Abstract

The collaborative magnetic levitation project, *Maggy*, has been in active development for a take-home lab platform designed to deploy control theory to a practical real-world physical setup. Earlier versions of the system relied on PD controllers and basic filtering techniques to stabilize the magnet. This thesis focused on enhancing the control architecture by implementing a linear Model Predictive Controller (MPC) and robust MPC (RMPC), both combined with a Moving Horizon Estimator (MHE) for real-time state feedback. A linearized model of the system was used to formulate the MPC problem as a constrained Quadratic Program (QP), enabling efficient real-time optimization in MATLAB. The performance of MPC and RMPC was evaluated through four simulation-based experiments addressing disturbance rejection, model mismatch, constraint handling, and reference tracking. Results showed that while standard MPC performs well under nominal conditions, it suffers from feasibility issues and degraded performance when subject to strong disturbance or uncertainties. In contrast, the RMPC consistently demonstrated improved robustness and reliability, particularly in external perturbations or reference tracking. The use of slack variables allowed for enhanced feasibility under strict bounds. Although the implementation was limited to a linear framework due to time constraints, the study demonstrated the practical benefits of robust control and laid the groundwork for future real-time deployment and nonlinear MPC extensions. The thesis highlights the importance of robustness in control design for safety-critical systems like magnetic levitation systems

Preface

This thesis has been written as part of my Master's degree in Industrial Cybernetics at the Department of Engineering Cybernetics, NTNU. The work presented in this thesis is part of a collaborative project involving several students, aimed at developing a magnetic levitation system to be used in take-home lab assignments for control theory courses. It has been a great privilege to contribute to this research team.

I would like to express my sincere gratitude to Professor Damiano Varagnolo and PhD candidate Hans Alvar Engmark for their invaluable guidance and support throughout the project. Their technical insights and thoughtful feedback have been instrumental in shaping this thesis.

I also wish to give a huge thanks to my collaborator and fellow student, Ørnulf Damsgaard, for the joint development of the control system and for the excellent partnership throughout the project. Working together made the experience both enjoyable and deeply rewarding.

Finally, I would like to express my gratitude to my family and friends for their unwavering support, guidance and patience during my studies.

Mikael Mjelde
Trondheim, June 2025

Contents

List of Figures v

List of Tables vii

Nomenclature viii

1 Introduction 1

1.1 Background . . . 1

1.2 System Description . . . 1

1.3 Previous Work . . . 2

1.4 Thesis Contribution . . . 3

1.5 United Nations Sustainable Development Goals . . . 3

1.5.1 SDG 4 - Quality Education . . . 3

1.5.2 SDG 9 - Industry, Innovation and Infrastructure . . . 3

1.5.3 SDG 11 - Sustainable Cities and Communities . . . 4

1.5.4 SDG 17 - Partnerships for the goals . . . 4

1.6 Declaration of generative AI and AI-assisted technologies during this project . . . 4

1.7 Outline of the Thesis . . . 4

2 Scope 5

2.1 Problem Statement . . . 5

2.2 Goals . . . 5

2.3 Limitations and Assumptions . . . 5

3 Literature Review 6

3.1 Model Predictive Control: Theory and Practice - a Survey . . . 6

3.2 MPHC - Applications to Industrial Processes . . . 6

3.3 DMC - Dynamic Matrix Control - A Computer Control Algorithm . . . 6

3.4 Real-Time Optimization and Nonlinear Model Predictive Control of Processes Governed by Differential-Algebraic Equations . . . 7

