fast solver times.

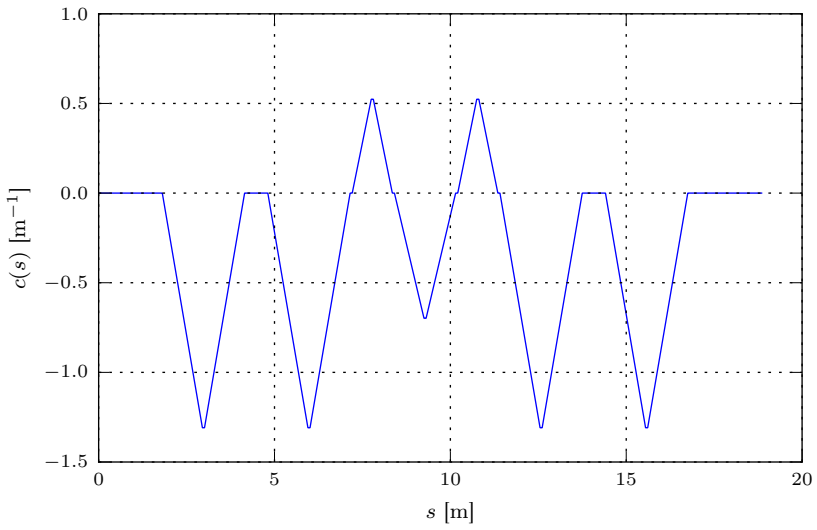

Figure 4.9: Curvature of the experimental race track

The first 3 laps were controlled by a a path following MPC with a kinematic bicycle model at $v_{Ref} = 1.0\,\mathrm{m\,s}^{-1}$, this reference speed yields a path following lap time of approx. $t_{pf} = 21\,\mathrm{s}$. These 3 laps are required to create a safe set containing two full laps and to provide enough data for the system identification. The LMPC strategy starts in the 4^{th} lap.

Results and discussion We can see in Fig. 4.10 that the lap time decreases rapidly within the first few learning iterations before it converges to a trajectory that takes approx. $t = 7.55\,\mathrm{s} \pm 0.25\,\mathrm{s}$

at an average speed of $2.5\,\mathrm{m\,s^{-1}}$. The iteration cost does not continuously decrease but instead, it oscillates around a mean value. This is due to the model mismatch between the MPC formulation and the real car dynamics. The model mismatch arises from inaccurate measurements that are used for the system identification and from the polynomial approximation of the curvature. Robust LMPC and how to handle model mismatch is subject of current research.
Figures 4.11 and 4.12 illustrate the converging behavior of velocities and lateral deviation. Even though especially lateral velocity measurements are very noisy, the combination of system identification and LMPC manages to converge to an optimal trajectory. Figure 4.13 visualizes the car's velocity at each point of the track during one optimal iteration.
We found out that, in order to ensure feasibility and to avoid numerical issues, the lane width w_{Lane} in Eq. (3.20) is a tuning parameter and had to be chosen about 5 cm smaller than the actual lane width. That way, this soft constraint can be violated by a small amount while the car still stays on the track.
It can be seen in Fig. 4.12 that, even in steady state, the system still exhibits some oscillations (compare laps 25 and 29). This is due to many uncertainties in the entire system. On one hand, these result from rare WiFi communication problems in which the car does not receive new commands for a split second. On the other hand, sometimes the MPC solver cannot find an optimal solution within the time frame of 0.1 s. In this case sub-optimal commands are used.
Figure 4.14 illustrates the predicted trajectory at one timestep in lap 6. At this point, the safe set consists of the two previous laps 4 and 5. The terminal predicted state lies within the boundaries of the safe set - as it is enforced by the terminal constraints.
Figure 4.15 shows the evolution of system coefficients during one lap in iteration 20. Since the velocity has only little variance, the car dynamics and its coefficients stay nearly constant. The two coefficients $c_{v_x,1}$ and $c_{v_y,2}$ are zero, because their related features are too small to find any correlations (refer to Eq. (4.18)). The remaining coefficients can be used to calculate unknown parameters:

Table 4.2: BARC physical parameters

Physical meaning	Coefficient	Value
Cornering stiffness front tires	c_{α_F}	$7.9\,\mathrm{N\,rad^{-1}}$
Cornering stiffness rear tires	c_{α_R}	$6.1\,\mathrm{N\,rad^{-1}}$
Moment of inertia	I_z	$0.025\,\mathrm{kg\,m^2}$

