ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Berkeley
UNIVERSITY OF CALIFORNIA

Maximilian Brunner

Repetitive Learning MPC and its application on 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

January 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 strategy is extended to repetitive systems. The presented controller is able to improve its performance with every iteration by learning from previous iterations. It is applied to an autonomous race driving problem. Simulations on a kinematic and a dynamic bicycle model show the controller's performance. Additionally, the strategy is applied to a 1:10 scale RC car. An estimator using an indoor ultrasound based GPS as well as an IMU and wheel encoders is developed. The system's dynamics are identified online using Linear Regression. Experiments are performed on a small race track, reaching maximum velocities of $3\,\mathrm{m\,s^{-1}}$.

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.

- 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
- $J^{\text{LMPC},j}_{t\to t+N}(\boldsymbol{x}_t^j)$: optimal prediction cost at x_t^j

- $h(\boldsymbol{x}, \boldsymbol{u})$: 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: $L_F = 0.125\,\mathrm{m}$
- 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 the trajectory that optimizes the given process. Previous research like Iterative Learning Control (ILC, [2]) or Learning Model Predictive Control (LMPC, [3]) has already focused on improving trajectories over many iterations.
As opposed to ILC, the theory of LMPC allows finding an optimal, reference-free trajectory, given a general optimization function. However, so far it has only been tested for batch processes, which start every iteration at the same initial state.
We are now interested in extending this theory to continuous repetitive tasks which are characterized by a smooth transition between iterations, abolishing the need for same 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

We would like to develop an efficient method that finds an optimal trajectory in a continuous repetitive system. We would like 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. We would also like to use an online system identification strategy that allows us to account for undetected or unknown system dynamics. 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, most of them try to find a time optimum by computationally expensive offline optimization or sophisticated online methods like MPCC [4]. This thesis is organized as follows: ...

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) is presented which is then extended to repetitive tasks.

2.1 Model Predictive Control

Model Predictive Control is a concept that has been developed since the late 1970s, starting from Chemical Process Industries [5]. Since then, increasing computational power which is essential to quickly solving complex optimization problems has helped further research on this method and has extended it to other fields beyond chemical industries. This section gives a brief overview on the mathematical concepts of MPC [6].
In general, an MPC controller predicts the state dynamics over a finite time horizon, given a system model and control inputs. An objective function is minimized to find the optimal control input which is then applied to the system.
Assume that a system can be described by its discrete time state equations

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

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

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

Additionally, the *objective function* from time instant 0 to time instant N is defined by

$$J_{0\to N}(\boldsymbol{x}_0, U_{0\to N}) = \sum_{k=0}^{N-1} h(\boldsymbol{x}_k, \boldsymbol{u}_k) + p(\boldsymbol{x}_N) \tag{2.3}$$

where N is the time horizon and $\boldsymbol{x}_k$ is the state vector at time k obtained by the state equations. 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 $\boldsymbol{u}_0$ is applied to the system. This results in a Constrained Finite Time Optimal Control (CFTOC) problem:

$$J^*_{0\to N}(\boldsymbol{x}_0, U_{0\to N}) = \min_{U_{0\to N}} \left[\sum_{k=0}^{N-1} h(\boldsymbol{x}_k, \boldsymbol{u}_k) + p(\boldsymbol{x}_N) \right] \tag{2.4}$$

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 is nonconvex.
Solving eq. 2.4 and applying the first optimal input at every time step results in a *Receding horizon controller (RHC)*.

2.2 Learning Model Predictive Control

Learning Model Predictive Control (LMPC) is a novel control theory that allows to find an optimal trajectory in an iterative task by learning from previous iterations. It is based on the concept of Iterative Learning Control (ILC). ILC is a control strategy that minimizes 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] gives an overview on 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 under only low computational cost.
Definition of iterative processes
There are two types of iterative processes that have to be distinguished: *Batch processes* and *Continuous repetitive processes* [7]. 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 Following theory has been adapted from [3].
First, we define the stage cost of a state $\boldsymbol{x}_t^j$ at time t in iteration j as follows:

$$h(\boldsymbol{x}_F, 0) = 0 \text{ and } h(\boldsymbol{x}_t^j, u_t^j) > 0 \;\forall\; \boldsymbol{x}_t^j \in \mathbb{R}^n \setminus \{\boldsymbol{x}_F\}, \boldsymbol{u}_t^j \in \mathbb{R}^m \setminus \{0\} \tag{2.5}$$

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:

$$J^j = J^j_{0\to\infty}(\boldsymbol{x}_0^j) = \sum_{k=0}^{\infty} h(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j). \tag{2.6}$$

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

$$J^j_{t\to\infty}(\boldsymbol{x}_t^j) = \sum_{k=t}^{\infty} h(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j). \tag{2.7}$$

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

$$\mathcal{SS}^j = \left\{ \bigcup_{i\in M^j} \bigcup_{t=0}^{\infty} \boldsymbol{x}_t^i \right\} \tag{2.8}$$

with

$$M^j = \left\{ k \in [0, j] : \lim_{t\to\infty} \boldsymbol{x}_t^k = \boldsymbol{x}_F \right\}. \tag{2.9}$$

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

$$Q^j(\boldsymbol{x}) = \begin{cases} \min\limits_{(i,t)\in F^j(\boldsymbol{x})} J^i_{t\to\infty}(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in \mathcal{SS}^j \\ +\infty, & \text{if } \boldsymbol{x} \notin \mathcal{SS}^j \end{cases} \tag{2.10}$$

with

$$F^j(\boldsymbol{x}) = \left\{ (i,t) : i \in [0, j], t \geq 0 \text{ with } \exists\; \boldsymbol{x}_t^i \in \mathcal{SS}^j \text{ and } \boldsymbol{x}_t^i = \boldsymbol{x} \right\}. \tag{2.11}$$

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

$$J^{\text{LMPC},j}_{t\to t+N}(\boldsymbol{x}^j_t) = \min_{\boldsymbol{u}_{t|t},\dots,\boldsymbol{u}_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, u_{k|t}) + Q^{j-1}(\boldsymbol{x}_{t+N|t}) \right] \tag{2.12a}$$

$$\text{s.t.} \tag{2.12b}$$

$$\boldsymbol{x}_{k+1|t} = f(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}), \forall k \in [t, ..., t+N], \tag{2.12c}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}^j_t, \tag{2.12d}$$

$$\boldsymbol{x}_{k|t} \in \mathcal{X}, \boldsymbol{u}_k \in \mathcal{U}, \forall k \in [t, ..., t+N], \tag{2.12e}$$

$$\boldsymbol{x}_{t+N|t} \in \mathcal{SS}^{j-1}. \tag{2.12f}$$

It can be shown that this control strategy is recursively feasible and asymptotically stable. 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 x^j_0 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 ... using the mathematical tools we introduced in section 2.2.

Not-so-mathematical description of the proving strategy: Use two consecutive laps, stacked together, for the safe set and cost function.
Start by constructing a safe set consisting of laps 0 and 1 in which initial and final states are equal (path following). Then start lap 2 and use this safe set and its Q function. At the end of lap 2, the controller can predict beyond the finish line, into the safe set of lap 1. After the finish line has been reached and the laps have been switched (from lap 2 to lap 3), we can use the stacked safe set of laps 0+1 and laps 1+2. However, the initial state is not necessarily the same as before.
This idea of stacking safe sets and Q functions is presented below.

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

$$f(\boldsymbol{x}_k + \boldsymbol{p}, \boldsymbol{u}_k) = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{p} \tag{2.13}$$

$$h(\boldsymbol{x} + \boldsymbol{p}, \boldsymbol{u}) = h(\boldsymbol{x}, \boldsymbol{u}). \tag{2.14}$$

If the dynamics are periodic in one state, the periodicity is 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. $\boldsymbol{x}\boldsymbol{e}^T_i \geq P$.
Continuous repetitive systems are defined by smooth transitions between two iterations, i.e. the final state of one iteration becomes the initial state of the following iteration. This implicates that, at the end of one iteration, the control needs to take the beginning of the next iteration into account in order to find an optimal input.
We achieve this by creating a safe set that extends beyond the final state of one iteration to the end of the consecutive iteration.

Definitions First, we define the stacked sampled Safe Set $\hat{\mathcal{SS}}^j$ at iteration j that extends beyond the finish line (*note:* it basically just contains stacks of two consecutive trajectories, e.g. laps 0+1, 1+2, 2+3, ...)

$$\hat{\mathcal{SS}}^j = \left\{ \bigcup_{i \in M^j} \left[\left(\bigcup_{t=0}^{\infty} \boldsymbol{x}_t^i \right) \cup \left(\left(\bigcup_{t=0}^{\infty} \boldsymbol{x}_t^{i+1} \right) + \boldsymbol{p} \right) \right] \right\} \tag{2.15}$$

with

$$M^j = \left\{ k \in [0, j] : \lim_{t \to \infty} \boldsymbol{x}_t^{k+1} = \boldsymbol{x}_F \right\} \tag{2.16}$$

We can also write this definition of the stacked sampled safe set as follows:

$$\hat{\mathcal{SS}}^j = \left\{ \mathcal{SS}^j \cup \left(\mathcal{SS}^{j+1} + \boldsymbol{p} \right) \right\} \tag{2.17}$$

Note that this definition of the stacked sampled safe set can also be written as a function of x:

$$\boldsymbol{x} \in \hat{\mathcal{SS}}^j \rightarrow \begin{cases} \boldsymbol{x} \in \mathcal{SS}^j & \text{if } \boldsymbol{x} < P, \\ \boldsymbol{x} \in \mathcal{SS}^{j+1} & \text{if } \boldsymbol{x} \geq P \end{cases} \tag{2.18}$$

Similarly to the single iteration cost-to-go from eq. 2.7, we define the *stacked cost-to-go* $\hat{J}^j_{t \to \infty}(\boldsymbol{x}_t^j)$ of iteration j as follows:

$$\hat{J}^j_{t \to \infty}(\boldsymbol{x}_t^j) = \begin{cases} J^j_{t \to \infty}(\boldsymbol{x}_t^j) + J^{j+1} & \text{if } \boldsymbol{x} < P, \\ J^{j+1}_{t \to \infty}(\boldsymbol{x}_t^{j+1}) & \text{if } \boldsymbol{x} \geq P \end{cases} \tag{2.19}$$

Note: the stacked cost-to-go is a piecewise defined function, it is continuous through the lap switching condition (at P) and it is zero at $2P$. It is illustrated in fig. 2.1.