3.5 Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation . 7

3.6 Successive Linearization Based Model Predictive Control of Variable Stiffness Actuated Robots . . . 8

3.7 Real-time MHE-based nonliner MPC of a Pendubot system . . . 8

3.8 Implementation of Linear Model Predictive Control - Tutorial . . . 9

3.9 A Simple and Efficient Tube-based Robust Output Feedback Model Predictive Control Scheme . . . 9

4 Theory **11**

4.1 Optimization . . . 11

4.1.1 The Optimization Problem . . . 11

4.1.2 Unconstrained and Constrained Optimization - Optimality Conditions . . . 13

4.1.3 Duality . . . 15

4.1.4 Solving Optimization Problems . . . 15

4.1.5 SQP . . . 16

4.2 Dynamic Optimization . . . 17

4.2.1 Control . . . 18

4.2.2 Linear Quadratic Control . . . 19

4.3 Model Predictive Control . . . 20

4.3.1 Linear MPC . . . 21

4.3.2 Reference Tracking . . . 23

4.3.3 Robust MPC . . . 24

4.3.4 Nonlinear MPC . . . 25

4.4 State Estimation . . . 26

4.4.1 Kalman Filter . . . 27

4.4.2 Moving Horizon Estimation . . . 28

4.5 Numerical Simulation . . . 29

4.6 Practical Tools . . . 30

4.6.1 Finite Difference Linearization . . . 31

4.6.2 Root Mean Squared Error . . . 31

5 Methodology **32**

5.1 System Model . . . 32

5.2 MPC problem formulation . . . 34

5.2.1 Reference Tracking Formulation . . . 35

5.2.2 Robust MPC Formulation . . . 36

5.2.3 Slack Variable Formulation . . . 36

5.3 Closed-loop Setup . . . 38

5.4 Code . . . 39

5.4.1 System Model Initialization . . . 40

5.4.2 MPC Tuning . . . 41

5.4.3 MPC Class . . . 42

5.4.4 MPC Bounds . . . 45

5.4.5 Main Control Loop . . . 46
5.4.6 RMSE test . . . 48
5.5 Experiments . . . 50
5.5.1 Experiment 1 - Disturbance Rejection . . . 50
5.5.2 Experiment 2 - Model Mismatch . . . 51
5.5.3 Experiment 3 - Constraint Handling and Feasibility . . . 51
5.5.4 Experiment 4 - Reference Tracking . . . 51

6 Results **52**
6.1 Experiment 1 - Disturbance Rejection . . . 52
6.1.1 Low Disturbance . . . 52
6.1.2 High Disturbance . . . 56
6.2 Experiment 2 - Model Mismatch . . . 60
6.3 Experiment 3 - Constraint Handling and Feasibility . . . 64
6.3.1 Without Slack variables . . . 64
6.3.2 With Slack variables . . . 71
6.4 Experiment 4 - Reference Tracking . . . 78

7 Discussion **85**
7.1 Methodology discussion . . . 85
7.2 Result discussion . . . 85
7.2.1 Experiment 1 - Disturbance Rejection . . . 85
7.2.2 Experiment 2 - Model Mismatch . . . 86
7.2.3 Experiment 3 - Constraint Handling . . . 87
7.2.4 Experiment 4 - Reference Tracking . . . 89
7.2.5 Improvements for the experiments . . . 90
7.3 Future work . . . 90

8 Conclusion **92**

Bibliography **93**

List of Figures

1 Illustration of maglev system . . . 1
2 The current version 4.1 of the maglev . . . 2
3 UN SDGs: 4, 9, 11, and 17. . . . 3

4 Geographical representation of the minimization problem example in Equation (3) adapted from [18, p. 5] . . . 12

5 Block diagram of the closed-loop control system. *System* is either the simulated model or the real physical maglev system. . . . 38

6 RMSE result with several horizon N and timestep dt, with open-loop vs. closed-loop. 50

7 Simulation of state positions for RMPC and MPC with `disturbance_1` (See Table 4) added. . . . 53

8 Simulation of state velocity for RMPC and MPC with `disturbance_1` (See Table 4) added. . . . 54

9 Simulation of control sequence for RMPC with `disturbance_1` (See Table 4) added. 55

10 Simulation of control sequence for MPC with `disturbance_1` (See Table 4) added. 55

11 Simulation of state positions for RMPC and MPC with `disturbance_2` (See Table 4) added. . . . 57

12 Simulation of state velocity for RMPC and MPC with `disturbance_2` (See Table 4) added. . . . 58

13 Simulation of control sequence for RMPC with `disturbance_2` (See Table 4) added. 59

14 Simulation of control sequence for MPC with `disturbance_2` (See Table 4) added. 59

15 Simulation of RMPC and MPC with MHE estimates and model mismatch. 61

16 Simulation of RMPC and MPC with MHE estimates and model mismatch. 62

17 Control sequence of RMPC with MHE estimates and model mismatch. 63

18 Control sequence of MPC with MHE estimates and model mismatch. 63

19 MPC simulation of position with and without MHE estimates with strict control bounds, see Table 6. . . . 65

20 MPC simulation of velocity with and without MHE estimates with strict control bounds, see Table 6. . . . 66

21 Control sequence for MPC with MHE estimates with strict control bounds, see Table 6. . . . 67

22 Control sequence for MPC optimal with strict control bounds, see Table 6. 67

23 RMPC simulation of position with and without MHE estimates with strict control bounds, see Table 6. . . . 68

24 RMPC simulation of position with and without MHE estimates with strict control bounds, see Table 6. . . . 69

25 Control sequence for RMPC with MHE estimates with strict control bounds, see Table 6. . . . 70

26 Control sequence for RMPC optimal with strict control bounds, see Table 6. . . . 70

27 MPC simulation of position with and without MHE estimates with strict control bounds and slack variables active, see Table 6. . . . 72

28 MPC simulation of velocity with and without MHE estimates with strict control bounds and slack variables active, see Table 6. . . . 73

29 Control sequence for MPC with MHE estimates with strict control bounds and slack variables active, see Table 6. . . . 74

30 Control sequence for MPC optimal with strict control bounds and slack variables active, see Table 6. 74

31 RMPC simulation of position with and without MHE estimates with strict control bounds and slack variables active, see Table 6. 75

32 RMPC simulation of position with and without MHE estimates with strict control bounds and slack variables active, see Table 6. 76

