Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2021

Real-time Model Predictive Control with Complexity Guarantees Applied on a Truck and Trailer System

Edvin Bourelius

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

Real-time Model Predictive Control with Complexity Guarantees Applied on a Truck and Trailer System

Edvin Bourelius

LiTH-ISY-EX--22/5454--SE

Supervisor: **Daniel Arnström**
ISY, Linköping University

Examiner: **Daniel Axehill**
ISY, Linköping University

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

In model predictive control an optimization problem is solved in every time step, which in real-time applications has to be solved within a limited time frame. When applied on embedded hardware in fast changing systems it is important to use efficient solvers and crucial to guarantee that the optimization problem can be solved within the time frame.

In this thesis a path following controller which follows a motion plan given by a motion planner is implemented to steer a truck and trailer system. To solve the optimization problems which in this thesis are quadratic programs the three different solvers DAQP, qpOASES and OSQP are employed. The computational time of the active-set solvers DAQP, qpOASES and the operator splitting solver OSQP are compared, where the controller using DAQP was found the fastest and therefore most suited to use in this application of real-time model predictive control.

A certification framework for the active-set method is used to give complexity guarantees on the controller using DAQP. The exact worst-case number of iterations when the truck and trailer system is following a straight path is presented. Furthermore, initial experiments show that given enough computational time/power the exact iteration complexity can be determined for every possible quadratic program that can appear in the controller.

Acknowledgments

First of all, I would like to thank my supervisor Daniel Arnström for his feedback and guidance throughout this thesis. I also would like to thank my examiner Daniel Axehill for the opportunity to do this thesis.

Linköping, December 2021
Edvin Bourelius

Contents

Notation **ix**

1 Introduction **1**
1.1 Background and motivation 1
1.2 Problem formulation 1
1.3 Delimitations 2
1.4 System overview 3
1.4.1 Path planner 3
1.5 Outline 4

2 Path following error model for a general 2-trailer with a car-like tractor **5**
2.1 Vehicle model 5
2.2 Path-following control 7
2.3 Path-following error model 7
2.4 Linearized model 8
2.4.1 Straight-path linear model 10

3 Model Predictive Control **13**
3.1 Optimal control 13
3.2 Linear MPC 14
3.2.1 Designing terminal cost 15
3.2.2 MPC algorithm 15
3.3 MPC as a quadratic program 15
3.3.1 Soft constraints 17

4 Quadratic Programming **19**
4.1 Karush-Kuhn-Tucker conditions 19
4.2 Active-set method 20
4.2.1 Active-set algorithm 21
4.3 Conversion to the dual problem 21
4.4 Warm starts 22
4.5 Complexity certification 22
4.6 Parametric quadratic programming method 22

4.7 Operator splitting method 23

5 Implementation **25**
5.1 Control problem formulation 25
5.1.1 Terminal cost 26
5.1.2 Penalizing rate-of-chance 26
5.2 Constraints 27
5.2.1 Constraints in QP form 28
5.3 Creating mpQP for certification 29

6 Results **31**
6.1 Comparing linearization model and straight-path model 32
6.2 Performance 33
6.2.1 Distance from reference 33
6.2.2 Joint angles 34
6.2.3 Solver performance 35
6.2.4 Cost difference 37
6.2.5 Warm starts 37
6.3 Performance with tighter constraints 38
6.3.1 Joint angles 38
6.3.2 Solution performance 39
6.4 Certification 40
6.4.1 Straight path 41
6.4.2 Primitives 42

7 Conclusions **45**
7.1 Conclusions 45
7.2 Future work 46

Bibliography **47**

Notation

Abbreviations

Notation	Meaning
MPC	Model predictive control
OCP	Optimal control problem
QP	Quadratic program(ming)
KKT	Karush-Kuhn-Tucker
mpQP	Multi-parametric quadratic program(ming)
DARE	Discrete algebraic Riccati equation

Operators

Notation	Meaning
$[\cdot]_i$	ith row of vector or matrix
$\mathbf{diag}(A, B, C)$	Block diagonal matrix containing blocks A, B and C.

1 Introduction

1.1 Background and motivation

Model predictive control (MPC) is a powerful control strategy that can easily handle multi-variable system while taking constraints of both system states and control actions into account. Because of its strengths and conceptual simplicity MPC has recently become a wide spread control strategy for complex systems [16]. Also, recent developments in software and hardware has made it possible to apply MPC on embedded hardware, for example in the automotive industry [9, 10].

MPC uses a model of the system to predict future states. These predictions are then used to compute an optimal control signal by solving an optimization problem, which in this thesis is a quadratic program(QP). This optimization problem is solved in every time step and in real-time applications this optimization problem has to be solved within a limited time frame. When implementing the MPC on embedded hardware the time frame in which the optimization problem can be reasonably solved is highly dependant on the computational resources available. Hence, to be able to solve the problem within the time frame, it is not only important to employ efficient solvers for the optimization but also critical to certify that it is possible to solve the problem fast enough.

1.2 Problem formulation

In this thesis MPC will be applied to steer a truck and trailer system and the purpose of this thesis is in three parts; implementation of a real-time MPC, applying efficient QP solvers to solve the MPC problem and, finally, to give guarantees on

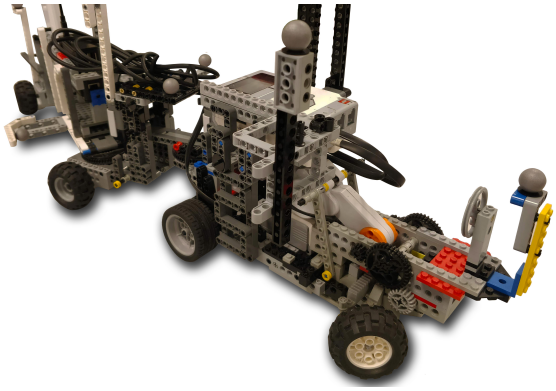

Figure 1.1: *Lego truck*

the complexity of the QP problems.

The vehicle in which the MPC is applied is a LEGO-truck including onboard RPi which will run the MPC and an onboard LEGO EV3 that runs the motor and low level control of the wheels. The truck can be seen in Figure 1.1.

There exist several different methods and solvers suited for real time MPC, for example active-set methods [2, 4, 11], interior point methods [12, 23] and gradient projection methods [3, 18]. In this thesis the active-set solvers DAQP [2] and qpOASES [11] together with the operator splitting method OSQP[22] will be applied to solve the MPC problem.

For some interior-point and gradient-projection methods it is possible to get bounds on the computational complexity [13, 19]. However in this thesis the certification framework [1] will be used to guarantee the **exact** computational complexity of the MPC using DAQP.

The investigations that will be performed in this thesis can be summarized with the following questions.

- Can a real-time MPC be applied to control the the vehicle?
- Which of the solvers DAQP, qpOASES or OSQP are most suited for this application of real time MPC?
- What is the guaranteed maximum computational complexity of the MPC?

1.3 Delimitations

The focus of this thesis is not to find the best performing MPC design or tuning. Instead the focus is to apply and compare the performance of the three different solvers when solving the MPC problem, while also guaranteeing the maximum computational complexity of the MPC.

1.4 System overview

There are three main modules which make up the autonomous vehicle system considered in this thesis, the state estimation, path planning and path following control. A system overview can be seen in Figure 1.2. The system is build with the Robot Operating System (ROS) [21] and the modules are created using ROS-nodes written in C++.

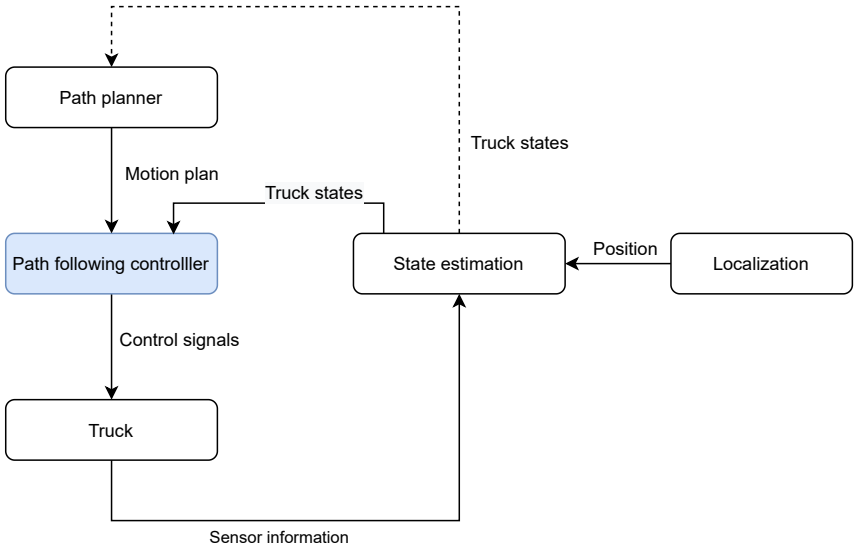

Figure 1.2: *System overview of the autonomous truck and trailer system*

The module considered in this thesis is the path following controller (highlighted in blue in Figure 1.2). The controller receives a motion plan from the path planner and the controller's mission is to follow this plan. The path planner takes the dynamics and constraints of the vehicle in consideration to produce a path that is feasible for the vehicle to follow.

The motion plan consists of a sequence of desired states, x_r, and controls, u_r. That is, the planner produces the optimal feed-forward steering angle of the vehicle, taking the dynamics of the vehicle into account. Using the path and feed-forward control, combined with the current state of the truck, the controller produces the feedback control which stabilizes the vehicle around the path.

1.4.1 Path planner

How the planner creates this motion plan is not within the scope of this thesis but some knowledge of how the planner works is needed to understand parts of it. To learn more about the techniques used in the motion planner, see [14] and [6]. The path planner uses a so-called *lattice-planner*. Given some goal, the

lattice planner combines *motion primitives* to build the motion plan offline before the vehicle is moving. However, when the vehicle starts moving and follows the motion plan an *improvement step* is made to iteratively update and improve the plan to the controller.

1.5 Outline

The outline of this thesis is

- **Chapter 2 - Path following error model for a general 2-trailer with a car-like tractor** describes the vehicle and the path following error model used in this thesis.
- **Chapter 3 - Model predictive control** gives an overview of the control method used to control the vehicle.
- **Chapter 4 - Quadratic programming** discussed how QPs can be solved.
- **Chapter 5 - Implementation** takes the vehicle model and techniques presented in previous chapters and describes the implementation of the MPC controller used to steer the vehicle.
- **Chapter 6 - Results** presents the tests and result of the developed controller showing path-following performance, solution performance for the QP solvers and finally the guaranteed complexity of the MPC.
- **Chapter 7 - Conclusions** presents future work and conclusions drawn from the result.

2

Path following error model for a general 2-trailer with a car-like tractor

This chapter presents the kinematic vehicle model of the truck and describes the path following error model used to stabilize the vehicle around the planned path.

2.1 Vehicle model

The vehicle has three parts, the car-like tracktor, a dolly and a semitrailer. The state vector $x = [x_3\ y_3\ \theta_3\ \beta_3\ \beta_2]^T$ is the representation of the configuration of the vehicle. Where (x_3, y_3) is the position of the rear axle, θ_3 is the orientation of the trailer, β_3 is the joint angle between the semitrailer and the dolly and β_3 is the joint angle between the dolly and the tractor. Finally the control is given by $u(t) = \frac{\tan(\alpha(t))}{L_1}$, where α is the steering angle, see Figure 2.1.

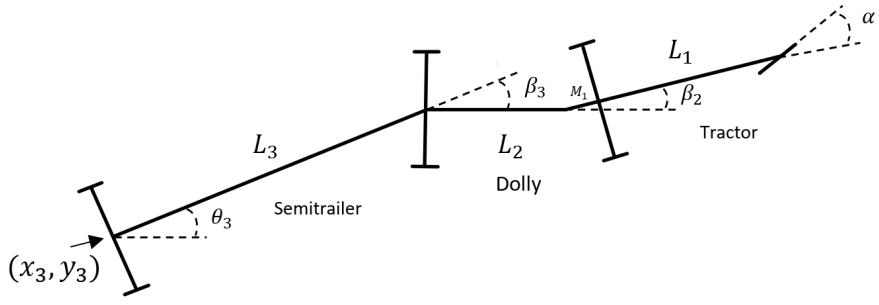

Figure 2.1: *Vehicle model*

There are four different parameters. L_1, L_2, L_3 and M_1, which describe the length of the different parts of the vehicle and can be used to describe a variety of different 2-trailer vehicles, the specific values used for the vehicle in this thesis are presented in Table 2.1.