This stacking extends the cost-to-go beyond the finish line and allows at the end of one lap to

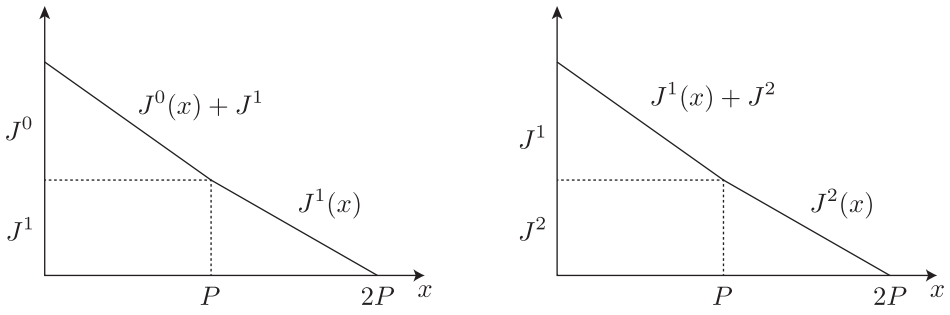

Figure 2.1: Stacked cost-to-go $\hat{J}^0$ and $\hat{J}^1$

consider the beginning of the next lap (for which the predicted $x_{t+N|t} > P$).
Using this definition of the stacked cost-to-go we define the stacked Q-function:

$$\hat{Q}^j(\boldsymbol{x}) = \begin{cases} \min\limits_{(i,t) \in F^j(\boldsymbol{x})} \hat{J}^i_{t \to \infty}(\boldsymbol{x}), & \text{if } x \in \hat{\mathcal{SS}}^j, \\ +\infty & \text{if } x \notin \hat{\mathcal{SS}}^j. \end{cases} \tag{2.20}$$

We can also write the stacked Q function depending on x:

$$\hat{Q}^j(\boldsymbol{x}) = \begin{cases} \min\limits_{i,t} \left[J^i_{t \to \infty}(\boldsymbol{x}) + J^{i+1} \right], & \text{if } \boldsymbol{x} < P, i \in [0, j] \\ \min\limits_{i,t} \left[J^{i+1}_{t \to \infty}(x) \right], & \text{if } \boldsymbol{x} > P, i \in [0, j]. \end{cases} \tag{2.21}$$

With these definitions, we redefine the LMPC function which yields the repetitive LMPC function:

$$\hat{J}^{\text{LMPC},j}_{t\to t+N}(\boldsymbol{x}^j_t) = \min_{\boldsymbol{u}_{t|t},\dots,\boldsymbol{u}_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, u_{k|t}) + \hat{Q}^{j-2}(\boldsymbol{x}_{t+N|t}) \right] \tag{2.22a}$$

$$\text{s.t.} \tag{2.22b}$$

$$\boldsymbol{x}_{k+1|t} = f(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}), \forall k \in [t, ..., t+N], \tag{2.22c}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}^j_t, \tag{2.22d}$$

$$\boldsymbol{x}_{k|t} \in \mathcal{X}, \boldsymbol{u}_k \in \mathcal{U}, \forall k \in [t, ..., t+N], \tag{2.22e}$$

$$\boldsymbol{x}_{t+N|t} \in \hat{\mathcal{SS}}^{j-2}. \tag{2.22f}$$

Note: Since we use stacks of 2 consecutive laps for the safe set and Q function, we can't use the previous safe set $j-1$ but instead the one even further back, $j-2$. Our stacked safe set $\hat{\mathcal{SS}}^0$ also has to contain two successful laps before the LMPC starts!

Recursive feasibility Recursive feasibility throughout one iteration is guaranteed as usual. However, we have to prove feasibility even through the switching between two laps.
Consider state $\boldsymbol{x}^j_t$ in iteration j. Then the terminal constraint enforces $\boldsymbol{x}^{*,j}_{t+N|t} \in \hat{\mathcal{SS}}^{j-2}$. Assume that $\boldsymbol{x}^j_t$ is the last state of iteration j, meaning that $\boldsymbol{x}^{*,j}_{t+1|t} \geq P$ and therefore $\boldsymbol{x}^{*,j}_{t+N|t} \geq P$.
Using the definition of the stacked safe set in eq. 2.18 leads to $\boldsymbol{x}^{*,j}_{t+N|t} \in \mathcal{SS}^{j-1}$.
As before, the state update and prediction are assumed identical, leading to

$$\boldsymbol{x}^{*,j}_{t+1|t} = \boldsymbol{x}^j_{t+1} = \boldsymbol{x}^{j+1}_0 \tag{2.23}$$

In iteration $j+1$ we use the the stacked safe set $\hat{\mathcal{SS}}^{j-1}$ with $\boldsymbol{x} < P$, leading to $\boldsymbol{x} \in \mathcal{SS}^{j-1}$. This means that the state trajectory

$$\left(x^{*,j}_{t+1|t}, \quad x^{*,j}_{t+2|t}, \quad ..., \quad x^{*,j}_{t+N-1|t}, \quad x^{i^*}_{t^*|t}, \quad x^{i^*}_{t^*+1|t}\right) = \tag{2.24}$$

$$\left(x^{*,j+1}_{0|t}, \quad x^{*,j+1}_{1|t}, \quad ..., \quad x^{*,j+1}_{N-2|t}, \quad x^{i^*}_{t^*|t}, \quad x^{i^*}_{t^*+1|t}\right) \tag{2.25}$$

is still feasible in iteration $j+1$.

Asymptotic stability The proof for asymptotic stability within an iteration works as in the standard LMPC case. As usual, we can show that $J^{\text{LMPC},j}_{0\to N}(x^j_t)$ is a decreasing Lyapunov function in lap j and we can show that $J^{\text{LMPC},j+1}_{0\to N}(x^j_t)$ is a decreasing Lyapunov function in lap $j+1$. However, we are interested in the stability when the lap switching happens. Looking at the LMPC cost at the state at the switching condition shows an incontinuity.

Consider state $\boldsymbol{x}_t^j = \boldsymbol{x}_0^{j+1}$ as the last state of iteration j and the first state of iteration $j+1$.

$$J_{0\to N}^{\text{LMPC},j}(x_t^j) = \min_u \left[\sum_{k=0}^{N-1} h(x_{k|t}, u_{k|t}) + \hat{Q}^{j-2}(x_{N|t})\right] \quad \text{and } x_{N|t} > P \tag{2.26}$$

$$= \min_u \left[\sum_{k=0}^{N-1} h(x_{k|t}, u_{k|t}) + Q^{j-1}(x_{N|t})\right] \tag{2.27}$$

$$= \min_u \left[\sum_{k=0}^{N-1} h(x_{k|t}, u_{k|t}) + \min_{i,t}\left[J_{t\to\infty}^i(x)\right]\right] \quad \text{and } i \in [0, j-1] \tag{2.28}$$

$$J_{0\to N}^{\text{LMPC},j+1}(x_0^{j+1}) = \min_u \left[\sum_{k=0}^{N-1} h(x_{k|0}, u_{k|0}) + \hat{Q}^{j-1}(x_{N|0})\right] \quad \text{and } x_{N|0} < P \tag{2.29}$$

$$= \min_u \left[\sum_{k=0}^{N-1} h(x_{k|0}, u_{k|0}) + \min_{i,t}\left[J_{t\to\infty}^i(x) + J^{i+1}\right]\right] \quad \text{and } i \in [0, j-1] \tag{2.30}$$

and therefore

$$J_{0\to N}^{\text{LMPC},j+1}(x_0^{j+1}) \geq J_{0\to N}^{\text{LMPC},j}(x_t^j). \tag{2.31}$$

This result essentially just shows that the LMPC cost at the initial state of lap $j+1$ is larger or equal to the LMPC cost at the same state, but evaluated in the previous lap j.
Question: Can we still assume stability through iterations since the cost function is stable all the way *to* the switching condition and it is stable starting *from* the switching condition?

Non decreasing iteration cost Using the strategy above, we will have to focus on the transition between two laps. The "problem" is that - at the beginning of an iteration, right after the transition - the controller doesn't even notice that laps were switched. The safe set is the same and the Q function only increased by a constant value.

Chapter 3

Application on race driving

This chapter presents the application of the previously developed LMPC to race driving. First, two car model formulations are presented which approximate real car dynamics with different accuracies. Then, the LMPC 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. As opposed to real world race driving in which usually multiple cars are involved, we are assuming single car race driving to simplify this problem. Mathematically, this can be represented by an optimization problem that 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 the shape of the track this can result in a very complex optimization problem which cannot be solved in real time.

3.2 Car model formulations

There are many ways to model the dynamics of a car. They range from simple models with few states which manage to approximate real car dynamics for low speeds to sophisticated models with a great number of states wich 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 whereas 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 [8]. It is based on the no-slip condition which means that the velocity vectors at both wheels A and B 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.

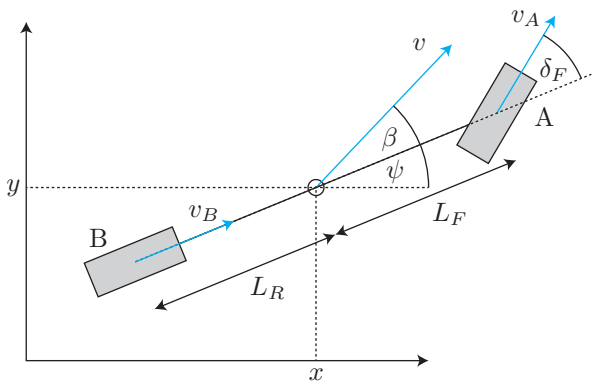

Figure 3.1: Kinematic bicycle model

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

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

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

As opposed to the kinematic bicycle model which is derived geometrically, the dynamic bicycle model is based on forces that occur between the front and rear tires and the road. Its states are the longitudinal and lateral velocity v_x and v_y which are represented in the body fixed frame, and the yaw rate $\dot{\psi}$. The state dynamics presented in eq. 3.3 can be derived by applying Newton's second law of motion and have been taken from [9]. The model equations have been simplified by considering the steering angle δ_F only in the lateral dynamics and neglecting the lateral tire force in the longitudinal dynamics. The corresponding model is illustrated in fig. 3.3.