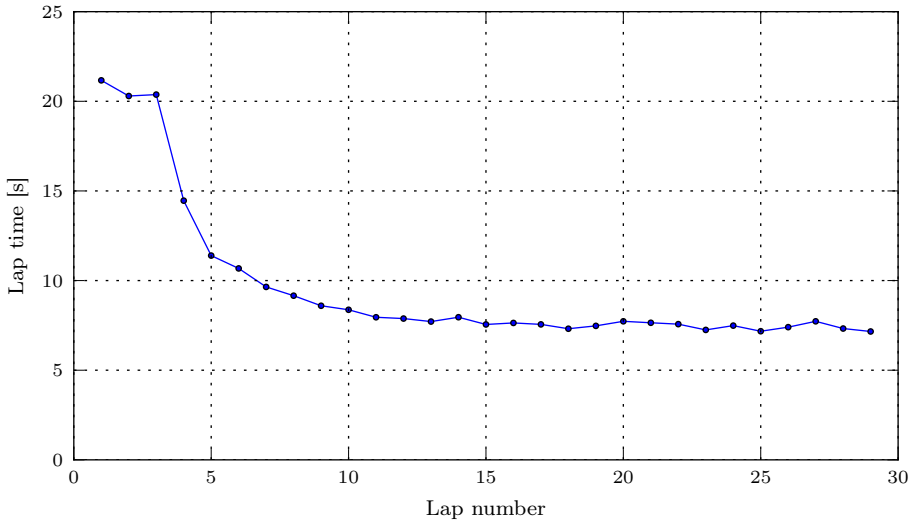

Figure 4.10: Lap times

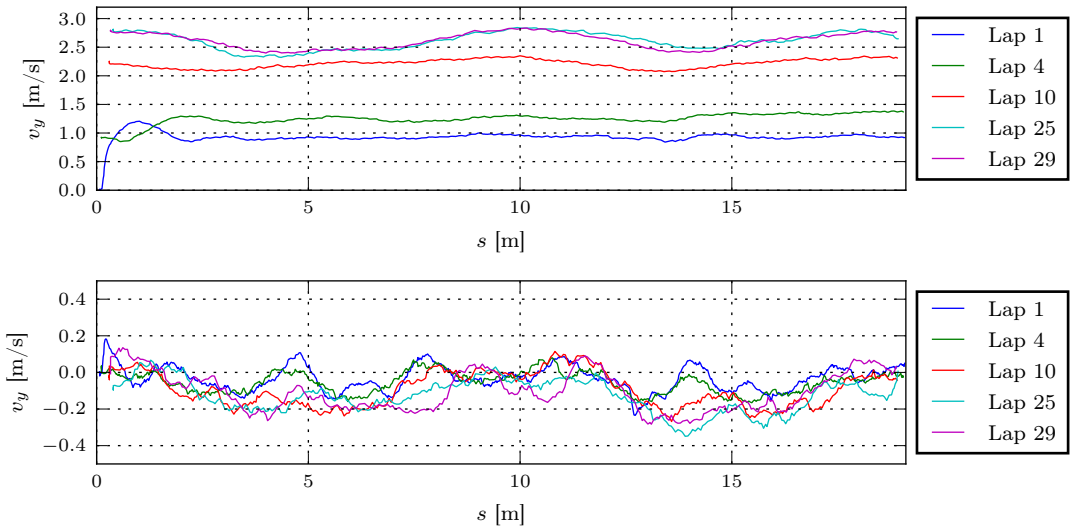

Figure 4.11: Longitudinal and lateral velocity

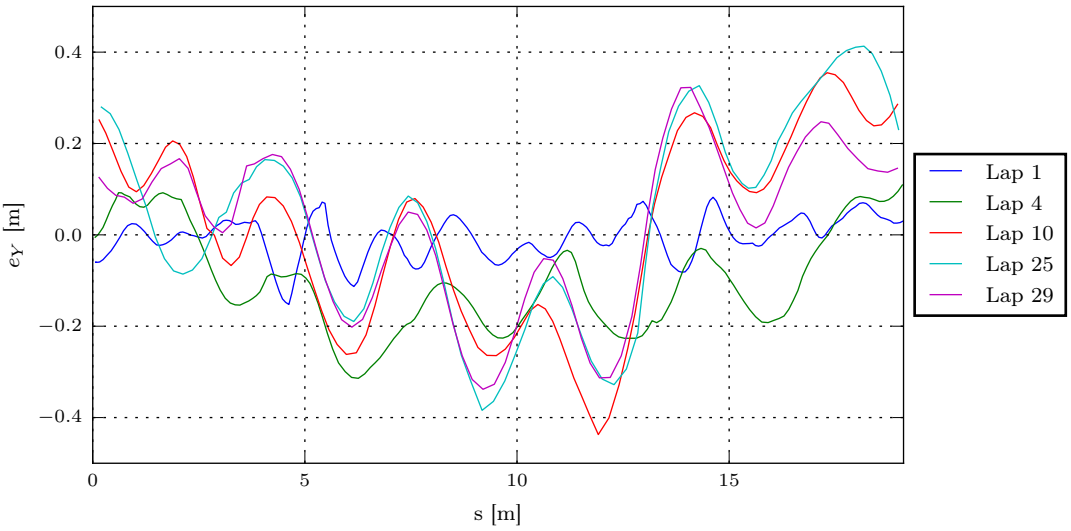

Figure 4.12: Lateral deviation

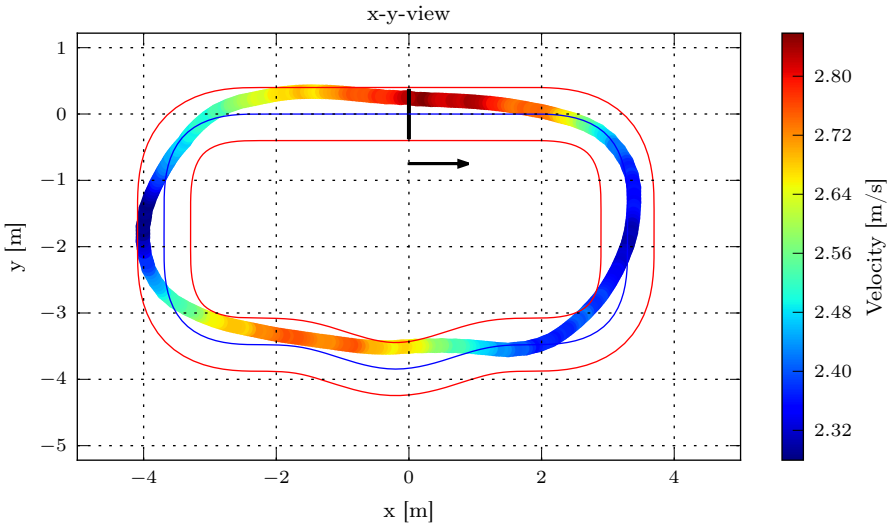

Figure 4.13: Velocity over x,y

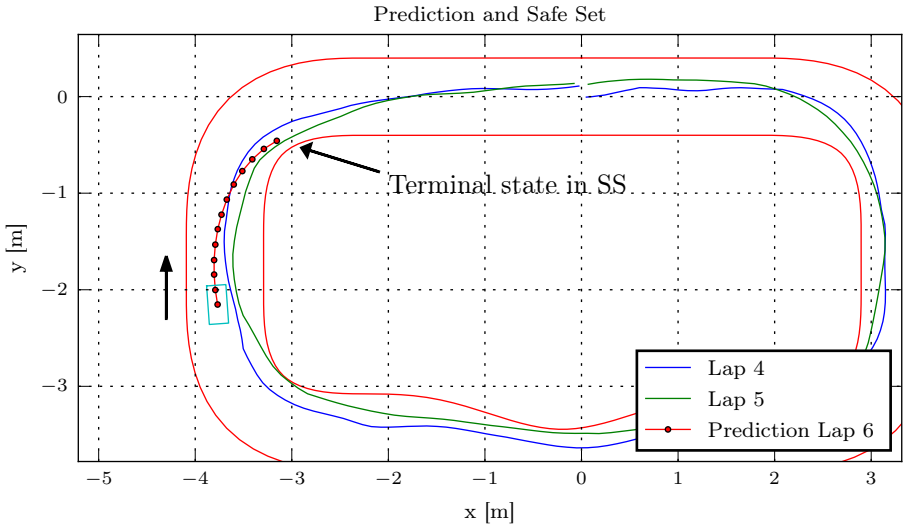

Figure 4.14: Prediction and safe set

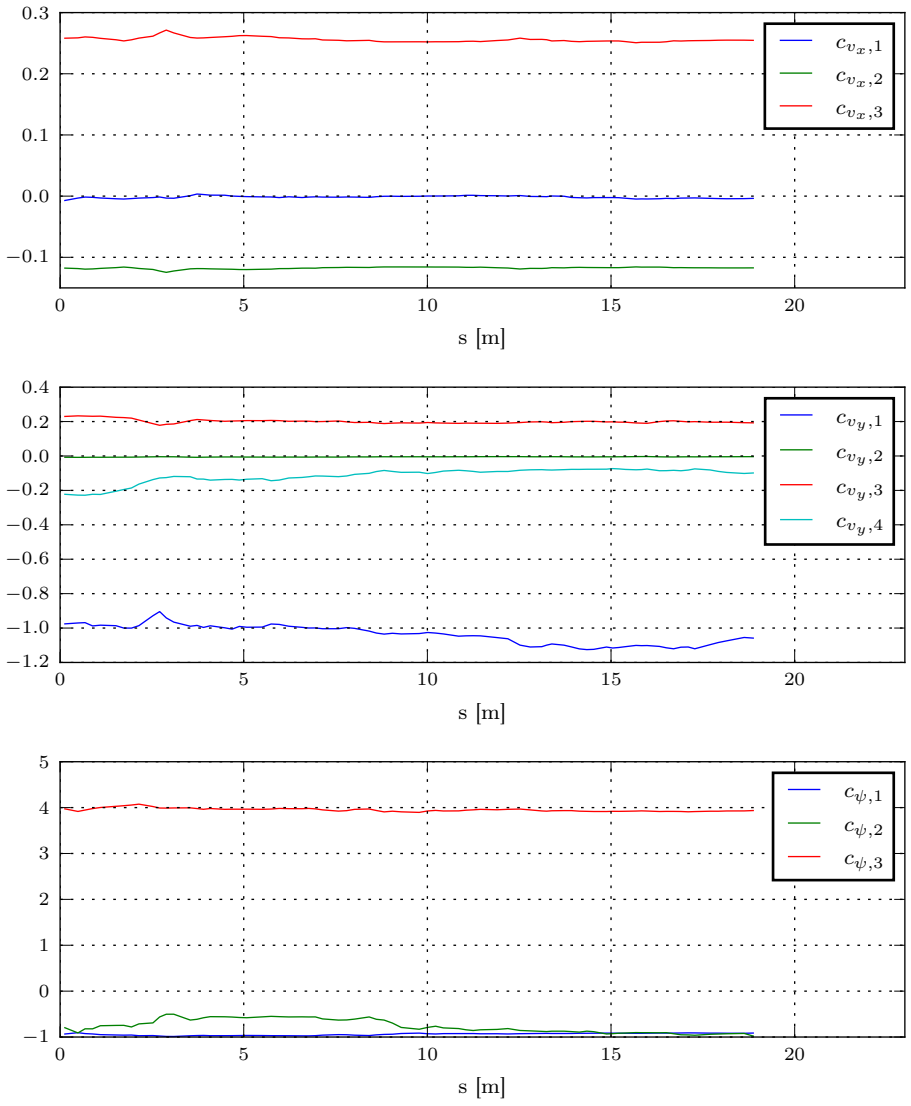

Figure 4.15: System identification coefficients, it. 20

Chapter 5

Conclusion and Outlook

5.1 Conclusion

The goal of this project was to extend the existing LMPC strategy to repetitive tasks and test this theory on the race driving problem.
We were able to show that - by only little extensions - the LMPC strategy is also applicable to continuous repetitive systems. The resulting controller is feasible and stable. In our application of race driving we could also show that the iteration cost is non-increasing. Simulations