Table 2.1: *A description of the different vehicle parameters.*

State	Description	Length [m]
L_1	The length of the wheelbase of the truck.	0.19
L_2	The length of the dolly.	0.135
L_3	The length of the trailer.	0.3
M_1	The length between the truck's rear wheels and the dolly's off axle hitch connection.	0.05

A kinematic model which describes the vehicle during low-speed maneuvers, presented in [15], is given by

$$\dot{x}_3 = v_3 \cos\theta_3 \tag{2.1a}$$
$$\dot{y}_3 = v_3 \sin\theta_3 \tag{2.1b}$$
$$\dot{\theta}_3 = v_3 \frac{\tan\beta_3}{L_3} \tag{2.1c}$$
$$\dot{\beta}_3 = v_3 \left(\frac{\sin\beta_2 - M_1 \cos\beta_2 u}{L_2 C_1(\beta_2, \beta_3, u)} - \frac{\tan\beta_3}{L_3} \right) \tag{2.1d}$$
$$\dot{\beta}_2 = v_3 \left(\frac{L_2 u - \sin\beta_2 + M_1 \cos\beta_2 u}{L_2 C_1(\beta_2, \beta_3, u)} \right) \tag{2.1e}$$

where $C_1(\beta_2, \beta_3, u) = \cos\beta_3(\cos\beta_2 + M1 \sin\beta_2 u)$, which is used to describe the relationship of the velocity of the semitrailers axle, v_3 and the velocity of the trackor, v, $v_3 = vC_1(\beta_2, \beta_3, u)$.

The model (2.1) assumes that the wheels roll without slipping and assumes that the vehicle is driving on a flat surface.

The stability of the system is highly dependant on the direction of travel, that is, if the vehicle is reversing ($v < 0$) or driving forward ($v > 0$). When reversing the joint-angles kinematic are unstable and there is a high risk of *jack-knifing* [15], which is when the tractor or dolly and the semitrailer fold over like a jack-knife. Conversely the system is stable when driving forward.

In short the kinematic model is represented as

$$\dot{x} = v_3 f(x, u). \tag{2.2}$$

2.2 Path-following control

The objective of the path-following controller is to stabilize the vehicle around the path computed by the motion planner. That is, given a path $(x_r(s), u_r(s))$, the controller should minimize the path-following error $\tilde{x} = x(t) - x_r(s(t))$, where the parameter $s(t)$ is the progression along the path. The nominal path is the obtained reference from the motion planner, which consists of a sequence of states $x_r(s)$ and control inputs $u_r(s)$

2.3 Path-following error model

In order to be able to minimize the path following error, a path following error model is used. This section will present this model. For a derivation of the model from (2.1) and details, see [15].

To describe the system as deviation from the nominal path an error state $\tilde{x} = [\tilde{z}_3 \ \tilde{\theta}_3 \ \tilde{\beta}_3 \ \tilde{\beta}_2]^T$ is used. Given the path $(x_r(s), u_r(s))$, which can be seen as a sequence of nominal vehicle configurations, the error state is the errors of the vehicle with respect to the corresponding nominal vehicle, see Figure 2.2.

The first state is the signed lateral distance $\tilde{z}_3$. The orientation error of the trailer is $\tilde{\theta}_3 = \theta_3(t) - \theta_{3r}(s(t))$. The joint angle error between the truck and dolly $\tilde{\beta}_2 = \beta_2(t) - \beta_{2r}(s(t))$ and $\tilde{\beta}_3 = \beta_3(t) - \beta_{3r}(s(t))$ is the joint angle error between the dolly and semitrailer. The control is defined as the curvature deviation $\tilde{u} = u(t) - u_r(s(t))$, where $u_r(s(t)) = \frac{\tan(\alpha_r(s(t)))}{L_1}$.

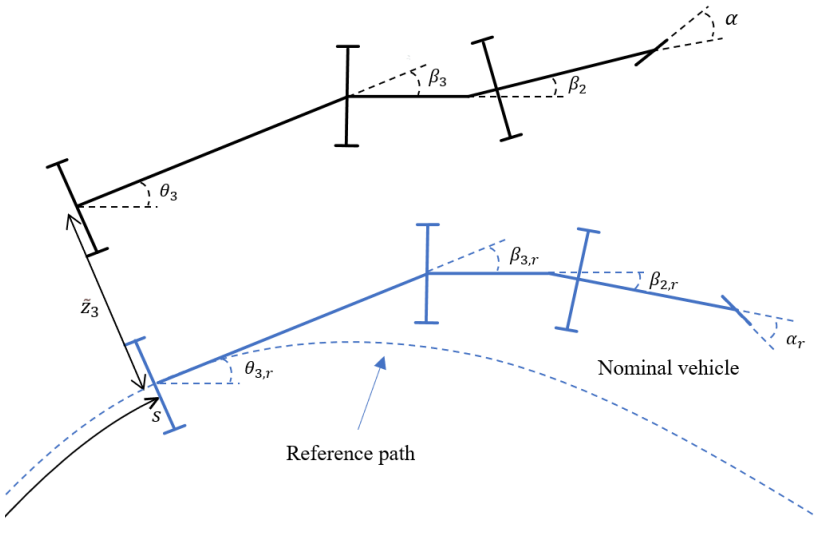

Figure 2.2: *Nominal and perturbed vehicle configuration*

Now the deviation from the nominal path is modeled as

$$\frac{d\tilde{z}_3}{ds} = \bar{v}_3(1 - \kappa_{3r}\tilde{z}_3)\tan(\tilde{\theta}_3) \tag{2.3a}$$

$$\frac{d\tilde{\theta}_3}{ds} = \bar{v}_3\left((1 - \tilde{z}_3\kappa_{3_r})\frac{\tan(\tilde{\beta}_3 + \beta_{3r})}{L_3\cos(\tilde{\theta}_3)} - \kappa_{3_r}\right) \tag{2.3b}$$

$$\frac{d\tilde{\beta}_3}{ds} = \frac{\bar{v}_3(1 - \kappa_{3r}\tilde{z}_3)}{\cos(\tilde{\theta}_3)}\left(\frac{\sin(\tilde{\beta}_2 + \beta_{2r}) - M_1(\tilde{u} + u_{ref})\cos(\tilde{\beta}_2 + \beta_{2r})}{C_1L_2} - \frac{\tan(\tilde{\beta}_3 + \beta_{3r})}{L_3}\right) - \frac{d\beta_{3r}}{ds} \tag{2.3c}$$

$$\frac{d\tilde{\beta}_2}{ds} = \frac{\bar{v}_3(1 - \kappa_{3r}\tilde{z}_3)}{\cos(\tilde{\theta}_3)}\left(\frac{(L_2 - M_1)(\tilde{u} + u_{ref})\cos(\tilde{\beta}_2 + \beta_{2r}) + \sin(\tilde{\beta}_2 + \beta_{2r})}{L_2C_1}\right) - \frac{d\beta_{2r}}{ds}, \tag{2.3d}$$

where

$$\kappa_{3_r} = \frac{\tan(\beta_{3_r})}{L_3}.$$

Here $\bar{v}_3 = \text{sign}(v)$. Importantly the model (2.3) does not depend on the velocity (assuming the velocity is constants).

In short the error model in (2.3) is represented as

$$\frac{d\tilde{x}}{ds} = \tilde{f}(\tilde{x}, \tilde{u}, s). \tag{2.4}$$

2.4 Linearized model

In this thesis the control problem is an MPC problem that is converted into a QP problem. To be able to do the conversion into a QP problem the model needs to be linear. The path-following error model (2.3) is therefore linearized around the origin $(\tilde{x}, \tilde{u}) = (0, 0)$, giving the linear system

$$\frac{d\tilde{x}}{ds} = A(s)\tilde{x} + B(s)\tilde{u}, \tag{2.5}$$

with the reference-dependant matrices

$$A(s) = \frac{\partial\tilde{f}(0,0,s)}{\partial\tilde{x}} = \bar{v}_3\begin{bmatrix} 0 & 1 & 0 & 0 \\ a_{21} & 0 & a_{23} & 0 \\ a_{31} & 0 & a_{33} & a_{34} \\ a_{41} & 0 & a_{43} & a_{44} \end{bmatrix}, \quad B(s) = \frac{\partial\tilde{f}(0,0,s)}{\partial\tilde{u}} = \bar{v}_3\begin{bmatrix} 0 \\ 0 \\ b_3 \\ b_4 \end{bmatrix}. \tag{2.6}$$

The matrix coefficients are given by

$$a_{21} = \frac{\tan^2 \beta_{3r}}{L_3^2} \tag{2.7a}$$

$$a_{23} = \frac{1}{(L_3 \cos^2(\beta_{3r})} \tag{2.7b}$$

$$a_{31} = \frac{(\tan \beta_{3r})(u_r M_1 \cos \beta_{3r} - \sin \beta_{3r})}{L_3 L_2 \cos \beta_{3r}(\cos \beta_{3r} + u_r M_1 \sin \beta_{3r})} + \frac{\tan^2 \beta_{3r}}{L_3^2} \tag{2.7c}$$

$$a_{33} = \frac{(\sin \beta_{3r})(\sin \beta_{3r}) - u_r M_1 \cos \beta_{3r}))}{(L_2 \cos(\beta_{3r})^2(\cos(\beta_{3r}) + u_r M_1 \sin(\beta_{3r})))} - \frac{1}{L_3 \cos(\beta_{3r})^2} \tag{2.7d}$$

$$a_{34} = \frac{1 + u_r^2 M_1^2}{L_2 \cos(\beta_{3r})(\cos(\beta_{3r}) + u_r M_1 \sin(\beta_{3r}))^2} \tag{2.7e}$$

$$a_{41} = -\frac{\tan(\beta_{3r})}{L_3 L_2} \frac{(u_r L_2 - \sin(\beta_{3r}) + M_1 \cos(\beta_{3r} u_r))}{(\cos(\beta_{3r})(\cos(\beta_{3r}) + M_1 u_r \sin(\beta_{3r})))} \tag{2.7f}$$

$$a_{43} = \frac{\tan(\beta_{3r})}{L_2} \frac{(u_r L_2 + u_r M_1 \cos(\beta_{3r}) - \sin(\beta_{3r}))}{\cos(\beta_{3r})(\cos(\beta_{3r}) + M_1 u_r \sin(\beta_{3r}))} \tag{2.7g}$$

$$a_{44} = \frac{-(1 + u_r^2 M_1^2 + u_r^2) L_2 M_1 \cos(\beta_{3r}) - u_r L_2 \sin(\beta_{3r})}{L_2 \cos(\beta_{3r})(\cos(\beta_{3r}) + u_r M_1 sin(\beta_{3r}))^2} \tag{2.7h}$$

$$b_3 = -\frac{M_1}{L_2 \cos(\beta_{3r}) (M_1 u_r \sin(\beta_{2r}) + \cos(\beta_{2r}))^2} \tag{2.7i}$$

$$b_4 = \frac{L_2 \cos(\beta_{2r}) + M_1}{L_2 \cos(\beta_{3r}) (M_1 u_r \sin(\beta_{2r}) + \cos(\beta_{2r}))^2}. \tag{2.7j}$$

The MPC formulation that we will consider also requires discrete system dynamics. Therefore the linear system matrices are discretized through Euler forward approximation

$$F_k = I + \Delta_s A_k \tag{2.8a}$$

$$G_k = \Delta_s B_k. \tag{2.8b}$$

The discrete time dynamics of the vehicle is now given by

$$\tilde{x}_{k+1} = F_k \tilde{x}_k + G_k \tilde{u}_k. \tag{2.9}$$

The dynamics can be used iteratively to describe any state

$$\begin{aligned}
\tilde{x}_1 &= F_0 \tilde{x}_0 + G_0 \tilde{u}_0 \\
\tilde{x}_2 &= F_1 \tilde{x}_1 + G_1 \tilde{u}_1 = F_1 F_0 \tilde{x}_0 + F_1 G_0 \tilde{u}_0 + G_1 \tilde{u}_1 \\
&\vdots \\
\tilde{x}_k &= \prod_{i=0}^{k-1} F_i \tilde{x}_0 + \sum_{j=0}^{k-1} \prod_{i=0}^{k-1-j} F_i G_j \tilde{u}_j
\end{aligned}$$

To describe all states in a more condensed way the states and controls are stacked into vectors

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{x}_0 \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_N \end{bmatrix}, \qquad \tilde{\mathbf{u}} = \begin{bmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_{N-1} \end{bmatrix}. \tag{2.10}$$

Now the controls can be used to express the states in matrix form.

$$\tilde{\mathbf{x}} = \mathbf{F}\tilde{x}_0 + \mathbf{G}\tilde{\mathbf{u}} \tag{2.11}$$

with the matrices

$$\mathbf{F} = \begin{bmatrix} I \\ F_0 \\ F_1F_0 \\ F_2F_1F_0 \\ \vdots \\ F_{N-1}\dots F_1F_0 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ G_0 & 0 & \dots & 0 \\ F_1G_0 & G_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \\ F_{N-1}\dots F_0\,G_0 & F_{N-2}\dots F_0\,G_1 & \dots & G_{N-1} \end{bmatrix}. \tag{2.12}$$

In Chapter 3 these matrices will be used to create a quadratic program from the MPC problem.