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

These state dynamics assume that there is only lateral but no longitudinal tire slip which might occur during high accelerations or decelerations. This is done to simplify the model and since longitudinal acceleration is a control input that we can restrict to remain within bounds to avoid longitudinal tire slip. The tire forces F_F and F_R depend on the lateral slipping angle α_F and α_R of

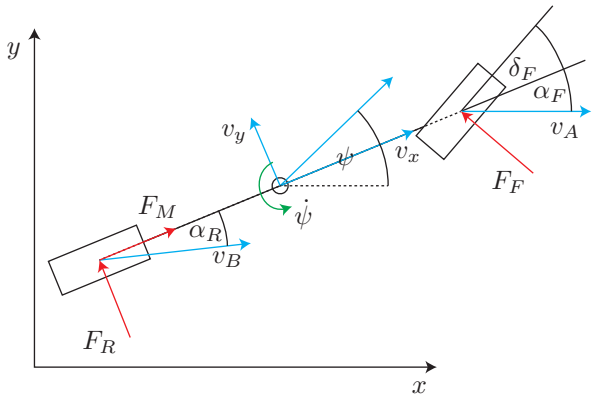

Figure 3.2: Dynamic bicycle model

the front and the rear tire. One common function that approximates the tire forces is the *Pacejka* function from [10] (often referred to as *Magic formula*):

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

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.

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

Table 3.1: Pacejka coefficients for different road conditions

Eq. 3.5 show the relations between the pacejka function and the tire forces as well as the calculation of slip angles which can be derived geometrically.

$$\begin{aligned} F_F &= -\frac{1}{2} \cdot m \cdot g \cdot \mu \cdot f_P(\alpha_F) \\ F_R &= -\frac{1}{2} \cdot m \cdot g \cdot \mu \cdot f_P(\alpha_R) \\ \alpha_F &= \arctan\left(\frac{\dot{y} + L_F\dot{\psi}}{|\dot{x}|}\right) - \delta_F \\ \alpha_R &= \arctan\left(\frac{\dot{y} - L_R\dot{\psi}}{|\dot{x}|}\right) \end{aligned} \tag{3.5}$$

The dynamic bicycle model is more complex than the kinematic model but approximates real car dynamics very well even for higher speeds and various street 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 might 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,

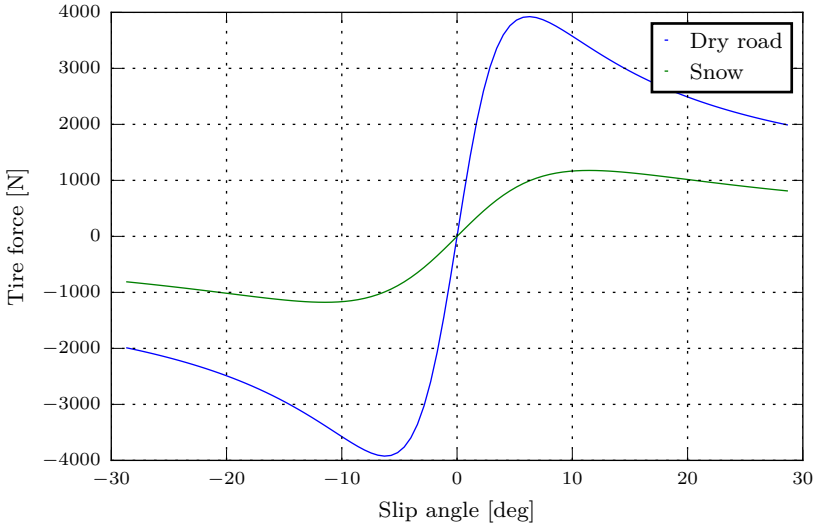

Figure 3.3: Pacejka tire model

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 [11]. The coordinate frame is illustrated in fig. 3.4.

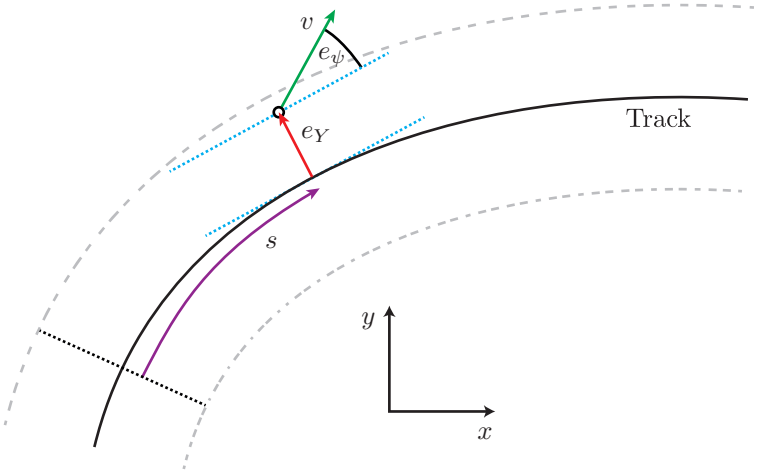

Figure 3.4: Dynamic bicycle model

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 [8]):

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

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

$$\dot{e}_\psi = \frac{v}{L_F} \cdot \sin(\beta) - \frac{ds}{dt} \cdot c(s) \tag{3.8c}$$

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

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

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

$$\dot{e}_\psi = \dot{\psi} - \frac{ds}{dt} \cdot c(s) \tag{3.9b}$$

with

$$\frac{ds}{dt} = \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_Y \cdot c(s)} \tag{3.10}$$

We are going to use the model 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

$$\boldsymbol{x}_t^j = \begin{pmatrix} v_{x,t}^j & v_{y,t}^j & \dot{\psi}_t^j & e_{\psi,t}^j & e_{y,t}^j & s_t^j \end{pmatrix} \tag{3.11}$$

$$\boldsymbol{u}_t^j = \begin{pmatrix} a_t^j & \delta_t^j \end{pmatrix}. \tag{3.12}$$

and equation 2.12 as the optimization function. The racing problem is formulated by using a constant cost function on whenever the car is located between the start and finish line and zero cost when the car has crossed the finish line [3]. Thus the stage cost is defined as follows:

$$h(\boldsymbol{x}_t, \boldsymbol{u}_t) = \begin{cases} 1 & \text{if } \boldsymbol{x}_t \notin \mathcal{L}, \\ 0 & \text{if } \boldsymbol{x}_t \in \mathcal{L}. \end{cases} \tag{3.13}$$

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

$$\mathcal{L} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \boldsymbol{x}\boldsymbol{e}_6^T = s > s_{target} \right\} \tag{3.14}$$

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:

$$-\frac{w_{Lane}}{2} \leq e_Y \leq \frac{w_{Lane}}{2} \tag{3.15}$$

with lane width w_{Lane}.

3.4 Approximation of the safe set and LP relaxation

The LMPC problem from 2.22 minimizes its objective function over the safe set which is a set of discrete states of previous iterations. Therefore, this formulation is a Mixed Integer Nonlinear Programming (MINLP) problem. Since 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 [12]. We define the time varying approximated safe set $\tilde{\mathcal{SS}}_t^{j,j-1}$ and the time varying approximated terminal cost function $\tilde{Q}_t^{j,j-1}(\cdot)$.

$$\tilde{\mathcal{SS}}_t^{j,j-1} = \left\{ \boldsymbol{x} \in \mathbb{R}^5, \lambda \in [0,1] : x \in \lambda \tilde{\boldsymbol{x}}_t^{j,j-1} + (1-\lambda)\tilde{\boldsymbol{x}}_t^{j,j-2} \right\} \tag{3.16}$$

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

$$\tilde{\boldsymbol{x}}_t^{j,k} = \left\{ \boldsymbol{x} \in \mathbb{R}^5 : \forall i \in \{v_x, v_y, \dot{\psi}, e_\psi, e_y\}, i = \begin{pmatrix} s^n & s^{n-1} & \dots & s & 1 \end{pmatrix} \boldsymbol{\Gamma}_{t,i}^{j,k} \right\}, \tag{3.17}$$

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

$$C_t^{j,j-1}(\boldsymbol{x}) = \begin{cases} \begin{pmatrix} s^n & s^{n-1} & \dots & s & 1 \end{pmatrix} \boldsymbol{\Delta}_t^{j,k}, & \text{if } \boldsymbol{x} \in \tilde{\boldsymbol{x}}_t^{j,k}, \\ +\infty, & \text{if } \boldsymbol{x} \notin \tilde{\boldsymbol{x}}_t^{j,k}, \end{cases} \tag{3.18}$$

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 $Q^{j-1}(\cdot)$:

$$\tilde{Q}_t^{j,j-1}(\boldsymbol{x}, \lambda) = \begin{cases} \lambda C_t^{j,j-1}(\boldsymbol{x}) + (1-\lambda) C_t^{j,j-2}(\boldsymbol{x}), & \text{if } (\boldsymbol{x}, \lambda) \in \tilde{\mathcal{SS}}_t^{j,j-1} \\ +\infty, & \text{if } (\boldsymbol{x}, \lambda) \notin \tilde{\mathcal{SS}}_t^{j,j-1} \end{cases} \tag{3.19}$$

Having defined the relaxation of the safe set and approximation of the terminal cost function we can rewrite the LMPC formulation from eq. 2.12:

$$\tilde{J}_{t\to t+N}^{\text{LMPC},j}(\boldsymbol{x}_t^j) = \min_{\lambda, \boldsymbol{u}_{t|t}, \dots, \boldsymbol{u}_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} h(\boldsymbol{x}_{k|t}, u_{k|t}) + \tilde{Q}^{j,j-1}(\boldsymbol{x}_{t+N|t}, \lambda) \right] \tag{3.20a}$$

$$\text{s.t.} \tag{3.20b}$$

$$\boldsymbol{x}_{k+1|t} = f(\boldsymbol{x}_{k|t}, \boldsymbol{u}_{k|t}), \forall k \in [t, \dots, t+N], \tag{3.20c}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}_t^j, \tag{3.20d}$$

$$\boldsymbol{x}_{k|t} \in \mathcal{X}, \boldsymbol{u}_k \in \mathcal{U}, \forall k \in [t, \dots, t+N], \tag{3.20e}$$

$$(\boldsymbol{x}_{t+N|t}, \lambda) \in \tilde{\mathcal{SS}}^{j,j-1}. \tag{3.20f}$$

In practice, the polynomial degree 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 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. Is this true? Normally, we would have to consider *all* previous trajectories. 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.