and experiments on a real car showed that it is real time feasible and converges to a locally optimal solution. Our control strategy does not need any offline computations beforehand and works in real time - even for a complex system with 6 states and long prediction horizons of 30 time steps.
We developed an experimental setup using a 1:10 scale remote-controlled race car (BARC) that is easy to transport and set up at different locations.
We presented an online system identification method that allowed us to optimize the trajectory without prior knowledge of the system's parameters. And even though we were using low-cost sensors we were able to develop an estimator that provides reliable state estimates of sufficient quality for race driving.
In experiments we reached velocities of up to $3\,\mathrm{m\,s^{-1}}$ on a racetrack with maximum curvatures of $1.2\,\mathrm{m^{-1}}$ which corresponds - according to the scaling factor - to maximum velocities of $100\,\mathrm{km\,h^{-1}}$ and a minimum curve radius of $8\,\mathrm{m}$.
A strong advantage of using the LMPC strategy in combination with online system identification is that the model can slowly be learned more and more accurate from iteration to iteration while decreasing the iteration cost at the same time. This makes it possible to optimize a repetitive task without prior knowledge of the system parameters. However, the system's dynamics have to be known in order to construct a linear regression model.
Certain assumptions and approximations still remain a challenge. Using polynomials to approximate the curvature and the safe set did not allow to use longer time horizons, as the polynomials would have to be more complex to fit these functions. Eventually, a tradeoff between the MPC horizon length, the complexity of the polynomials (mainly determined by their degrees), and the solving time of the solver has to be found.