2.4.1 Straight-path linear model

A special case of (2.6) is the straight-path reference configuration, that is, when $\beta_{3,r} = 0, \beta_{2,r} = 0$ and $u_r = 0$. The linear model matrices are then given by

$$A = \bar{v}_3 \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{L_3} & 0 \\ 0 & 0 & -\frac{1}{L_3} & \frac{1}{L_2} \\ 0 & 0 & 0 & -\frac{1}{L_2} \end{bmatrix}, B = \bar{v}_3 \begin{bmatrix} 0 \\ 0 \\ \frac{-M_1}{L_2} \\ \frac{L_2+M_1}{L_2} \end{bmatrix}, \bar{v}_3 \in \{-1, 1\}. \tag{2.13}$$

Now, the discrete model matrices are constant

$$F = I + \Delta_s A \tag{2.14a}$$

$$G = \Delta_s B \tag{2.14b}$$

and we get the dynamics

$$\tilde{x}_{k+1} = F\tilde{x}_k + G\tilde{u}_k. \tag{2.15}$$

Again, using the linear model recursively the state at time step k can be expressed as

$$\tilde{x}_k = F^k\tilde{x}_0 + \sum_{i=0}^{k-1} F^{k-1-i}G\tilde{u}_i. \tag{2.16}$$

Now the matrices in (2.12) are instead

$$\mathbf{F} = \begin{bmatrix} I \\ F \\ F^2 \\ \vdots \\ F^N \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ G & 0 & \dots & 0 \\ F\,G & G & \dots & 0 \\ \vdots & & \ddots & \\ F^{N-1}\,G & F^{N-2}\,G & \dots & G \end{bmatrix}. \tag{2.17}$$

Note that **F** and **G** are constant, which is of significance since they only have to be calculated once. In contrast to (2.12) which depends on the reference and, hence, needs to be computed at each time-step.

3

Model Predictive Control

The control strategy used in this thesis is model predictive control (MPC). As the name suggests, the core of MPC is to use a *model* to *predict* the future states of the system and thereafter choose the *control*. MPC is an optimal control strategy where the control is the solution to an optimization problem. This chapter introduces optimal control, presents the control strategy used in this thesis (discrete linear model predictive control) and concludes with showing how the MPC formulation can be cast as a quadratic program.

3.1 Optimal control

Often in practice, as is the case in this thesis, the states of a system is constrained by the system dynamics. Moreover the controls can not be chosen freely when the system states and control actions are constrained. Optimal control is a control strategy where the system dynamics, together with the state and control constraints, can be taken into consideration to produce an optimal control action, in some sense.

With the discrete time dynamics

$$x_{k+1} = f_d(x_k, u_k), \tag{3.1}$$

and the constraints of the states and controls,

$$u \in \mathbf{U}, x \in \mathbf{X}. \tag{3.2}$$

The discrete time optimal control problem is

$$\min_{\mathbf{x},\mathbf{u}} \quad V(\mathbf{x},\mathbf{u}) = V_N(x_N) + \sum_{k=0}^{N-1} l(x_k, u_k) \tag{3.3a}$$

$$\text{subject to} \quad x_{k+1} = f_d(x_k, u_k), \quad k = 0, ..., N-1 \tag{3.3b}$$

$$u_k \in \mathbf{U}, \quad k = 0, ..., N-1 \tag{3.3c}$$

$$x_k \in \mathbf{X}, \quad k = 1, ..., N \tag{3.3d}$$

$$x_0 = x. \tag{3.3e}$$

Hence, the objective is to minimize the so-called *objective function* $V(\mathbf{x},\mathbf{u})$ with the state and control sequences $\mathbf{x}$ and $\mathbf{u}$. Choosing the objective function is the users way of defining what states and controls are desirable. $V(\mathbf{x},\mathbf{u})$ is also known as the cost function and as often done in practice the cost function in decomposed into a *terminal cost* $V_N(x_N)$ and a *stage cost* $l(x_k, u_k)$. The integer N is called the *prediction horizon* which is a design variable that decides how many steps into the future the states are predicted.

3.2 Linear MPC

In this thesis an optimal control problem (OCP) in the form of (3.3a) is solved in every time instance. After the optimal control problem is solved only the first control u_0 in the sequence $\mathbf{u}$ is used as the control action. By recurrently solving the OCP, consequences beyond the horizon can be accounted for in later iterations. Secondly, model errors or disturbances can lead to deviations from the desired state trajectory, which can be counteracted when re-solving the OCP and the control is recomputed, leading to robust feedback behaviour. This strategy is what makes MPC powerful.

The focus of this thesis is to use linear discrete time dynamics.

$$x_{k+1} = f_d(x_k, u_k) = Fx_k + Gu_k, \quad F \in \mathbb{R}^{n_x \times n_x} \text{ and } G \in \mathbb{R}^{n_x \times n_u}. \tag{3.4}$$

A common way to design the cost function is to use a quadratic cost [7]. That is

$$V_N(x_N) = \|x_N\|_{P_N} = x_N^T P_N x_N, \tag{3.5a}$$

$$l(x_k, u_k) = \|x_k\|_Q^2 + \|u_k\|_R^2 = x_k^T Q x_k + u_k^T R u_k, \tag{3.5b}$$

where $P_N \succeq 0$, $Q \succeq 0$ and $R \succ 0$ are design matrices enabling tuning of which states are more or less important and which controls are more or less expensive.

It is also common to use polyhedral constraints [7]. That is we assume the constraints on states and controls have the polyhedral form

$$A_z x_k + A_u u_k \leq b, \quad k = 0, \ldots, N-1 \tag{3.6a}$$

$$A_z x_N \leq b_N. \tag{3.6b}$$

Now, using (3.4), (3.5) and (3.6) the *linear discrete time MPC problem* is formulated as

$$\min_{\mathbf{x},\mathbf{u}} \quad V_N(x_N) = x_N^T P_N x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k \tag{3.7a}$$

$$\text{subject to} \quad x_{k+1} = Fx_k + Gu_k, \quad k = 0, \ldots, N-1 \tag{3.7b}$$

$$A_z x_k + A_u u_k \leq b, \quad k = 0, \ldots, N-1 \tag{3.7c}$$

$$A_z x_N \leq b_N, \tag{3.7d}$$

$$x_0 = x. \tag{3.7e}$$

Important to note is that the initial state in the MPC x_0 is constrained to the current state x. Hence, the MPC is dependent on the current state and MPC can be seen as a map from the current state to optimal control $u_0^*(x)$.

3.2.1 Designing terminal cost

Ideally the horizon N would be chosen such that $N \to \infty$. This is obviously not possible in practice. However, by choosing P_N wisely, it is possible to emulate infinite horizon cost by approximating the remaining running cost after the horizon. Choosing P_N as the solution to the discrete-time algebraic Riccati equation (DARE):

$$P = F^T PF - (F^T PG)(R + G^T PG)^{-1}(G^T PF) + Q,$$

that is, the optimal cost of the infinite horizon Linear Quadratic problem, approximates the remaining running cost after the horizon with the optimal cost for the infinite horizon Linear Quadratic problem [14].

3.2.2 MPC algorithm

The MPC strategy presented thus far can be summarized into Algorithm 1, which describes how MPC is implemented.

Algorithm 1 Model predictive control

1: Measure or estimate x_0
2: Compute $\mathbf{u}$ by solving (3.7)
3: Use first element u_0 of sequence $\mathbf{u}$ as the control action
4: Time update, $k = k + 1$
5: Go to 1:

3.3 MPC as a quadratic program

What remains to address is how the OCP in Step 2 in Algorithm 1 is solved. This is done by first casting (3.7) as a Quadratic Programming problem also known as

a Quadratic Program(QP). Using the design choices that were made in Section 3.2, that is, using linear time discrete dynamics $x_{k+1} = Fx_k + Gu_k$ and using quadratic cost with matrices $P_N \succeq 0$, $Q \succeq$ and $R \succ 0$ the MPC can be formulated as a QP. This section will derive how to do this.

A typical QP problem formulation and the formulation used is

$$\min_x \quad \frac{1}{2}x^T Hx + f^T x \tag{3.8a}$$
$$\text{subject to} \quad Ax \leq b. \tag{3.8b}$$

Starting with the MPC objective function. Using (2.11) the objective function can be expressed in a condensed way

$$x_N^T P_N x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k = \tag{3.9}$$
$$= \mathbf{x}^T \mathbf{Q}\mathbf{x} + \mathbf{u}^T \mathbf{R}\mathbf{u} \tag{3.10}$$
$$= (\mathbf{F}x_0 + \mathbf{G}\mathbf{u})^T \mathbf{Q}(\mathbf{F}x_0 + \mathbf{G}\mathbf{u}) + \mathbf{u}^T \mathbf{R}\mathbf{u} \tag{3.11}$$
$$= \mathbf{x}_0^T \mathbf{F}^T \mathbf{Q}\mathbf{F}\mathbf{x}_0 + 2\mathbf{x}_0^T \mathbf{F}^T \mathbf{Q}\mathbf{G}\mathbf{u} + \mathbf{u}^T \mathbf{G}^T \mathbf{Q}\mathbf{G}\mathbf{u} + \mathbf{u}^T \mathbf{R}\mathbf{u}, \tag{3.12}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are block-diagonal matrices containing the design matrices, $\mathbf{Q} = \mathbf{diag}(Q, ..., Q, P_N)$ and $\mathbf{R} = \mathbf{diag}(R, ..., R)$.

The term $\mathbf{x}_0^T \mathbf{F}^T \mathbf{Q}\mathbf{F}\mathbf{x}_0$ does not depend on $\mathbf{u}$ and hence can be excluded. Now dividing by 2 the MPC objective function can now be expressed in the form of (3.8a).