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.

Parameter	Value
Mass	$m = 2.0\,\mathrm{kg}$
Axle distances	$L_F = L_R = 0.125\,\mathrm{m}$
Moment of inertia	$I_z = 0.03\,\mathrm{kg\,m^2}$
Road friction	$\mu = 0.8$
Stiffness factor	$B = 10$
Shape factor	$C = 1.9$
Peak factor	$D = 1.0$

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 MPC 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.
Lane constraints are implemented as soft constraints in the cost function:

$$J_{Lane} = \sum_{i=1}^{N} [c_1 \epsilon_i^2 + c_2 \epsilon_i], \epsilon_i = |e_{Y,i}| - \frac{w_{Lane}}{2} \tag{3.21}$$

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.

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:

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

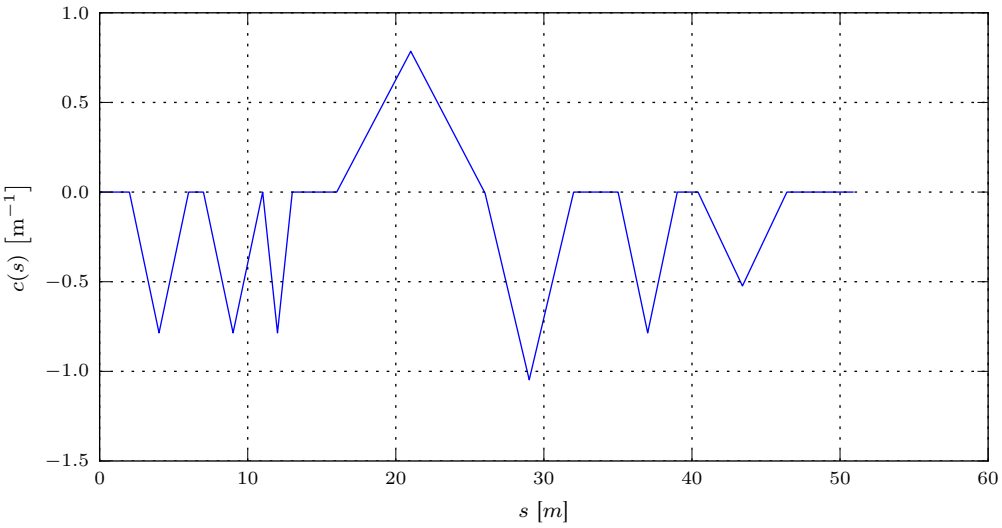

Figure 3.5: Curvature $c(s)$

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

$$J_{deriv} = \sum_{i=1}^{N-1} (\boldsymbol{x}_{i+1} - \boldsymbol{x}_i)^T \boldsymbol{Q}_x (\boldsymbol{x}_{i+1} - \boldsymbol{x}_i) + \sum_{i=1}^{N-2} (\boldsymbol{u}_{i+1} - \boldsymbol{u}_i)^T \boldsymbol{Q}_u (\boldsymbol{u}_{i+1} - \boldsymbol{u}_i) \tag{3.22}$$

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 3.6 to 3.9 show the results of the race problem with the kinematic model.

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 (that extends beyond the end of one lap).
Figure 3.6 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, but it doesn't completely converge to straight lines between the curves, which would be the global optimum. This is due to the selected horizon length which doesn't allow the MPC to predict a benefit from going along straight instead of curved lines. This behavior of leaving a lane boundary just to come back to it shortly after is even clearer in fig. 3.7 (at about $s = 0$ and $33\,\mathrm{m}$).
Figure 3.8 shows the velocity during three selected laps. The car reaches its maximum velocity (which is limited by constrained acceleration) in the first LMPC lap and keeps it throughout the entire lap. Figure 3.9 shows that the trajectory converges to a optimal trajectory within very few iterations.

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 acceleration input was constrained to $-1.8\,\mathrm{m\,s^{-2}} \leq a \leq 1.8\,\mathrm{m\,s^{-2}}$ in order to prevent too high velocites which can lead to numerical issues at our chosen sampling time of $dt = 0.02\,\mathrm{s}$. We used the physical parameters of the BARC and a Pacejka model with coefficients $B = 5.0, C = 2.0, D = 1.0$. Figures 3.10 to 3.15 show the simulation results.

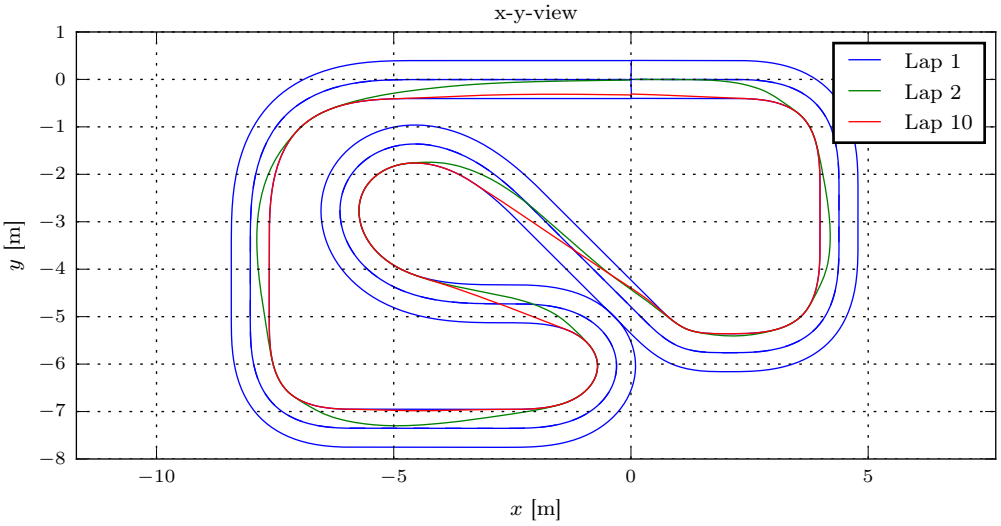

Figure 3.6: Simulation kinematic model

The first two laps are controlled by a path following controller at a reference speed of $2.0\,\mathrm{m\,s^{-1}}$. After that, the LMPC strategy is able to reduce the lap time to 14 s at an average speed of $3.6\,\mathrm{m\,s^{-1}}$. 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. Figure 3.13 illustrates shows a plot of longitudinal and lateral acceleration which is also known as the *friction circle*. It can be seen that the maximum lateral accelerations are about symmetric with maximal values of $1.3\,\mathrm{m\,s^{-2}}$. Longitudinal accelerations are smaller which results from the constrained acceleration and derivative cost on the acceleration input, simulating low tire forces in longitudinal direction.