5.2 Outlook and future work

While we were assuming no model mismatch in simulations of the LMPC strategy, we clearly had some model mismatch in real world experiments. How to handle this model mismatch and what assumptions and assertions can still be made in this scenario is subject of current research.
Also, so far the LMPC solution converges to a local optimum but we don't know a method to find a global optimum. Developing a systematic approach that finds a global optimum (e.g. by artificially disturbing the solution trajectory) is a topic that could improve this method.

Another idea that can lead to faster convergence to the optimal solution is to use symmetries of state dynamics within iterations. As in the car racing example, this would lead to similar treatment of similar track sections (e.g. deceleration before, acceleration in curves).
This work already made some simplifications to the LMPC problem (e.g. relaxing the terminal set, approximating the curvature by polynomials). However, the resulting LMPC is still a complex nonlinear problem which remains a challenge to solve efficiently and in real time. Further simplifications or transformations of this problem might allow even shorter solution times and therefore faster reaction times.
In order to improve lap times of the BARC experimental setup even more and to allow higher velocities, the estimation system would need to be improved. One major difficulty proved to be the proper estimation of the car's heading angle. A very promising method would be to use a stationary camera system (e.g. from [6]). Using markers on different points on the car would allow measuring not only the current position of a car but also its heading angle. Also using an onboard camera that detects lane boundaries or even the entire surroundings (e.g. Simultaneous Localization and Mapping, SLAM) might be used in the estimation, although this may be computationally expensive.

Bibliography

[1] IFR. Executive Summary of World Robotics 2016 Service Robots. page 8, 2016.

[2] Jay H. Lee and Kwang S. Lee. Iterative learning control applied to batch processes: An overview. *Control Engineering Practice*, 15(10 SPEC. ISS.):1306–1318, 2007.

[3] DA D.a. Bristow, Marina Tharayil, and AG a.G. Alleyne. A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(June):96–114, 2006.

[4] Ugo Rosolia and Francesco Borrelli. Learning Model Predictive Control for Iterative Tasks. (4), 2016.

[5] R. S. Sharp and Huei Peng. Vehicle dynamics applications of optimal control theory. *Vehicle System Dynamics*, 49(7):1073–1111, 2011.

[6] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1 : 43 scale RC cars. (July 2014):628–647, 2015.

[7] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. An Efficient Minimum-Time Trajectory Generation Strategy for Two-Track Car Vehicles. 23(4):1–15, 2015.

[8] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[9] Francesco Borrelli. Constrained Optimal Control of Linear and Hybrid Systems. 2003.

[10] Youqing Wang, Furong Gao, and Francis J. Doyle. Survey on iterative learning control, repetitive control, and run-to-run control. *Journal of Process Control*, 19(10):1589–1600, 2009.

[11] R. Rajamani. *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer US, 2005.

[12] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. *Proc. IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, 2015.

[13] Egbert Bakker, Lars Nyborg, and Hans B. Pacejka. Tyre modelling for use in vehicle dynamics studies. In *SAE Technical Paper*. SAE International, 02 1987.

[14] Alain Micaelli and Claude Samson. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. 2006.

[15] Ugo Rosolia, Ashwin Carvalho, and Francesco Borrelli. Autonomous Racing using Learning Model Predictive Control.

[16] Berkeley autonomous race car. http://www.barc-project.com, 01 2017.

[17] Marvelmind robotics. http://www.marvelmind.com, 01 2017.

[18] Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and X.-Rong Li. *Estimation with Applications to Tracking and Navigation.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[19] Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. GPS/IMU data fusion using multisensor Kalman ltering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 2006.

[20] Robot operating system. http://www.ros.org, 01 2017.

[21] Ipopt - interior point optimizer. https://projects.coin-or.org/Ipopt, 01 2017.

[22] Julia. http://julialang.org, 01 2017.