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T(\mathbf{G}^T \mathbf{Q}\mathbf{G} + \mathbf{R})\mathbf{u} + x_0^T \mathbf{F}^T \mathbf{Q}\mathbf{G}\mathbf{u} \tag{3.13}$$

Now consider the constraints

$$A_z x_k + A_u u_k \leq b, \quad k = 0, \ldots, N-1 \tag{3.14}$$
$$A_z x_N \leq b. \tag{3.15}$$

Again, with the states $\mathbf{x}$ and controls $\mathbf{u}$ from (2.10), these can be expressed in matrix form.

$$\mathbf{A}_z \mathbf{x} + \mathbf{A}_u \mathbf{u} \leq \mathbf{b}$$

where

$$\mathbf{A}_z = \mathbf{diag}(A_z, ..., A_z, A_z), \quad \mathbf{A}_u = \begin{bmatrix} \mathbf{diag}(A_u, ..., A_u) \\ 0 \end{bmatrix}.$$

Moreover (2.11) is used to express the constraints in a condensed way.

$$\begin{aligned}\mathbf{A}_z\mathbf{x} + \mathbf{A}_u\mathbf{u} &= \\ \mathbf{A}_z(\mathbf{F}x_0 + \mathbf{G}\mathbf{u}) + \mathbf{A}_u\mathbf{u} &= \\ \mathbf{A}_z\mathbf{F}x_0 + \mathbf{A}_z\mathbf{G}\mathbf{u} + \mathbf{A}_u\mathbf{u} &\leq \mathbf{b} \\ &\Leftrightarrow \\ (\mathbf{A}_z\mathbf{G} + \mathbf{A}_u)\mathbf{u} &\leq \mathbf{b} - \mathbf{A}_z\mathbf{F}x_0.\end{aligned}$$

Now the complete QP problem can be expressed as

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T(\mathbf{G}^T\mathbf{Q}\mathbf{G} + \mathbf{R})\mathbf{u} + x_0^T\mathbf{F}^T\mathbf{Q}\mathbf{G}\mathbf{u} \tag{3.16a}$$

$$\text{subject to} \quad (\mathbf{A}_z\mathbf{G} + \mathbf{A}_u)\mathbf{u} \leq \mathbf{b} - \mathbf{A}_z\mathbf{F}x_0 \tag{3.16b}$$

and it is possible to identify the QP matrices

$$H = (\mathbf{G}^T\mathbf{Q}\mathbf{G} + \mathbf{R}), \tag{3.17a}$$

$$f^T = x_0^T\mathbf{F}^T\mathbf{Q}\mathbf{G}, \tag{3.17b}$$

$$A = (\mathbf{A}_z\mathbf{G} + \mathbf{A}_u) \tag{3.17c}$$

$$b = \mathbf{b} - \mathbf{A}_z\mathbf{F}x_0 \tag{3.17d}$$

3.3.1 Soft constraints

To ensure that the QP is always feasible, the state constraints are usually softened [24]. This is done by adding the optimization variable ϵ_s to the constraints as

$$Ax \leq b \rightarrow Ax \leq b + S\epsilon_s,$$

where S is a selection matrix deciding to which rows ϵ_s is added, that is, which constraints are softened. Then, to minimize the violation of the constraints the term $\rho\epsilon_s$ is added to the objective function, where ρ is its weight. The QP becomes

$$\min_{x,\epsilon_s} \quad \begin{bmatrix} x \\ \epsilon_s \end{bmatrix}^T \begin{bmatrix} H & 0 \\ 0 & \rho \end{bmatrix} \begin{bmatrix} x \\ \epsilon_s \end{bmatrix} + \begin{bmatrix} f \\ 0 \end{bmatrix}^T \begin{bmatrix} x \\ \epsilon_s \end{bmatrix}$$

$$\text{subject to} \quad \begin{bmatrix} A & -S \end{bmatrix} \begin{bmatrix} x \\ \epsilon_s \end{bmatrix} \leq b.$$

4

Quadratic Programming

In Chapter 3 we saw how the linear discrete time MPC problem could be cast as a QP. This chapter will present methods on how to solve this QP, although the methods will not be described in detail. Initially the Karush-Kuhn-Tucker (KKT) conditions will be presented, followed by overviews of the active-set method and the operator splitting method which are the methods the solvers used in this thesis use. Finally how the complexity of active-set algorithms can be certified is presented.

4.1 Karush-Kuhn-Tucker conditions

Necessary conditions for optimally in nonlinear optimization are the Karush-Khun-Tucker (KKT) conditions. Assuming x^* is the solution to (3.8), the KKT-conditions are

$$Hx^* + A^T\lambda = -f \tag{4.1a}$$
$$Ax^* \leq b \tag{4.1b}$$
$$\lambda \geq 0 \tag{4.1c}$$
$$([b]_i - [A]_i x^*)[\lambda]_i = 0, \forall i \in \mathcal{N}_m, \tag{4.1d}$$

where λ is the dual variables, also known as the Lagrange multipliers, and $[\,\cdot\,]_i$ denotes the ith row in respective matrix or vector. By using quadratic cost and positive definite design matrices in the MPC, as done in Section 3.2 the created QP is convex, and for convex problems the KKT-conditions are also sufficient for optimality [8]. The conditions (4.1a), (4.1b) and (4.1c) ensure that the negative gradient at x^* is perpendicular to the feasible set and are called *stationary-*, *dual*

feasibility -, and *complementary slackness* condition. (4.1b) is called the *primal feasibility* condition because it ensures primal feasibility.

So, when the QP is convex, finding x^* that satisfies the KKT conditions means the optimal solution to the QP has been found. However, since (4.1d) is nonlinear, finding this x^* is non-trivial.

4.2 Active-set method

An easier problem to solve would be solving the *equality constrained quadratic program* (EQP)

$$\min_x \quad \frac{1}{2}x^T Hx + f^T x \tag{4.2a}$$
$$\text{subject to} \quad Ex = d, \tag{4.2b}$$

This is easier because if all constraints are equalities the complementary slackness condition (4.1d) can be considered trivially satisfied. This means that only a linear system of equations

$$\begin{bmatrix} H & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda \end{bmatrix} = \begin{bmatrix} -f \\ d \end{bmatrix}, \tag{4.3}$$

has to be solved [8]. Active-set methods exploit this and instead of solving (3.8) directly, they solve a sequence of problems in the form (4.2a).

Let $\mathcal{A}(x)$ be the set of all inequality constraints in (3.8b) that hold with equality at x, also known as *active constraints*. That is, all rows i where $[A]_i x = [b]_i$, or stacking all rows together $[A]_{\mathcal{A}} x = [b]_{\mathcal{A}}$. This set $\mathcal{A}(x)$ is known as the *active set* at x. Furthermore assuming x^* is a solution to (3.8) it is also a solution to

$$\min_x \quad \frac{1}{2}x^T Hx + f^T x \tag{4.4a}$$
$$\text{subject to} \quad [A]_i x = [b]_i, \quad \forall i \in \mathcal{A}(x^*). \tag{4.4b}$$

This can be confirmed by the KKT conditions. The complementary slackness gives $[\lambda]_i = 0, \forall i \notin \mathcal{A}(x^*)$, hence,

$$Hx^* + [A]^T_{\mathcal{A}(x^*)}[\lambda]_{\mathcal{A}(x^*)} = -f. \tag{4.5}$$

By definition

$$[A]_{\mathcal{A}(x^*)} x^* = [b]_{\mathcal{A}(x^*)}.$$

and (4.1d) is automatically fulfilled. We then get the linear system

$$\begin{bmatrix} H & [A]^T_{\mathcal{A}(x^*)} \\ [A]_{\mathcal{A}(x^*)} & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda \end{bmatrix} = \begin{bmatrix} -f \\ d \end{bmatrix}, \tag{4.6}$$

which is the solution to the EQP. Hence, if $\mathcal{A}(x^*)$ is found it is only necessary to solve (4.6) to find the optimal solution x^*. Therefore the goal of an active-set method is to find the active set $\mathcal{A}(x^*)$.

In these methods, $\mathcal{A}(x^*)$ is found by iteratively updating a "guess" of the active set called the *working set*, $\mathcal{W}$. There are several different ways to update $\mathcal{W}$, however this will not be covered in detail in this thesis. For details see [2] and [11].

4.2.1 Active-set algorithm

The active-set method can be summarised by a prototypical algorithm

Algorithm 2 Active-set method

1: **While** true **do**
2: (x,λ) $\leftarrow$ Solve KKT system (4.3)
3 : **if**(x, λ) is primal and dual feasible **then**
4 : **return**(x^*, λ^*, A^*) $\leftarrow$ (x, λ, $\mathcal{W}$)
5 : **else**
6 : Update $\mathcal{W}$ based on primal and/or dual violation, i.e., $Ax \leq b$ or $\lambda \geq 0$

In the last step in Algorithm 2 constraints are added or removed to $\mathcal{W}$, which can be done in different ways. In this thesis two methods are covered; DAQP uses the dual method [2] and qpOASES the parametric method [11].

4.3 Conversion to the dual problem

The dual method involves solving the so called *dual problem*

$$\min_{\lambda} \quad \frac{1}{2}\lambda^T MM^T \lambda + d^T \lambda \tag{4.7a}$$

$$\text{subject to} \quad \lambda \geq 0, \tag{4.7b}$$

where the conversion from the primal problem (3.8) is made by solving the equation systems

$$R^T M^T = A^T, \quad R^T v = f \tag{4.8}$$

for M and v, and then calculating

$$d = b + Mv \tag{4.9}$$

where R is the upper Cholesky factor of H, i.e. $H = R^T R$. When the optimal dual solution to (4.7a), λ^*, has been found the primal solution can be acquired by solving the triangular system

$$Rx^* = -(\lambda^* + v). \tag{4.10}$$

Since R is a triangular matrix the equation systems in (4.8) and (4.10) can be solved efficiently [20].

4.4 Warm starts

Often in linear MPC, the neighboring MPC problems from one iteration to the next are very similar. For example, this is the case when using the straight path model since H and A are constant and only the vectors f and b need to be updated. This means that the solutions to the corresponding QPs often are close. So, when solving every QP except the first one, some QP solvers can be initialized with the previous solution, which often reduces the computations needed [17].

4.5 Complexity certification

Recall the important insight from Section 3.2 that the MPC can be seen as a map from the current state $x = x_0$ to the optimal solution $u^*(x_0)$. This map is evaluated by the optimization algorithm that solves the QP that is made from the MPC problem. Hence, from every possible initial state we get a different QP. Defining Θ_0 as the space of all possible initial states and introducing the parameter $\theta \in \Theta_0$, we get a so called *multi-parametric quadratic program* (mpQP)

$$\min_x \quad \frac{1}{2}x^T H x + \theta^T f_\theta^T x \tag{4.11a}$$

$$\text{subject to} \quad Ax \leq b + W\theta. \tag{4.11b}$$

As shown in [1] solutions to the KKT-system that appear in iteration k using the active set algorithm are affine in this parameter θ, that is,

$$x_k^* = F_k^* \theta + G_k^*. \tag{4.12}$$

For details how F_k^* and G_k^* are calculated, see [1]. This affine structure is exploited in explicit MPC [5] to pre-compute all solutions $u^*(\theta)$ offline, and can also be used to certify the complexity of active-set algorithms[1].

An interpretation of the complexity certification method in [1] is that, using the affine property, iterations of the active-set algorithm are executed parametrically and the parameter space Θ_0 is recursively partitioned into a finite number of regions. The working set changes made for each region are saved and all parameters θ in the same region signify that they produce the same working-set changes before finding $\mathcal{A}(x^*)$. Since the exact sequence of working set changes for all θ is determined, we know exactly which EQPs are solved. This means that the exact behaviour, in terms of iterations and/or floating point operations, of the active-set method can be determined.

4.6 Parametric quadratic programming method

The method qpOASES uses is sometimes called the *parametric quadratic programming method*, this section will explain the core concept, for details see [11]. Similarly to Section 4.5 the parametric method uses a parameterized QP. The QP is

parameterized with the parameter $\tau \in [0, 1]$ and the QP is

$$\min_x \quad \frac{1}{2}x(\tau)^T H x(\tau) + f(\tau)^T x \tag{4.13a}$$

$$\text{subject to} \quad Ax(\tau) \leq b(\tau). \tag{4.13b}$$

The main idea is to trace the primal-dual solution $z^*(\tau) = (x^*(\tau), \lambda^*(\tau))$ from a QP with known optimal solution $z^*(0)$ to the optimal solution of the QP that needs to be solved $z^*(1)$ using a linear homotopy.