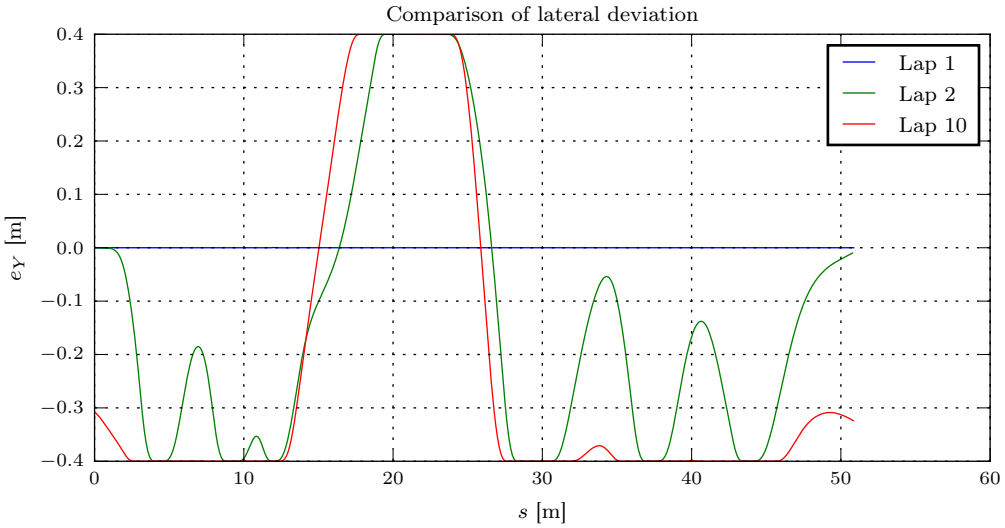

Figure 3.7: Lateral error e_Y

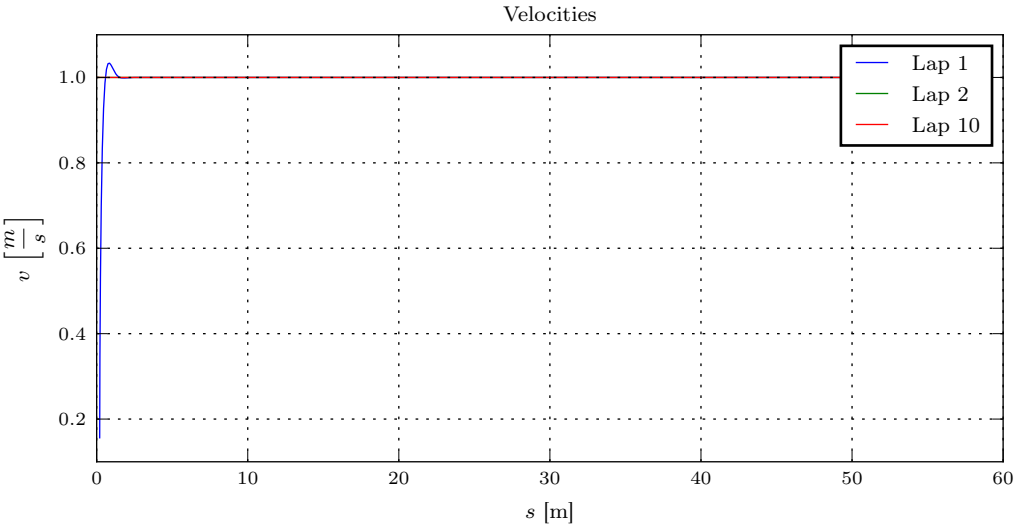

Figure 3.8: Velocity v

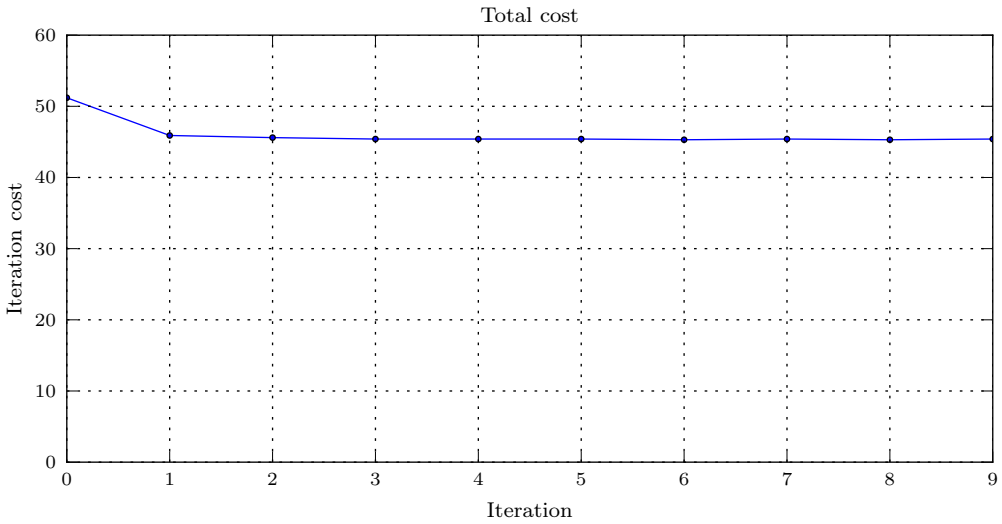

Figure 3.9: Cost

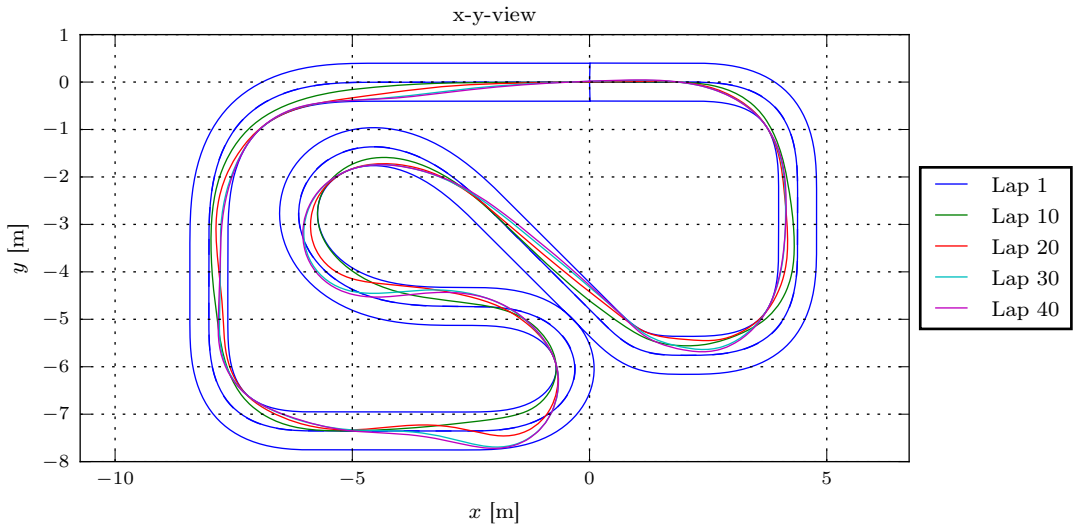

Figure 3.10: 2D view

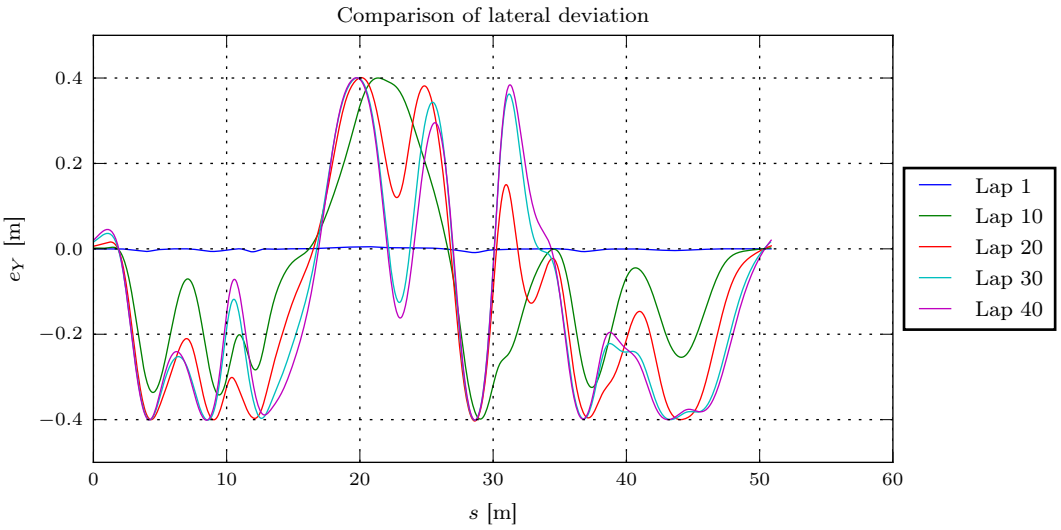

Figure 3.11: Lateral position error

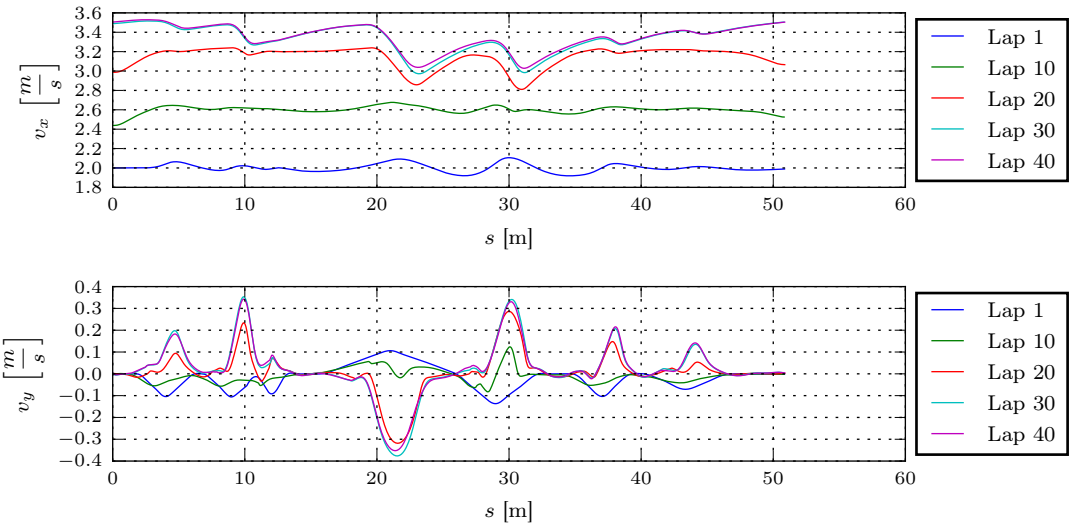

Figure 3.12: Velocities

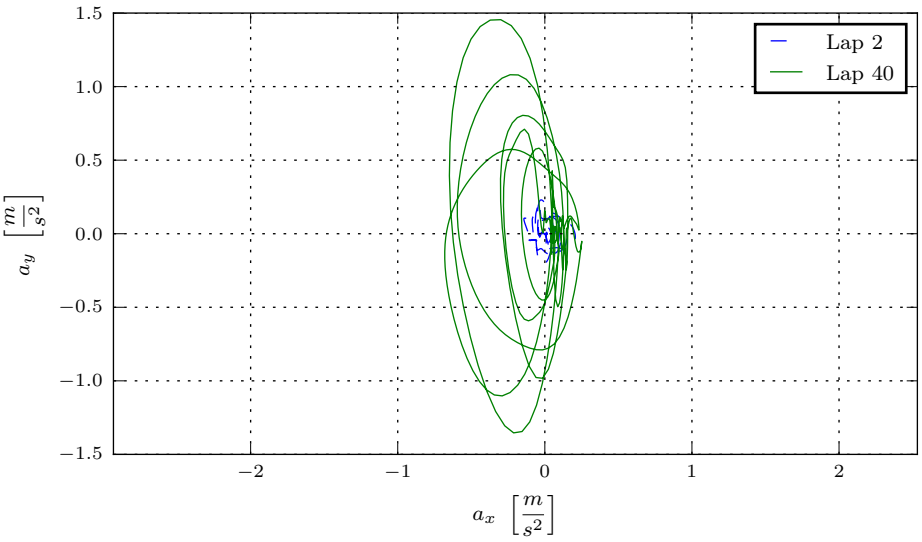

Figure 3.13: Friction circle

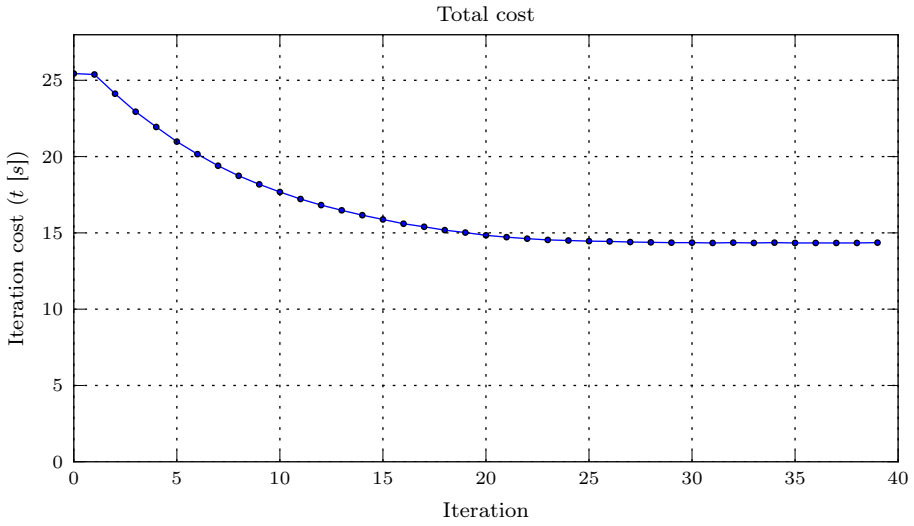

Figure 3.14: Iteration cost

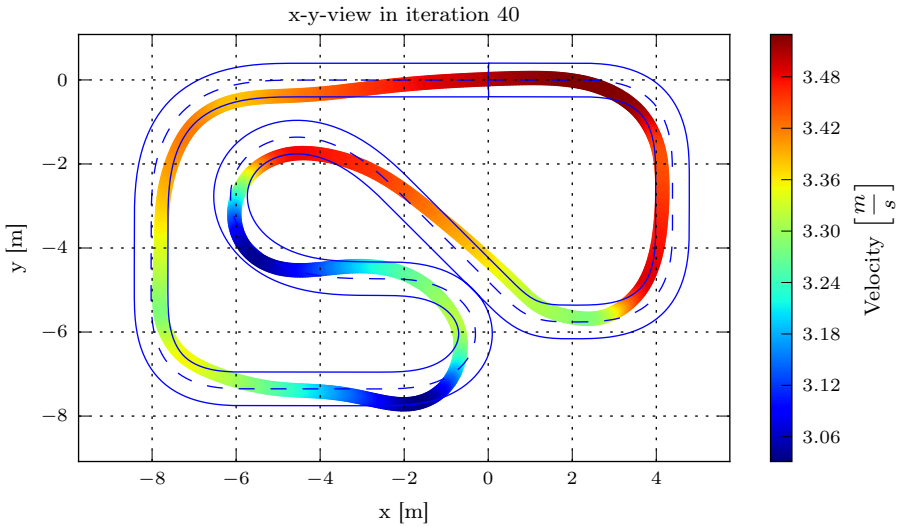

Figure 3.15: Velocity over x and y

Chapter 4

Implementation

In this chapter the presented control strategy is implemented for a 1:10 scale remote-controlled car. The car is a commercially available hobby race car of the model "Basher RZ-4 1/10 Rally Racer" which has been modified to include sensors and onboard CPUs. First, the system and its available sensors and actuators are introduced. A system identification strategy is presented which allows online estimation of system dynamics. Next, the state estimation and experiments are shown. Figure 4.1 shows a simplified control block diagram. After an introduction of the system, this chapter goes through the blocks of system identification, state estimation, and control.

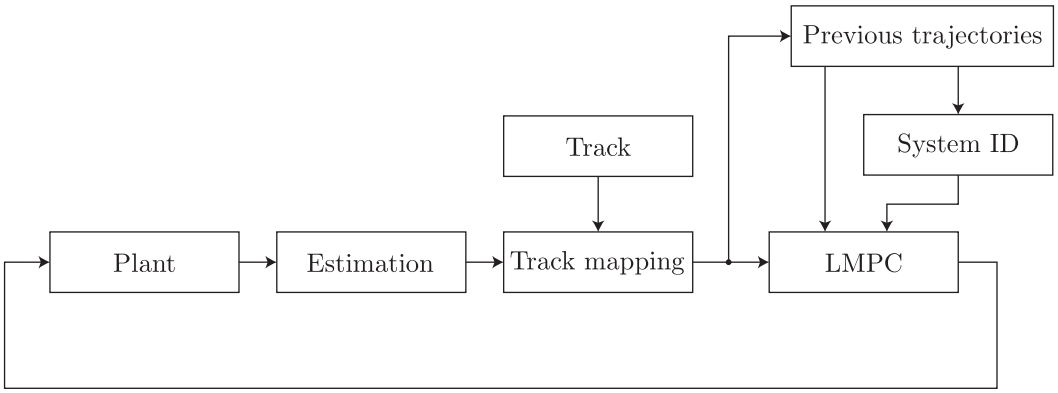

Figure 4.1: Control structure

4.1 Introduction to the BARC

The car used for our experiments is a remote controlled race car 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, [**?**]) and it has been used previously in student projects. The basic setup is shown in figure 4.2.

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 (like communication with USB devices and external computers).

Figure 4.2: Basic BARC setup

4.1.1 Sensors

For the purpose of race driving the estimation of absolute position, velocity and orientation are needed. The main idea in the selection of sensors was to provide a low cost solution that is not bound to a specific testing area. This is why following 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 can be 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 signal at each half spin of the wheel. The arduino sends the average speed resulting form the signals to the odroid.

GPS To determine the absolute position of the car, an indoor positioning system from *Marvelmind robotics* [13] 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. This section first describes the procedure to find the steering mapping followed by the acceleration mapping.

Steering mapping 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:

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

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 are 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

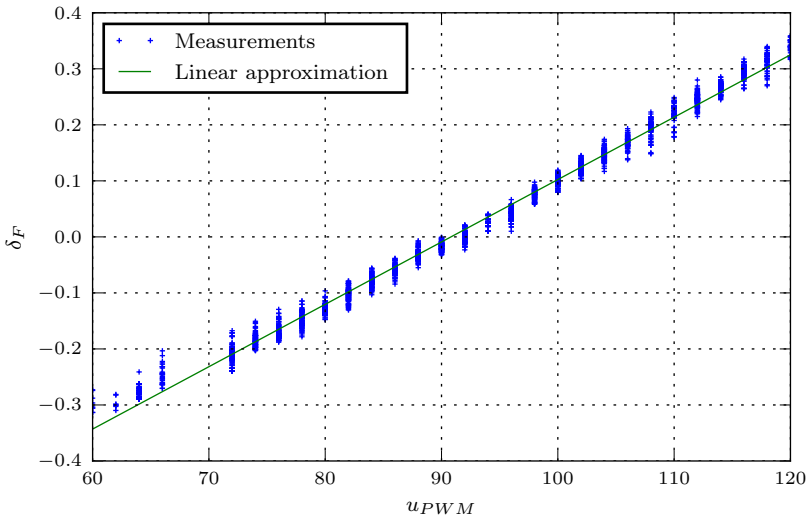

Figure 4.3: Steering mapping

wheelbase $L_R + L_F$, to a minimum 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 $c_{max} = 1.24\,\text{m}^{-1}$.
Open loop experiments with fast changing steering inputs showed that the servo exhibits a short time delay of about 0.2 seconds. This delay is shown in fig. 4.4. Since the LMPC formulation uses a sample time of $dt = 0.1\,\text{s}$, we model this delay as a 2-step delay:

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

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 linear functions. Additionally, we have to find the motor drag which we model

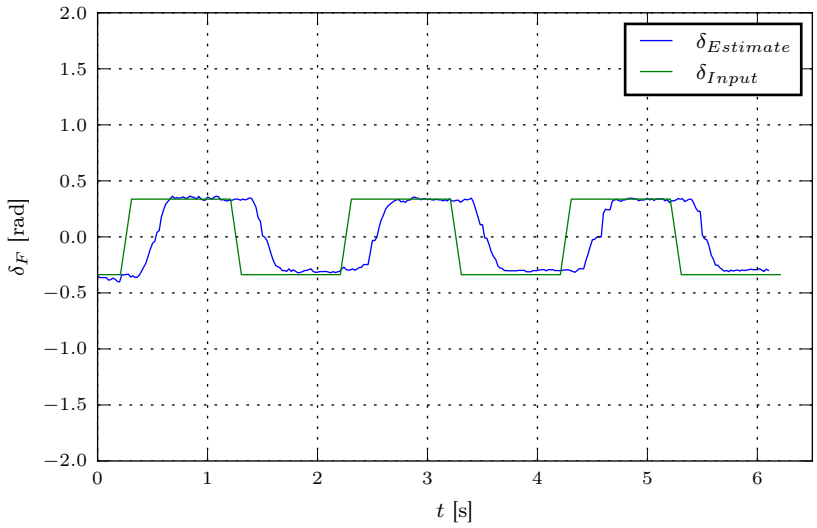

Figure 4.4: Steering delay

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:

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

$$\dot{v} = c_1^- \cdot u_{PWM} + c_2^- - c_M \cdot v \; \forall \dot{v} < 0 \tag{4.8}$$

To identify the coefficients of these two regimes, two groups of open loop tests are performed: Firstly, different (positive) accelerations are sent to the motor for a fixed interval of 4 seconds before the car is stopped. Secondly, the car is 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.
We received following results for the coefficients:

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

Table 4.1: Pacejka coefficients for different road conditions

Both the linear function and the measured accelerations are illustrated in fig. 4.5.

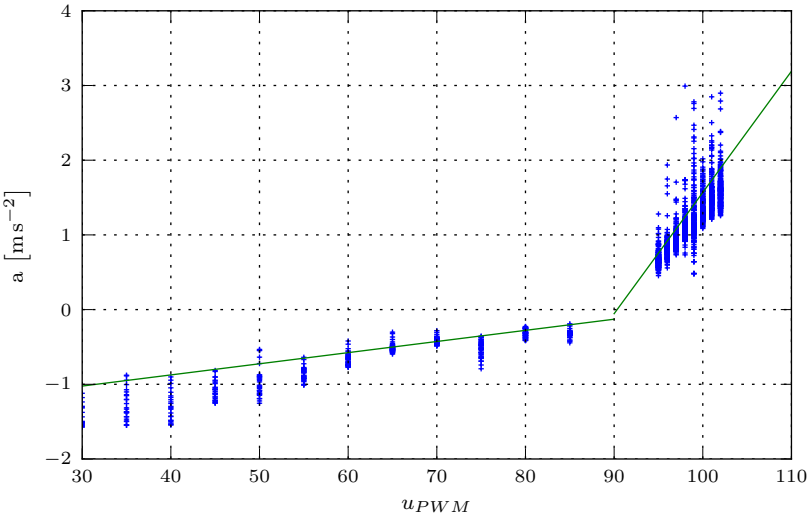

Figure 4.5: Acceleration mapping

4.2 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:

$$v_{x,k+1} - v_{x,k} = \theta_{x,1} \cdot v_{y,k} \cdot \dot{\psi}_k + \theta_{x,2} \cdot v_{x,k} + \theta_{x,3} \cdot a_k \tag{4.9}$$

$$v_{y,k+1} - v_{y,k} = \theta_{y,1} \cdot \frac{v_{y,k}}{v_{x,k}} + \theta_{y,2} \cdot v_{x,k}\dot{\psi}_k + \theta_{y,3} \cdot \frac{\dot{\psi}_k}{v_{x,k}} + \theta_{y,4} \cdot \delta_k \tag{4.10}$$

$$\dot{\psi}_{k+1} - \dot{\psi}_k = \theta_{\psi,1} \cdot \frac{\dot{\psi}_k}{v_{x,k}} + \theta_{\psi,2} \cdot \frac{v_{y,k}}{v_{x,k}} + \theta_{\psi,3} \cdot \delta_k \tag{4.11}$$

with states v_x, v_y, $\dot{\psi}$ and parameters $\theta_{i,j}$. The features $\boldsymbol{X}$ of this Linear Regression problem are functions of the states and inputs. Using multiple samples n, each equation can be rewritten as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}. \tag{4.12}$$

Equation 4.13 shows the equations for the LR of v_x:

$$\boldsymbol{y} = \begin{pmatrix} v_{x,2} - v_{x,1} \\ v_{x,3} - v_{x,2} \\ \vdots \\ v_{x,n} - v_{x,n-1} \end{pmatrix} \qquad \boldsymbol{X} = \begin{pmatrix} v_{y,1} & \dot{\psi}_1 & a_1 \\ v_{y,2} & \dot{\psi}_2 & a_2 \\ \vdots & & \\ v_{y,n-1} & \dot{\psi}_{n-1} & a_{n-1} \end{pmatrix} \qquad \boldsymbol{\theta} = \begin{pmatrix} \theta_{x,1} \\ \theta_{x,2} \\ \theta_{x,3} \end{pmatrix} \tag{4.13}$$

The LR equations for v_y and $\dot{\psi}$ can be writte n in the same way.
To determine the optimal parameters $\boldsymbol{\theta}^*$, following minimization has to be solved:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2 \tag{4.14}$$

There are different approaches to solve this problem, in this case the *normal equation* method is used. This leads to following analytic solution:

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{4.15}$$

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.

4.3 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 that knows 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 tried 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 kinematic bicycle model whereas the other one does not know any bicycle dynamics. The filter model is illustrated in fig 4.6.

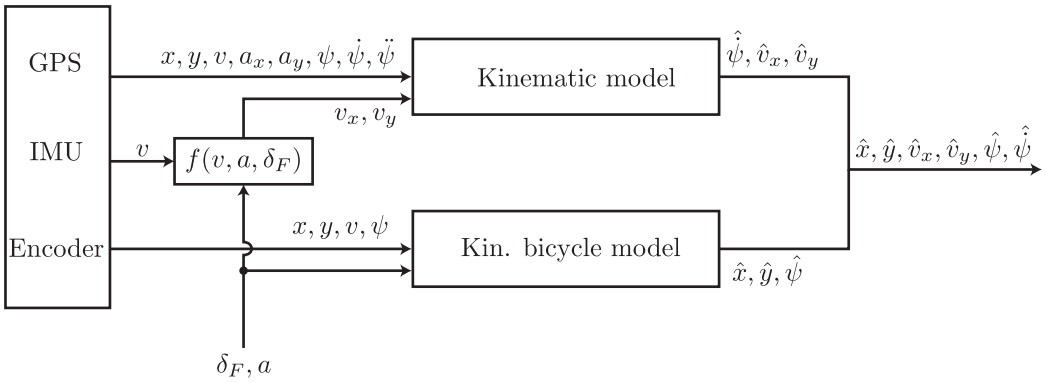

Figure 4.6: Filter model

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 kinematic 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, which is why we use this model for the estimation of the yaw rate and velocities.

4.3.1 Extended Kalman Filter

Kalman filters provide a general tool to calculate a state estimate using noisy and/or biased sensor measurements. 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 [14]. 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:

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}) \tag{4.16}$$
$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1} \tag{4.17}$$

Update:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{4.18}$$
$$\boldsymbol{S}_k = \boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^T + \boldsymbol{R}_k \tag{4.19}$$
$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^T\boldsymbol{S}_k^{-1} \tag{4.20}$$
$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k \tag{4.21}$$
$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1} \tag{4.22}$$

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 the 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 linearized functions $f(\boldsymbol{x}, \boldsymbol{u})$ and $h(\boldsymbol{x})$:

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k}, \boldsymbol{u}_k} \tag{4.23}$$
$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{4.24}$$

4.3.2 Filter model

The first filter features a kinematic bicycle model, eq. 4.25 shows its model equations:

$$\begin{aligned}
\dot{x} &= v \cdot \cos(\psi + \beta) \\
\dot{y} &= v \cdot \sin(\psi + \beta) \\
\dot{\psi} &= \frac{v}{L_R} \cdot \sin(\beta) \\
\dot{v} &= a - c_M \cdot v \\
\dot{d}_\psi &= 0
\end{aligned} \tag{4.25}$$

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:

$$\boldsymbol{z} = \begin{pmatrix} x_m & y_m & \psi_m & v_m \end{pmatrix}^T, \tag{4.26}$$

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

$$h(\hat{\boldsymbol{x}}_{k|k-1}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \hat{\boldsymbol{x}}_{k|k-1} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{\psi} + \hat{d}_\psi \\ \hat{v} \end{pmatrix} \tag{4.27}$$

The second filter features a pure kinematic motion model from [15] which integrates accelerations to find velocities and positions. Eq. 4.28 shows its model equations:

$$\begin{aligned}
\dot{x} &= \cos(\psi)v_x - \sin(\psi)v_y \\
\dot{y} &= \sin(\psi)v_x + \cos(\psi)v_y \\
\dot{v}_x &= a_x + \dot{\psi}v_y \\
\dot{v}_y &= a_y - \dot{\psi}v_x \\
\dot{\psi} &= \dot{\psi} \\
\ddot{\psi} &= 0 \\
\dot{a}_x &= 0 \\
\dot{a}_y &= 0 \\
\dot{d}_\psi &= 0
\end{aligned} \tag{4.28}$$

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

$$\boldsymbol{z} = \begin{pmatrix} x_m & y_m & v_{x,m} & v_{y,m} & \psi_m & \dot{\psi}_m & \ddot{\psi}_m & a_{x,m} & a_{y,m} \end{pmatrix}^T \tag{4.29}$$

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. In order to account for this uncertainty, we always extrapolate the previously received data by a polynomial to the time 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:

$$f = \frac{1}{T} = \frac{v}{\pi r} \tag{4.30}$$

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 about 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 a real update rate of 50 Hertz, we assume a rather high encoder measurement noise in the measurement noise matrix $\boldsymbol{R}$ to account for equal consecutive measurements.

4.3.3 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 polynom of p-th order, returning functions for the coordinates $(x, y) = f(s)$ and curvature $c = f(s)$
4. Use these functions to calculate the current curvilinear abscissa s, lateral position e_Y and heading error e_ψ.

4.4 Implementation

The estimator and MPC solver are implemented in the ROS framework (Robot Operating System, [16]). 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. 4.8. The ROS framework manages the execution of all

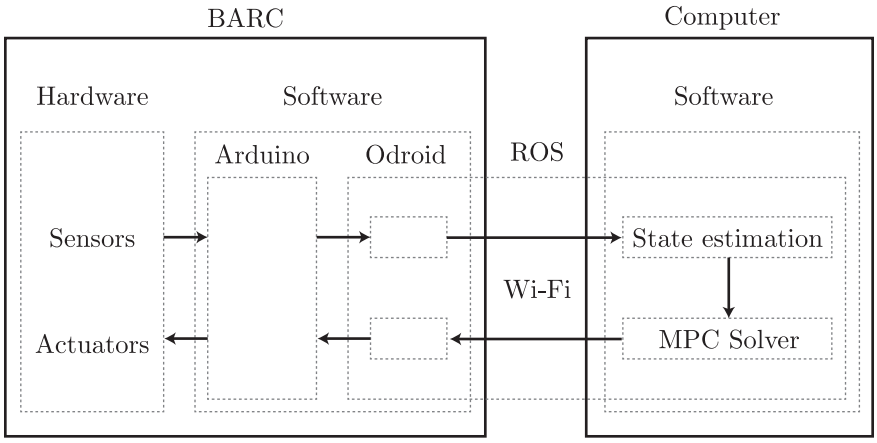

Figure 4.7: Software/Hardware setup

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, which is the highest frequency at which measurements are received by the IMU. Both the encoders and the GPS are running at lower or variable frequencies between 10 and 50 Hz.
The MPC optimization problem is solved by Ipopt [17] which is executed by a script written in Julia 0.4.6 [18]. 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 4.9 shows the timing diagram of ROS nodes and messages.

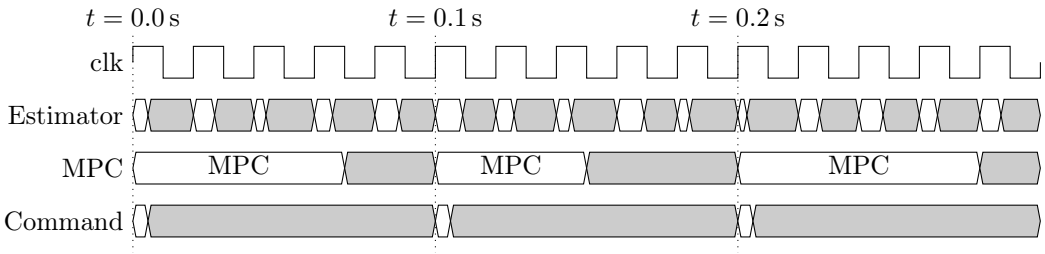

Figure 4.8: Timing diagram

Below are described all steps that are performed during one time step (asynchronously):

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

4.5 Experiments

The experiments were performed in a closed space at UC Berkeley. The available area had a size of about 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 turns in different directions with the maximum possible curvature of $1.24\,\mathrm{m}^{-1}$. The curvature is shown in fig. 4.10. 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.