4.7 Operator splitting method

The solver OSQP unlike qpOASES and DAQP does not use the active-set algorithm, it uses a operator splitting method. This section will only explain the core concept, for a thorough explanation read [22]. OSQP iteratively, i.a, solves a linear system

$$\begin{bmatrix} H + \sigma I & A^T \\ A & -\rho^{-1} I \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - f \\ z^k - \rho^{-1} y^k \end{bmatrix}. \tag{4.14}$$

to each iteration obtain a tuple (x^k, z^k, y^k) until the residuals

$$r^k_{\text{prim}} = Ax^K - z^k, \quad r^k_{\text{dual}} = Hx^k + f + A^T y^k, \tag{4.15}$$

converge to specified tolerance levels

$$\|r^k_{\text{prim}}\| \leq \epsilon_{\text{prim}}, \quad \|r^k_{\text{dual}}\| \leq \epsilon_{\text{dual}}. \tag{4.16}$$

5
Implementation

Using the MPC design from Chapter 3 together with the path-following error model presented in Chapter 2, the implemented MPC controller can now be presented. The MPC formulation is presented in Section 5.1 with the imposed constraints discussed in Section 5.2. Finally the mpQP for the certification is identified in Section 5.3.

5.1 Control problem formulation

The goal of the MPC is to follow the path given by the planner. For this the path-following error model is convenient since the reference in built into the error model and the MPC only needs to minimize the error state $\tilde{x} = [\tilde{z}_3 \ \tilde{\theta}_3 \ \tilde{\beta}_3 \ \tilde{\beta}_2]^T$. That is, the MPC minimizes the sequence of error states and controls within the horizon N, that is, the vectors

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{x}_0 \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_N \end{bmatrix}, \qquad \tilde{\mathbf{u}} = \begin{bmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_{N-1} \end{bmatrix}. \tag{5.1}$$

The MPC can now be formulated as

$$\underset{\tilde{\mathbf{x}},\tilde{\mathbf{u}},\epsilon_\beta}{\text{minimize}}\ V_N(\tilde{\mathbf{x}},\tilde{\mathbf{u}}) = \tilde{x}_N^T P_N \tilde{x}_N + \sum_{k=0}^{N-1} \tilde{x}_k^T Q \tilde{x}_k + \tilde{u}_k^T \tilde{u}_k + \rho\epsilon_\beta^2 \tag{5.2a}$$

$$\text{subject to } \tilde{x}_{k+1} = F_k \tilde{x}_k + G_k \tilde{u}_k, \quad k = 0, 1, ..., N-1 \tag{5.2b}$$

$$|\tilde{\beta}_{2,k} + \beta_{2,r,k}| \leq \beta_{2,\max} + \epsilon_\beta, \quad k = 1, ..., N \tag{5.2c}$$

$$|\tilde{\beta}_{3,k} + \beta_{3,r,k}| \leq \beta_{3,\max} + \epsilon_\beta, \quad k = 1, ..., N \tag{5.2d}$$

$$|\tilde{u}_k + u_{r,k}| \leq u_{\max}, \quad k = 0, 1, ..., N-1 \tag{5.2e}$$

$$\tilde{x}_0 = \tilde{x}(s(t)) \text{ given.} \tag{5.2f}$$

where the linear dynamics in (5.2b) are either the straight-path model (2.14) or the linearized model (2.8). If the straight-path model is used, (5.2b) is

$$\tilde{x}_{k+1} = F\tilde{x}_k + G\tilde{u}_k$$

To transform (5.2a) into the QP objective function, see Section 3.3.

5.1.1 Terminal cost

As discussed in Section 3.2.1 the terminal weight matrix P_N is selected to be the solution to the DARE. However, because the DARE can take a long time to solve, the straight-path model with matrices $F = I + \Delta_s A$ and $G = \Delta_s B$ are used in both cases. That is, even when (5.2b) uses F_k and G_k the straight-path matrices are used to solve the DARE. This means that P_N can be computed offline because F and G are constant. Important to remember is that A and B change sign with the direction of the vehicle and therefore two versions of P_N need to be computed.

5.1.2 Penalizing rate-of-chance

To reduce the rate-of-change in the steering of the truck, and to reduce erratic movements of the wheels, additional structure needs to be added. To reduce the speed, the change in steering angle is penalized. For every k the difference $(\tilde{u}_k - \tilde{u}_{k-1})$, hence, needs to be minimized. That is, the vector

$$\begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ \vdots \\ u_{N-1} - u_{N-2} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}}_{\Omega} \mathbf{u} - \underbrace{\begin{bmatrix} u_{-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\delta} \tag{5.3}$$

needs to be minimized, where u_{-1} is the control action from the MPC problem solved in the previous iteration. Using quadratic cost this can be done by adding

the stage cost

$$\sum_{k=0}^{N-1}(\tilde{u}_k - \tilde{u}_{k-1})^T(\tilde{u}_k - \tilde{u}_{k-1}) \tag{5.4}$$

$$= (\Omega\mathbf{u} - \delta)^T(\Omega\mathbf{u} - \delta) \tag{5.5}$$

$$= \mathbf{u}^T\Omega^T\Omega\mathbf{u} - \delta^T\Omega\mathbf{u} + \delta^T\delta, \tag{5.6}$$

to the objective function. Hence, (5.6) added to the QP objective function (3.13) gives

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T(\mathbf{G}^T\mathbf{Q}\mathbf{G} + \mathbf{R} + \Omega^T\mathbf{R}\Omega)\mathbf{u} + 2(\theta^T\mathbf{F}^T\mathbf{Q}\mathbf{G} - \delta^T\mathbf{R}\Omega)\mathbf{u}. \tag{5.7}$$

5.2 Constraints

There are some constraints that have to be taken into consideration when controlling the vehicle. First and foremost it is assumed that there are physical limitations on the steering angle α. This is taken into consideration by constraining the steering angle.

$$|\alpha_k| \leq \alpha_{\max} \rightarrow |u| \leq u_{\max} \rightarrow |\tilde{u}_k + u_{r,k}| \leq u_{\max}, \quad u_{\max} = \frac{\tan\alpha_{\max}}{L_1}.$$

Furthermore the joint angles also need to be constrained and there are two reasons for this. First, the region where the joint angle states can be estimated accurately can in practice be limited [15]. Secondly, if the joint angles get too large it is also highly probable that the truck will jack-knife when reversing. Therefore it is beneficial to constrain the joint angles

$$|\tilde{\beta}_{2,k} + \beta_{2,r,k}| \leq \beta_{2,\max}$$
$$|\tilde{\beta}_{3,k} + \beta_{3,r,k}| \leq \beta_{3,\max}.$$

The constraints on steering angle and joint angles are also considered in the motion planner. Despite that, deviations from the plan can occur during execution of the plan and these constraints are therefore also important to consider in the controller.

The values chosen for $u_{\max}$, $\beta_{3,\max}$ and $\beta_{2,\max}$ are

$$u_{\max} = 3.6$$
$$\beta_{3,\max} = 0.7$$
$$\beta_{2,\max} = 0.7.$$

The value of $u_{\max} = 3.6$ mean the maximum steering angle is 0.6 rad and was chosen to be the same value as in the planner. $\beta_{3,\max}, \beta_{2,\max} = 0.7$ is an inner approximation of the region used in [15]. Whether $\beta_{3,\max}, \beta_{2,\max} = 0.7$ guarantees that the vehicle does not jack-knife has not been tested.

5.2.1 Constraints in QP form

To be able to transform the constraints (5.2c) - (5.2e) into the QP constraints as in (3.16a) the matrices A_z and A_u need to be identified. The absolute-value constraints can be written as

$$\begin{aligned} -\beta_{2,\max} &\leq \tilde{\beta}_{2,k} + \beta_{2,r,k} \leq \beta_{2,\max}, \\ -\beta_{3,\max} &\leq \tilde{\beta}_{3,k} + \beta_{3,r,k} \leq \beta_{3,\max}, \\ -u_{\max} &\leq \tilde{u}_k + u_{r,k} \leq u_{\max}. \end{aligned}$$

First, the right hand side is

$$\begin{aligned} \tilde{\beta}_{2,k} &\leq \beta_{2,\max} - \beta_{2,r,k} \\ \tilde{\beta}_{3,k} &\leq \beta_{3,\max} - \beta_{3,r,k} \\ \tilde{u}_k &\leq u_{\max} - u_{r,k} \end{aligned}$$

which is the same as

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A_z^+} \tilde{x}_k + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{A_u^+} \tilde{u}_k \leq \underbrace{\begin{bmatrix} \beta_{2,\max} - \beta_{2,r,k} \\ \beta_{3,\max} - \beta_{3,r,k} \\ u_{\max} - u_{r,k} \end{bmatrix}}_{b^+}. \tag{5.8}$$

Similarly, the left hand side is

$$\begin{aligned} -\tilde{\beta}_{2,k} &\leq \beta_{2,\max} + \beta_{2,r,k} \\ -\tilde{\beta}_{3,k} &\leq \beta_{3,\max} + \beta_{3,r,k} \\ -\tilde{u}_k &\leq u_{\max} + u_{r,k} \end{aligned}$$

which is the same as

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A_z^-} \tilde{x}_k + \underbrace{\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}}_{A_z^-} \tilde{u}_k \leq \underbrace{\begin{bmatrix} \beta_{2,\max} + \beta_{2,r,k} \\ \beta_{3,\max} + \beta_{3,r,k} \\ u_{\max} + u_{r,k} \end{bmatrix}}_{b^-}. \tag{5.9}$$

Now we just need to stack the matrices found as

$$A_z = \begin{bmatrix} A_z^+ \\ A_z^- \end{bmatrix}, A_u = \begin{bmatrix} A_u^+ \\ A_u^- \end{bmatrix}, b = \begin{bmatrix} b^+ \\ b^- \end{bmatrix}.$$

With A_z, A_u and b the QP constraints can be formed as in Section 3.3.

5.3 Creating mpQP for certification

To be able to certify the complexity of the MPC the mpQP in the form of (4.13a) needs to be identified. Hence, we need to find f_θ and W. However, since u_{-1} is introduced in the objective function the parameter θ is not only the initial state x_0, but

$$\theta = \begin{bmatrix} \tilde{x}_0 \\ \tilde{u}_{-1} \end{bmatrix}. \tag{5.10}$$

Using the objective function (5.7) we have

$$f^T = (\tilde{x}_0^T \mathbf{F}^T \mathbf{Q}\mathbf{G} - \delta^T \mathbf{R}\Omega) = \theta^T (s_1 \mathbf{F}^T \mathbf{Q}\mathbf{G} - s_2 I_0^T \mathbf{R}\Omega) = \theta^T f_\theta^T \tag{5.11}$$

and with the introduced selection matrices

$$I_0 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \; s_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \; s_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{5.12}$$

f_θ is identified as $f_\theta = s_1 \mathbf{F}\mathbf{Q}\mathbf{G} - s_2 I_0^T \mathbf{R}\Omega$. Furthermore, using (3.17d)

$$\mathbf{b} - \mathbf{A}_z \mathbf{F} x_0 = \mathbf{b} - \mathbf{A}_z \mathbf{F} s_1^T \theta = b + W\theta. \tag{5.13}$$

from which it follows that $W = -\mathbf{A}_z \mathbf{F} s_1^T$.

Next we define the space of all possible θ, Θ_0. This space is defined as all values where $\theta_{lb} \leq \theta \leq \theta_{ub}$, where the lower and upper bounds are

$$\theta_{lb} = \begin{bmatrix} -0.2 & -\pi/4 & \beta_{3,min} & \beta_{3,min} & u_{-1,min} \end{bmatrix}^T \tag{5.14}$$

$$\theta_{ub} = \begin{bmatrix} 0.2 & \pi/4 & \beta_{3,max} & \beta_{3,max} & u_{-1,max} \end{bmatrix}^T. \tag{5.15}$$

To include all feasible initial states the values $\beta_{3,\min/\max}$, $\beta_{2,\min/\max}$ and $u_{-1,\min/\max}$ are the values used to constrain the states:

$$\begin{aligned} \beta_{3,\min} &= -\beta_{3,\max} - \beta_{3,r,\max} \\ \beta_{2,\min} &= -\beta_{2,\max} - \beta_{2,r,\max} \\ \beta_{3,\max} &= \beta_{3,\max} - \beta_{3,r,\max} \\ \beta_{2,\max} &= \beta_{2,\max} - \beta_{2,r,\max} \\ u_{-1,\min} &= -u_{\max} - u_{r,\max} \\ u_{-1,\max} &= u_{\max} - u_{r,\max}. \end{aligned}$$

$\beta_{3,r,\max}$, $\beta_{2,r,\max}$ and $u_{r,\max}$ are the maximum reference within the horizon, that is

$$\begin{aligned} \beta_{3,r,\max} &= \max_k \beta_{2,r,k}, \quad k = 0, ..., N \\ \beta_{2,r,\max} &= \max_k \beta_{3,r,k}, \quad k = 0, ..., N \\ u_{-1,\max} &= \max_k u_{r,k}, \quad k = 0, ..., N. \end{aligned}$$

As for the lateral distance $\tilde{z}_3$ and the orientation error $\tilde{\theta}$, which are not constrained in the MPC problem, the values 0.2 and $\pi/4$ where deemed sufficiently large for scenarios occurring in practice.

Important to note is that for this MPC all possible QPs can not be expressed only with the parameter θ. Because b in (5.8) and (5.9) contain the reference, $\mathbf{b}$ in the mpQP (5.13) depends on the reference sequence given by the planner. This means that for every reference sequence $(\beta_{3r,k}, \beta_{2r,k}, u_{r,k})$, $k = 0, ..., N$, we will have a different mpQP.

Also worth noting is that similar to u_{-1}, it is possible use the reference sequence $(\beta_{3r,k}, \beta_{2r,k}, u_{r,k})$, $k = 0, ..., N$ as a parameters by appending them to θ and introduce structure in W to add them to the constraints in a proper way. However, the reference sequence is created in the path planner using the non-linear vehicle model and the reference points are constrained with non-linear constraints, so if this is a good idea and how Θ_0 is to be defined would need investigation. Also, since this would significantly increase the dimension of the parameter space, the certification would be significantly more computationally demanding.

6
Results

To test the MPC controller and the QP solvers two missions, shown in Figure 6.1 and 6.2 where performed. Due to problems with the state estimation which were not investigated in this thesis and due to time constraints the test where performed in simulation. On these mission both the controller performance and the solver performance are measured. The controller performance is measured by how well the vehicle states follow the planned reference. For the QP solvers, the computational time indicates the performance. Lastly the complexity certification of the active-set solver DAQP will be presented.

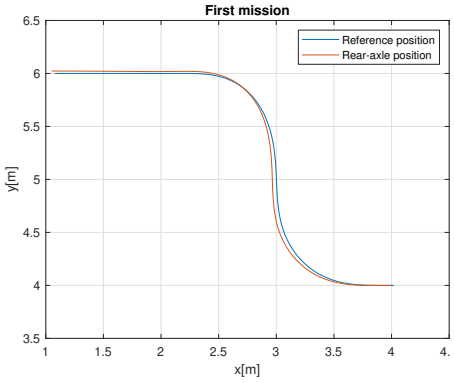

Figure 6.1: *First mission. Starting at right-hand side and reversing to the left.*

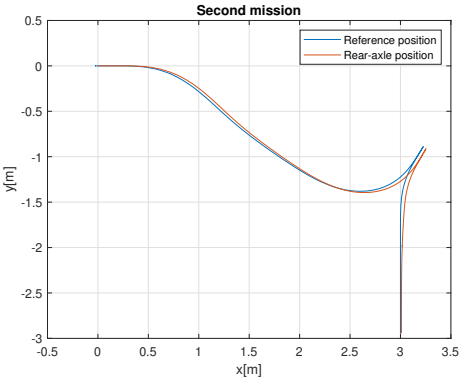

Figure 6.2: *Second mission. Starting forward at the left-hand side and then changing to reverse.*

Unless specified the test were performed with the design parameter values seen

in Table 6.1.

Parameter	Value
Horizon N	20
Sampling distance Δ_S	0.01 m
Controller frequency	20 Hz
Weight matrix Q	**diag**(1,1,1,1)
Weight ρ	100000

Table 6.1: *MPC design parameters.*

6.1 Comparing linearization model and straight-path model

To compare the different linearization models presented in Section 2.4 that are used to create the QP two missions were performed and the distance from the path to the rear axle of the vehicle was measured. The path-following error for the first and the second mission is seen in Figure 6.3 and 6.4.

Figure 6.3: *Path-following error for the first mission comparing the two models.*

Figure 6.4: *Path-following error for the second mission comparing the two models.*

The difference in computation time for solving the MPC problem was also compared, including time to create/calculate the matrices needed to make the QP problem, here called *MPC solution time*. These solution times are shown in Figure 6.5.

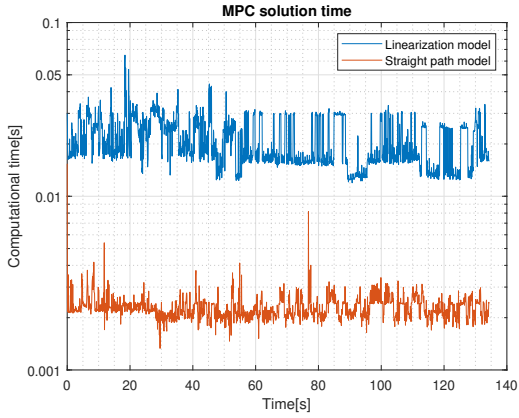

Figure 6.5: *Computation time to create and solve the entire MPC problem for the linearization model compared with the straight-path model. The test was executed on a laptop.*

Figures 6.3 and 6.4 illustrate that with current design and tuning of the MPC there is not much difference between the straight-path model and linearization model in regards to path-following performance. In both cases the terminal cost, P_N, in the MPC is calculated by solving the DARE with the straight-path matrices. The similar performance between the models may therefore indicate that a significant portion on the control performance may be the result of approximating the remaining cost-to-go after the horizon with the optimal LQ cost.

Since the H and A matrices in the QP needs to computed online the computation time when using the linearized model is significantly slower compared to when using the straight-path model, as shown in Figure 6.5. Even though this was tested on a laptop, we see the linearized model MPC breaking the 20 Hz loop rate. This would become even more critical on embedded hardware.

Since there is not a lot of difference in the path-following performance, but that the computation time is much slower for the linearization model, the test that follow are made with only the straight-path model.

6.2 Performance

This section will show further tests for MPC with the selected straight-path model, starting with path-following performance for different horizons and then displaying the QP solution performance for different solvers.

6.2.1 Distance from reference

To further test the path-following performance and the effect of varying the horizon in the MPC, the performance of four MPCs with different horizons were mea-

sured when performing the first mission. The distance from the reference path is seen in Figure 6.6. We see that the path-following performance only differs slightly between the MPCs.

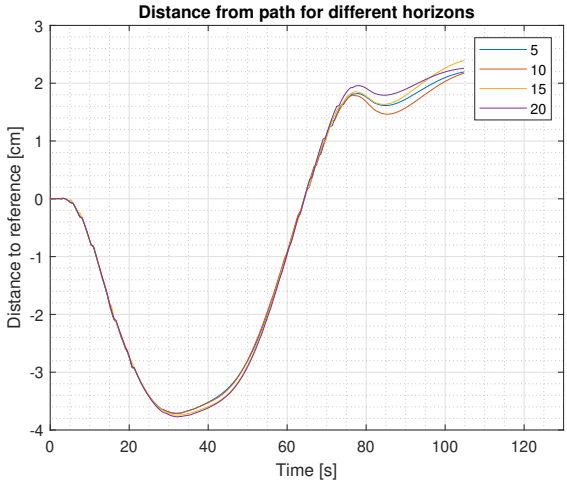

Figure 6.6: *Comparison of performance for different horizons.*

6.2.2 Joint angles

How well the MPC was able to follow the Joing angle references can be seen in Figure 6.7.

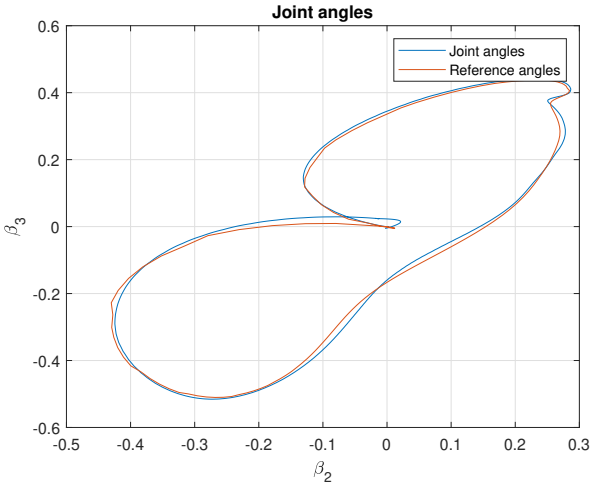

Figure 6.7: *Joint angles for the first mission.*

Worth noting is that the joint angles values are far from the constrained value of $\beta_{max} = 0.7$, which is outside the plotted area in Figure 6.7. Also this is while reversing along the sharpest curves the current planner produces. Hence the constraints are usually not hit during regular missions.

That the constraints are not usually active gives insight into why the performance of the MPC with different horizons are similar. Because the remaining cost-to-go after the horizon is approximated with the optimal infinite horizon LQ cost, the MPCs tested in Figure 6.6 are all essentially infinite horizon LQ controllers. The only difference between them is that the difference $(\tilde{u}_k - \tilde{u}_{k-1})$ for the first N controls are minimized. So for this application with this specific planner an LQ controller would have been sufficient to control the vehicle. However, disturbances might lead to deviations from the plan, and thus constraints are still important. Also, there are other constraints that could be beneficial or interesting to impose. For example:

- The change in steering angle could be constrained $|(\tilde{u}_k - \tilde{u}_{k-1}) - (u_{r,k} - u_{r,k-1})| \leq c_{\max,k},\ k = 0, 1, ..., N-1$ to guarantee that the change in steering angle is not too high. For example to reduce the stress of actuators and prolong their lifespan.

- Constraining the lateral distance from the path, $|\tilde{z}| \leq z_{\max}$, could also be needed if, e.g., the vehicle is to drive in narrow or tight environments where obstacles need to be avoided.

6.2.3 Solver performance

To test and compare the performance of the three different solvers DAQP, qpOASES and OSQP the computational time were measured. The first mission was executed with all solvers. To give a fair comparison between the solvers, all input to the controller was recorded during a execution of the mission using one of the solvers. The MPC was then run with the other solvers while the recorded input data was played back. This ensured that the QP problems which were created and solved were exactly the same for all solvers. The time it takes to solve the QPs, here called *QP solution time*, for the three solvers can be seen in Figure 6.8 and the MPC solution time using the different solvers in Figure 6.9. The mean value of the solution times can be seen in Table 6.2.

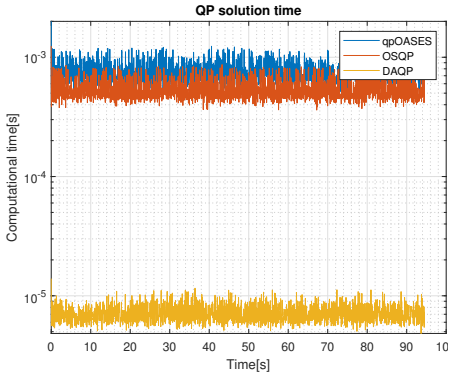

Figure 6.8: Computational time for solving the QP-problem. Test was executed on a RPi.

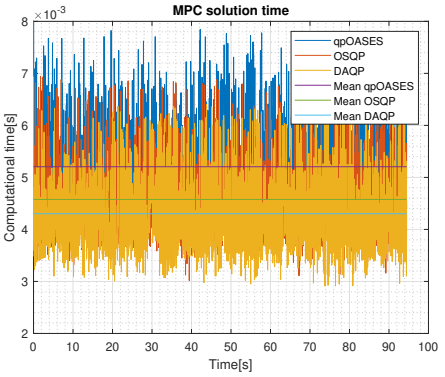

Figure 6.9: Computational to create and solve the entire MPC problem. Test was executed on a RPi.