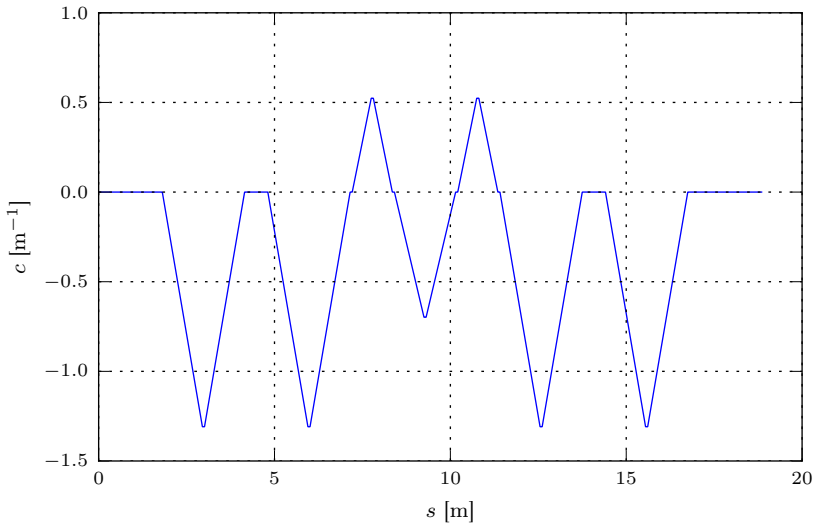

Figure 4.9: Curvature of the experimental race track

Results and discussion After driving 3 laps using a path following MPC with a kinematic bicycle model at $v_{Ref} = 1.0\,\mathrm{m\,s^{-1}}$, the LMPC strategy starts in the 4^{th} lap. The reference speed yields a path following lap time of about $t_{pf} = 21\,\mathrm{s}$. We can see in fig. 4.11 that the lap time decreases rapidly within the first few learning iterations before it converges to a trajectory that takes about $t_{22} = 7.5\,\mathrm{s}$ at an average speed of $2.5\,\mathrm{m\,s^{-1}}$.
Figures 4.12 and 4.13 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 4.14 visualizes the car's velocity at each point of the track during one optimal iteration.
As we can see, the iteration cost in fig. 4.11 does not continuously decrease but instead it oscillates around a mean value of about 7.5 s. This is due to the model mismatch between the MPC formulation and the real car dynamics, which also arises from inaccurate measurements that are used for the system identification. Robust ILMPC and how to handle model mismatch is subject of current research.
We found out that, in order to ensure feasibility and to avoid numerical issues, the coefficients of the soft lane constraints from eq. 3.21 had to be chosen in such a way that they can always be violated by a small amount. In contrast, too high values in combination with model mismatch can lead to solver difficulties.

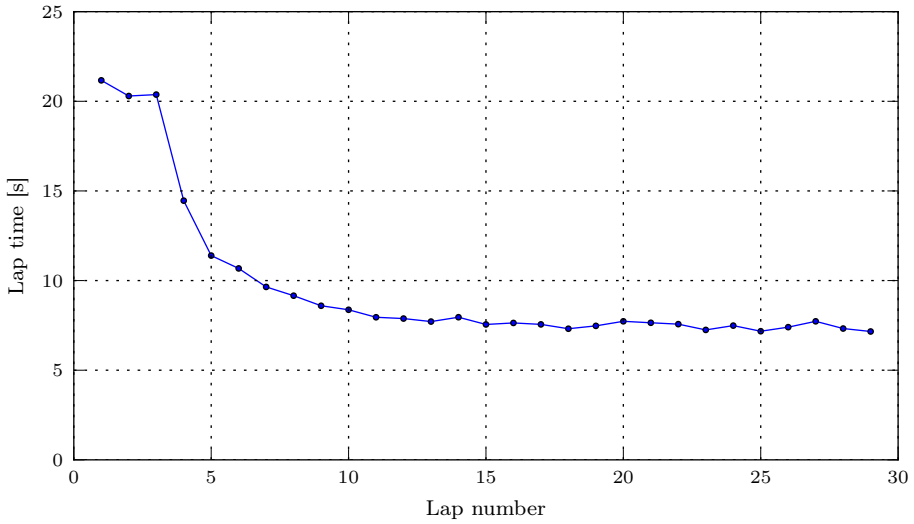

Figure 4.10: Lap times

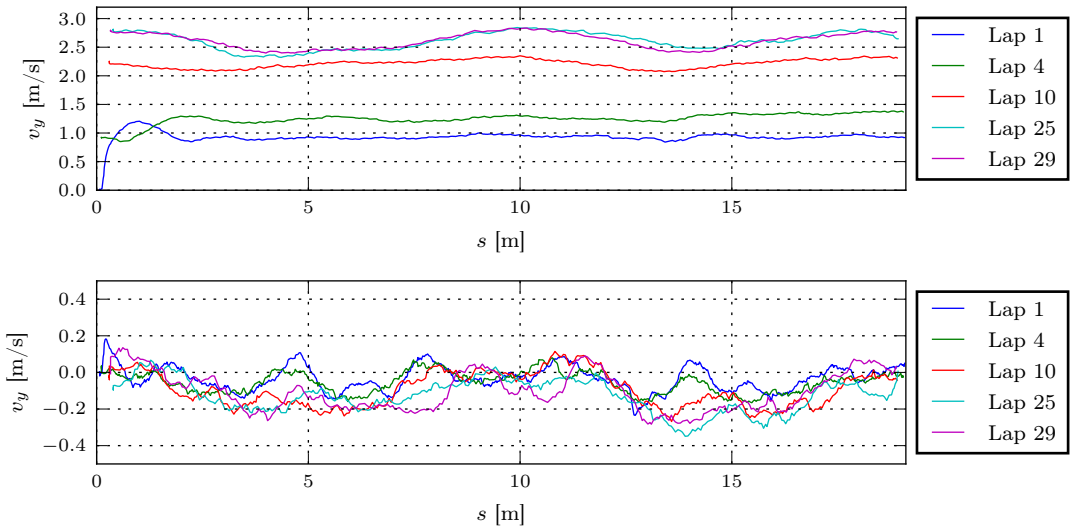

Figure 4.11: Longitudinal and lateral velocity

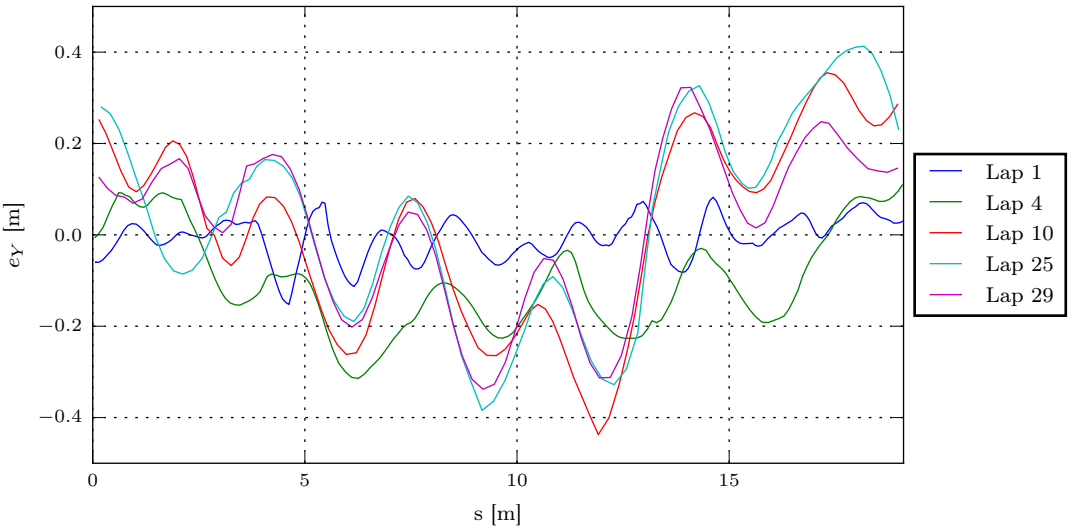

Figure 4.12: Lateral deviation

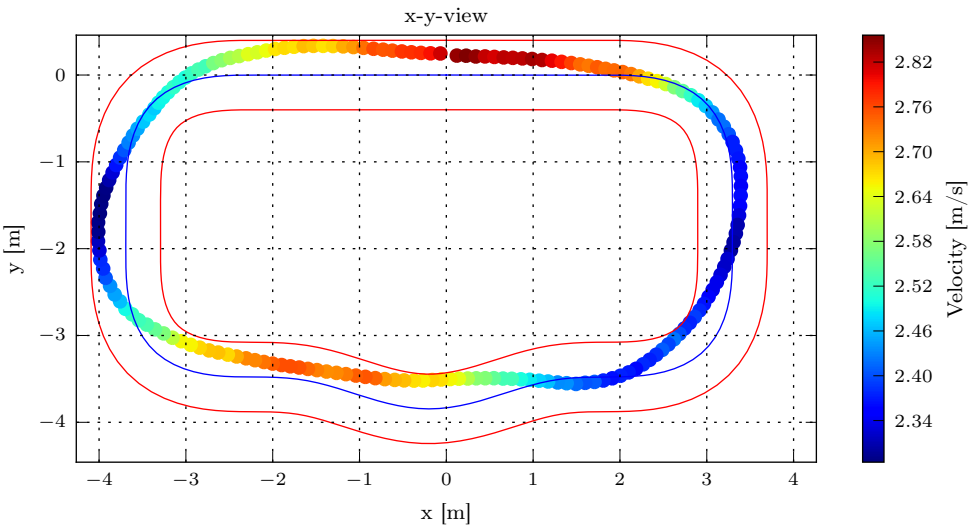

Figure 4.13: Velocity over x,y

Chapter 5

Conclusion and Outlook

5.1 Conclusion

The goal of this project was to extend the existing ILMPC strategy to repetitive tasks and test this theory on the race driving problem.
We were able to show that - by only little extensions - the ILMPC strategy is also applicable to continuous repetitive systems. The resulting controller is feasible and stable (but not necessarily nondecreasing/yields under certain assumptions non-decreasing iteration cost). 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 learnt 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.

5.2 Outlook and future work

(We were not able to prove non-decreasing cost...)
While we were assuming no model mismatch in simulations of the LMPC strategy, we clearly had some model mismatch in the 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 can be 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).

In order to improve lap times of the BARC experimental setup even more and to allow for 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 [4]). 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] Ugo Rosolia and Francesco Borrelli. Learning Model Predictive Control for Iterative Tasks. (4), 2016.

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

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

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

[7] 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.

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

[9] 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.

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

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

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

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

[14] 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.

[15] 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.

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

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

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