Table 6.2: *Means of MPC solution times using the different solvers.*

Solver	Mean MPC solution time [ms]	Mean QP solution time [ms]
qpOASES	5.3	0.75
OSQP	4.6	0.54
DAQP	4.3	0.0071

We see that the DAQP manages to solve the QP problem much faster than both OSQP and qpOASES. But when measuring the MPC solution time the difference is much smaller and the MPC solution time when usign DAQP is only slightly faster then when using OSQP. Hence, the time gained by using DAQP is in part lost when converting the primal problem into the dual problem. But there are still time which could be gained to improve the MPC solution time using DAQP. Instead of solving

$$Rx^* = -(\lambda^* + v),$$

for x^* it is possible to calculate

$$x^* = -R^{-1}(\lambda^* + v).$$

Since R is constant (H is constant and $H = R^T R$) and is calculated offline, R^{-1} can also be calculated offline. Also, only the first control is used to steer the vehicle. Hence, not the whole matrix multiplication in $x^* = -R^{-1}(\lambda^* + v)$ needs to be calculated. Only the first row in $-R^{-1}$ times first column in $(\lambda^* + v)$ is needed to get x_0^*.

Another factor which could improve the MPC solution time when using DAQP would be implementing the conversion from the primal problem to the dual problem, that is, solving (4.8) and (4.10), in pure C-code. Currently the creation of

the primal problem and conversion to the dual problem is made with two different C++ libraries. Hence, even though efforts have been made to reuse memory and keeping the triangular properties of R in mind when solving the system of equations there may be some overhead time costs. These would disappear in a C-implementation which would improve the solution time.

6.2.4 Cost difference

To ensure that the solvers give the same solution the objective value was calculated for every QP in the entire mission. The objective value is the value from inserting the solution into the objective function (5.7). The objective value for the solvers was then compared. The difference between qpOASES and DAQP can be seen in Figure 6.10 and between qpOASES and OSQP in Figure 6.11.

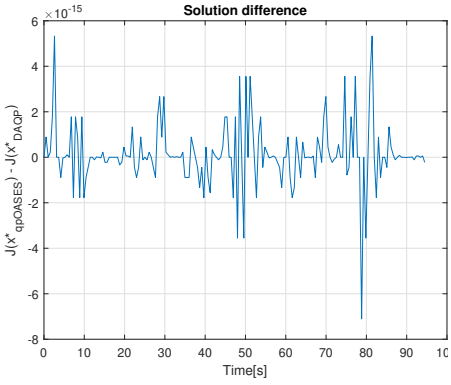

Figure 6.10: *Difference in objective value for an entire mission, comparing the active-set solvers qpOASES and DAQP.*

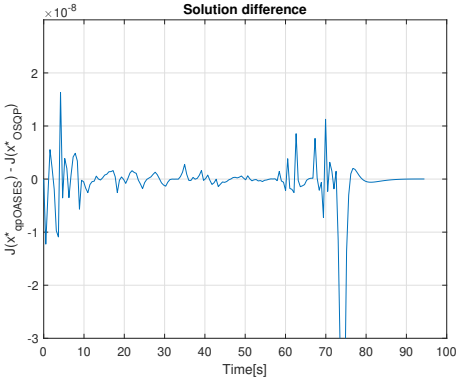

Figure 6.11: *Difference in objective value for an entire mission, comparing the active-set solver qpOASES with the operator splitting solver OSQP.*

The objective value difference between qpOASES and DAQP is in the magnitude of 10^{-15} and between qpOASES and OSQP it is of magnitude 10^{-8}. That the difference is this small makes sense because the condition number for H is low (0.1311), which indicates good numerics.

6.2.5 Warm starts

To show the benefits of warm starting the solver with the previous solution three test where made where the three solvers ran a mission with and without warm starting, seen in Figure 6.12-6.14. Significant improvement can be seen in all cases.

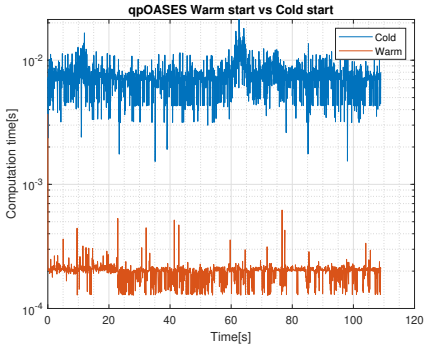

Figure 6.12: Comparison of QP solution time for warm start vs cold start.

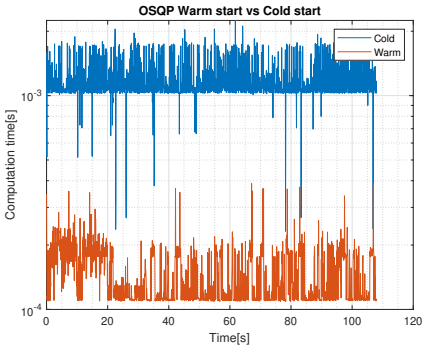

Figure 6.13: Warm start vs cold start for OSQP.

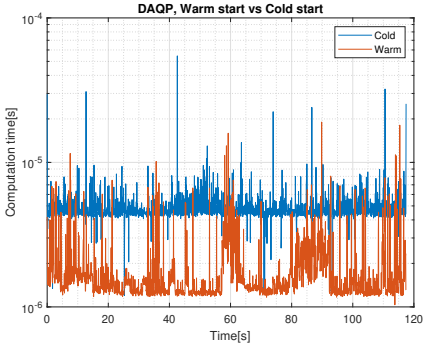

Figure 6.14: Warm start vs cold start for DAQP.

6.3 Performance with tighter constraints

The results in Section 6.2 showed that during a regular execution of a mission the constrains are not active. To show the performance of the MPC and the performance of the solvers in more challenging situations, the constraints on the joint angles were tightened to $\beta_{max} = 0.4$.

6.3.1 Joint angles

To see how the MPC handled the lowered constraint the first mission was executed with varying horizons, see Figure 6.15. Important to keep in mind for this test is that the constraint can be lower than the joint-angle reference. The system is not designed to be able to handle this situation and the controller is not able to follow the reference. Still it is interesting to see if the MPC is able to keep the joint angles within the constraints.

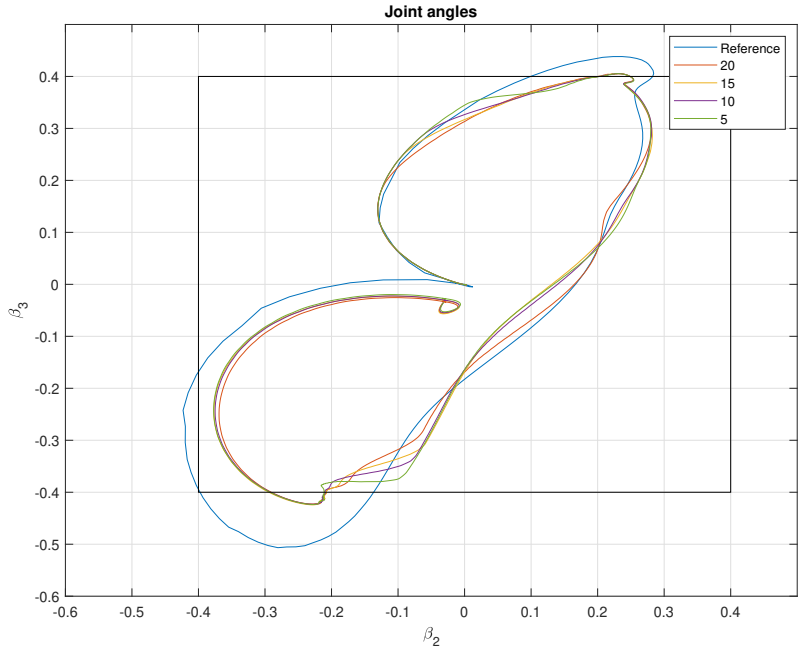

Figure 6.15: *2D plot of β-angles for the first mission run with lowered constraints $\beta_{max} = 0.4$ for different horizons in the MPC.*

Since the constraints are softened lowering β_{max} the MPC does not completely manage to keep the states under the limits, however it manages to significantly lower the joint angles values trying to keep them under the constraints. Interesting to note is that increasing the horizon the MPC notices the upcoming constraints earlier, as expected, but all controllers ends up violating the constraint an equal amount. This is most likely because the model used to predict the future states of the system is not only a linearized model, but a special case of a linearized model, and it can therefore not predict the future states perfectly.

6.3.2 Solution performance

Also interesting is to see how the solution time is affected when the constraints are encountered. The time it takes to solve the QP problem when $\beta_{max} = 0.4$ compared to when $\beta_{max} = 0.7$ is shown in Figure 6.16 - 6.18.

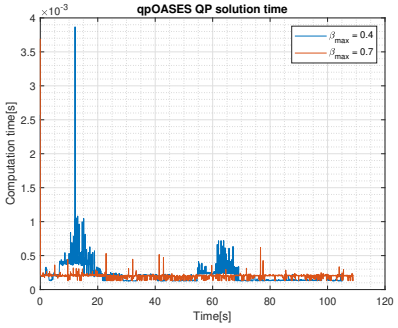

Figure 6.16: Comparison of QP solution times when constrains on β angles are lowered using qpOASES.

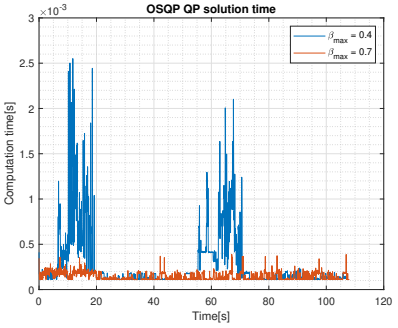

Figure 6.17: Comparison of QP solution times when constrains on β angles are lowered, using OSQP.

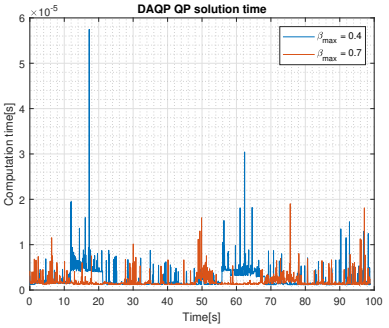

Figure 6.18: Comparison of QP solution times when constrains on β angles are lowered, using DAQP.

As shown activating constraints can significantly increase the QP solution time. It is therefore important to be able to guarantee the worst case complexity in more complex situations. Just testing the solution time for a restricted set of states can be highly misleading.

The significant increase in QP solution time is also seen in the OSQP solution time but for this method of solving QPs **exact** complexity cannot be guaranteed. Hence it is beneficial to use active-set methods combined with the certification framework in [1].

6.4 Certification

The certification framework in [1] was applied to certify the complexity of the MPC when DAQP is used. In Section 6.4.1 it is applied when the controller follows a straight path and in Section 6.4.2 it is applied to two different motion primitives.

6.4.1 Straight path

For the straight path, i.e, when $\beta_{3r}, \beta_{2r}, u_r = 0$, the QP is only dependant on θ. For this mpQP the maximum guaranteed iteration number has been certified for different horizons using [1], see Figure 6.19. The maximum guaranteed iteration number seems to increase, at least up to horizon 20, roughly linearly.

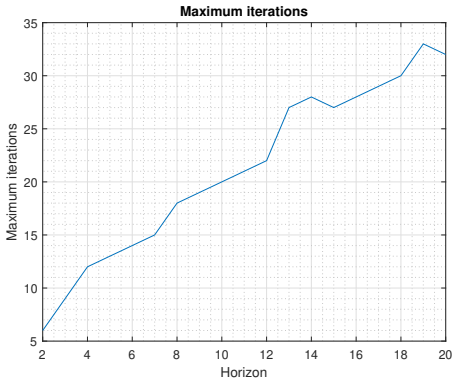

Figure 6.19: *Guaranteed maximum iteration number for the active-set algorithm for different horizons in the MPC, following a straight path in reverse motion.*

Fixing three of the parameters in θ it is possible to get a visualizable 2D slice of the partitioned space Θ. A 2D slice of the partition where the joint angles are varied and the other parameters are set to zero can be seen in Figure 6.20.

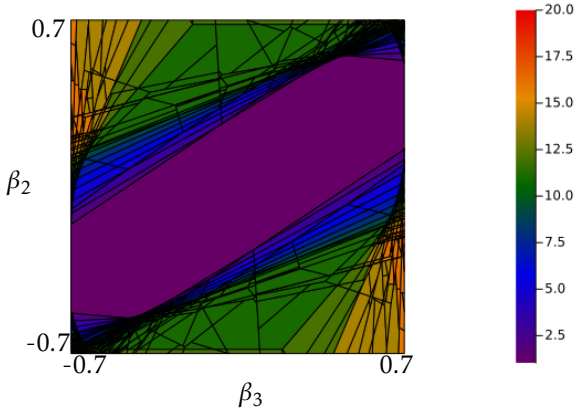

Figure 6.20: *A 2D plot of the partitioned space Θ showing maximum iterations when varying the joint angles. The other parameters are fixed to zero. The color bar shows number of iterations.*

In the 2D slice we can see that for small joint angles and when the angles increase with the same sign we need few iterations to solve the QP. This makes sense because this is where the truck is easier to control. When the angles have different signs the truck has a high risk of jack-knifing and is harder to control.

6.4.2 Primitives

The lattice-planner uses a finite amount of motion primitives to create a mission and in every motion primitive there is a finite amount of reference points. Hence if that the planner were to run without the improvement step, it is possible to certify the maximum iteration for every possible QP that can be created from the references.

The planner used in this thesis uses 800 motion primitives containing 50 reference points each. All these QPs have not been certified but could easily be done given enough time. It is also noted that this could be parallelized since the motion primitives can be considered separately. Below a small selection can be seen, where it is possible to see how the guaranteed maximum iteration number changes over a primitive. For a horizon of 5, every fifth QP is certified and the maximum iteration number for the mpQP is shown at the corresponding location on the primitive.

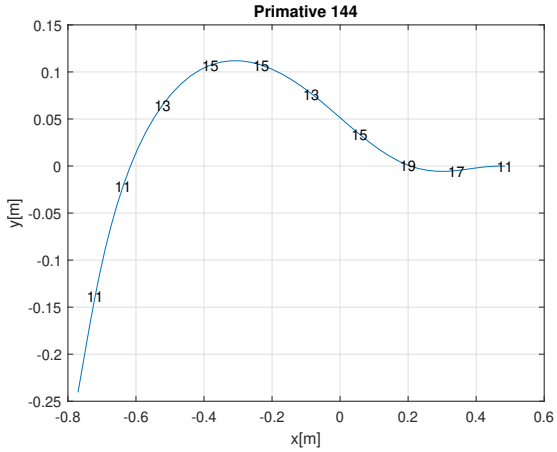

Figure 6.21: *Every fifth QP in the primative is certified. The numbers indicate the guaranteed maximum iteration at the given location on the curve.*

We see in both Figure 6.21 and 6.22 that when following the a motion plan the maximum guaranteed iterations can significantly increase from the straight path case, where the maximum iteration was 13 for a horizon of 5. It can however also be lower.

A 2D slice for one QP from each primitive can be seen in Figure 6.23 and Figure 6.24. For both primitives the truck would start at the right and reverse to the left

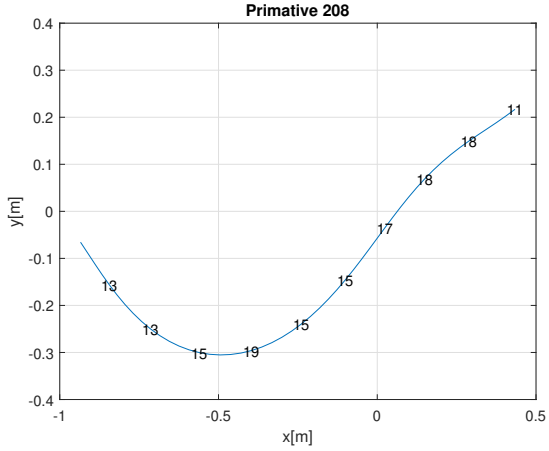

Figure 6.22: *Every fifth QP in the primative is certified. The numbers indicate the guaranteed maximum iteration at the given location on the curve.*

in the image. Hence primitive 144 shows a right turn and primitive 208 a left turn. During a right turn (144) the reference angles are negative and we see that the cold colors move (comparing with Figure 6.20) to the upper right (positive) corner. For a left turn (208) they are positive and we see that the cold colors move to the lower left (negative) corner. This is reasonable since the constraints

$$-\beta_{2,\max} - \beta_{2,r,k} \leq \tilde{\beta}_{2,k} \leq \beta_{2,\max} - \beta_{2,r,k},$$
$$-\beta_{3,\max} - \beta_{3,r,k} \leq \tilde{\beta}_{3,k}+ \leq \beta_{3,\max} - \beta_{3,r,k},$$

are shifted with the reference.

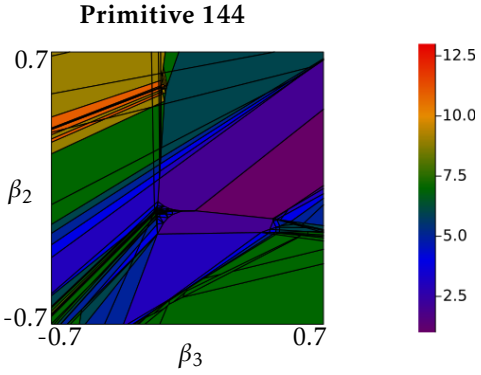

Figure 6.23: *A 2D slice of the partitioned space Θ showing maximum iterations when varying the joint angles. The color bar shows number of iterations.*

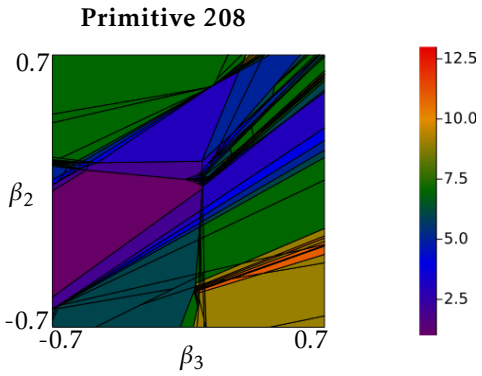

Figure 6.24: *Joint angles 2D plot of the partition for horizon 5, The color bar shows number of iterations.*

7
Conclusions

7.1 Conclusions

In this thesis a path-following MPC controller has been developed to successfully steer a truck and trailer system. A linearized model and a special case of the linearized model, i.e, the straight-path configuration case, were used to predict the system in the MPC. MPCs with both models saw similar path-following performance but the one using a straight-path model was deemed better suited to real-time applications because it had significantly faster computation time.

Three different QP solvers were successfully applied to solve the QP problems that were formed from the MPC problems. Of the three solvers DAQP is the best choice for two reasons. DAQP showed slightly faster solution times than the other two and the solution time has potential to be even faster. Also, DAQP uses the active-set method in which the complexity can be exactly certified using the recently presented certification framework applied in this thesis.

The certification framework [1] was used to certify the iteration complexity for solving every possible QP that can arise from the MPC controller. Initial experiments show how the exact iteration complexity can be determined by certifying every possible mpQP that can be created while traversing every motion primitive that is used in the motion planner. However, due to lack of time and computational power every motion primitive have not been considered. How the guaranteed maximum iterations change along two primitives has been displayed. Also, how the horizon of the MPC effects the guaranteed maximum number of iterations when following a straight path has been investigated and the guaranteed maximum number of iterations seems to increase roughly linearly with the horizon. The certification framework gives the exact working set changes used

in the active-set algorithm, however using these and calculating the guaranteed maximum computational time is still to be done.

7.2 Future work

The natural continuation on this a work would be to calculate the maximum computational time using the working set changes. Obtaining the maximum computational time would also open interesting avenues. Since the solution time for the MPC using the straight path model is fast an interesting avenue would be to guarantee that the MPC could be run on the on board LEGO EV3.

Since all motion primitives can be considered separately certification of every motion primitive could be parallelized. Thus, another natural continuation to this work would be to employ a high-performance computer cluster for the certification.

Another interesting investigation would be to see how the solution times and guaranteed maximum complexity would change introducing other constraints, for example the constraints suggested in Section 6.3.1.

Also investigating how the reference could be used as parameters in the certification could yield interesting result since then all the motion primitives do not need to be certified since we would only have one mpQP.

Bibliography

[1] Daniel Arnström and Daniel Axehill. A unifying complexity certification framework for active-set methods for convex quadratic programming. *IEEE Transactions on Automatic Control*, 2021. doi: 10.1109/TAC.2021.3090749.

[2] Daniel Arnström, Alberto Bemporad, and Daniel Axehill. A dual active-set solver for embedded quadratic programming using recursive LDL' updates. `https://arxiv.org/abs/2002.10291`, 2021.

[3] Daniel Axehill and Anders Hansson. A dual gradient projection quadratic programming algorithm tailored for model predictive control. In *2008 47th IEEE Conference on Decision and Control*, pages 3057–3064, 2008. doi: 10.1109/CDC.2008.4738961.

[4] Alberto Bemporad. A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control. *IEEE Transactions on Automatic Control*, 61(4):1111–1116, 2016. doi: 10.1109/TAC.2015.2459211.

[5] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(01)00174-1.

[6] Kristoffer Bergman. *Exploiting direct optimal control for motion planning in unstructured environments.* Linköping Studies in Science and Technology. Dissertations: 2133. Department of Electrical Engineering, Linköping University. ISBN 9789179296773.

[7] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems.* Cambridge University Press, 2017. ISBN 9781107016880.

[8] Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, 2004. ISBN 0521833787.

[9] Stefano Di Cairano and Ilya V. Kolmanovsky. Real-time optimization and model predictive control for aerospace and automotive applications. In *2018 Annual American Control Conference (ACC)*, pages 2392–2409, 2018. doi: 10.23919/ACC.2018.8431585.

[10] Hans Joachim Ferreau, Stefan Almér, Helfried Peyrl, Juan Luis Jerez, and Alexander Domahidi. Survey of industrial applications of embedded model predictive control. In *2016 European Control Conference (ECC)*, pages 601–601, 2016. doi: 10.1109/ECC.2016.7810351.

[11] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[12] Gianluca Frison and Moritz Diehl. HPIPM: a high-performance quadratic programming framework for model predictive control. `https://arxiv.org/pdf/2003.02547.pdf`, 2020. [online accessed: 23 August 2021].

[13] Pontus Giselsson. Execution time certification for gradient-based optimization in model predictive control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3165–3170, 2012. doi: 10.1109/CDC.2012.6427093.

[14] Oskar Ljungqvist, Daniel Axehill, Johan Löfberg, and Sebastien Gros. *Motion planning and feedback control techniques with applications to long tractor-trailer vehicles*. Linköping studies in science and technology. Dissertations: 2070. Linköping University, Department of Electrical Engineering. ISBN 9789179298586.

[15] Oskar Ljungqvist, Daniel Axehill, Henrik Pettersson, and Johan Lofberg. Estimation-aware model predictive path-following control for a general 2-trailer with a car-like tractor. `https://arxiv.org/abs/2002.10291`, 2020. [online accessed: 17 August 2021].

[16] David Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2014.10.128.

[17] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, 2006. ISBN 9780387303031.

[18] Panagiotis Patrinos and Alberto Bemporad. An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Transactions on Automatic Control*, 59(1):18–33, 2014. doi: 10.1109/TAC.2013.2275667.

[19] Stefan Richter, Colin Neil Jones, and Manfred Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2012. doi: 10.1109/TAC.2011.2176389.

[20] Conrad Sanderson and Ryan Curtin. An adaptive solver for systems of linear equations. In *2020 14th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2020. doi: 10.1109/ICSPCS50536.2020.9309998.

[21] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL `https://www.ros.org`.

[22] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020. doi: 10.1007/s12532-020-00179-2. URL `https://doi.org/10.1007/s12532-020-00179-2`.

[23] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010. doi: 10.1109/TCST.2009.2017934.

[24] A. Zheng and M. Morari. Stability of model predictive control with mixed constraints. *IEEE Transactions on Automatic Control*, 40(10):1818–1823, 1995. doi: 10.1109/9.467664.