33 Control sequence for RMPC with MHE estimates with strict control bounds and slack variables active, see Table 6. 77

34 Control sequence for RMPC optimal with strict control bounds and slack variables active, see Table 6. 77

35 Simulation of position of MPC with and without MHE estimates, with reference trajectory. 79

36 Simulation of velocity of MPC with and without MHE estimates, with reference trajectory. 80

37 Control sequence for MPC with MHE estimates, with reference trajectory. 81

38 Control sequence for MPC optimal, with reference trajectory 81

39 Simulation of position of RMPC with and without MHE estimates, with reference trajectory. 82

40 Simulation of velocity of RMPC with and without MHE estimates, with reference trajectory. 83

41 Control sequence for RMPC with MHE estimates, with reference trajectory. 84

42 Control sequence for RMPC optimal, with reference trajectory 84

List of Tables

1 Key Physical Parameters in `params` 33

2 `MPCclass.m` Input Arguments 42

3 Bounds selected for states and inputs 46

4 Tuning Parameters for Experiment 1 52

5 Model Changes for Simulation 60

6 Tuning Parameters for Experiment 3 64

7 Tuning Parameters for Experiment 4 78

Nomenclature

Δt	Discrete time step
α	Smoothing factor in EMA filter
ϵ	Slack variable
ρ	Scalar weight for linear term for slack variables
$\mathbf{A}_c$	Continuous-time state transition matrix
$\mathbf{A}_d$	Discrete-time state transition matrix
$\mathbf{B}_c$	Continuous-time control input matrix
$\mathbf{B}_d$	Discrete-time control input matrix
$\mathbf{C}$	Measurement matrix
$\mathbf{f}$	Linear cost term vector
$\mathbf{G}$	Quadratic cost term matrix
$\mathbf{K}$	Gain matrix
$\mathbf{N}$	Prediction Horizon
$\mathbf{Q}$	State weighting matrix
$\mathbf{R}$	Control input weighting matrix
$\mathbf{S}$	Weight matrix for quadratic term for slack variables
$\mathbf{u}$	Control input vector
$\mathbf{w}$	Process noise vector
$\mathbf{x}$	State vector
$\mathbf{y}$	Measurement vector
DAE	Differential Algebraic Equation
EMA	Exponential Moving Average
KF	Kalman Filter
lb	Lower bound vector
LQR	Linear Quadratic Regulator
MHE	Moving Horizon Estimation
MPC	Model Predictive Control
NEES	Normalized Estimation Error Squared
NIS	Normalized Innovations Squared
ODE	Ordinary Differential Equation
RMPC	Robust Model Predictive Control
RMSE	Root Mean Squared Error
ub	Upper bound vector

1 Introduction

This section's purpose is to introduce the reader to the maglev project, and what aspect of the project this thesis is. The background of the project will be introduced, following with a description of the physical maglev system. The previous work done for the whole maglev project will be presented, as well as a detailed outline of the structure of this thesis.

1.1 Background

Magnetic levitation, or maglev, is an extensively researched concept. The maglev concept is particularly popular in applications such as high-speed trains, from which the term *maglev* originates from [1]. The underlying technology relies on magnetic interactions, specifically, the repulsive force generated when two like poles of magnets are brought close together. By harnessing this force, it is possible to achieve levitation without physical contact, which can dramatically reduce friction in dynamic systems. The project, which this thesis is a part of, is based on an existing educational maglev platform called *Maggy*. The Maggy is currently under development, while this thesis is being written. Throughout this thesis the Maggy will be referred to as "the maglev system".

Professor Daminao Varagnolo, who serves as the thesis supervisor, and PhD cadidate Hans Alvar Engmark, the co-supervisor, started the maglev system project. Their primary goal was to foster collaboration among students while providing a hands-on opportunity to apply theoretical knowledge of control systems to a real-world physical setup. Over the years, several students have contributed to the development of the project with project reports, bachelor's theses, and master's theses. This thesis is part of such a collaboration and was conducted jointly with a fellow student, Ørnulf Damsgaard. Both theses contribute to the development of the overall control system for the maglev system, with this thesis focusing on the control aspect. Damsgaard's thesis cover the state estimation aspect.

1.2 System Description

The physical maglev system consists of a modular circuit board designed for magnetic levitation. It features four permanent magnets and four solenoids arranged symmetrically to levitate a separate permanent magnet positioned above the board. An illustration of the maglev system is shown in Figure 1.

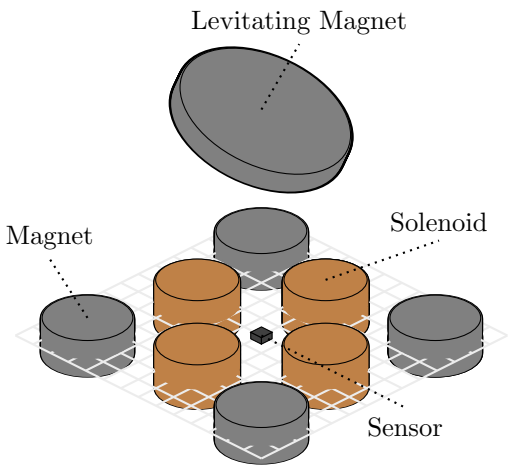


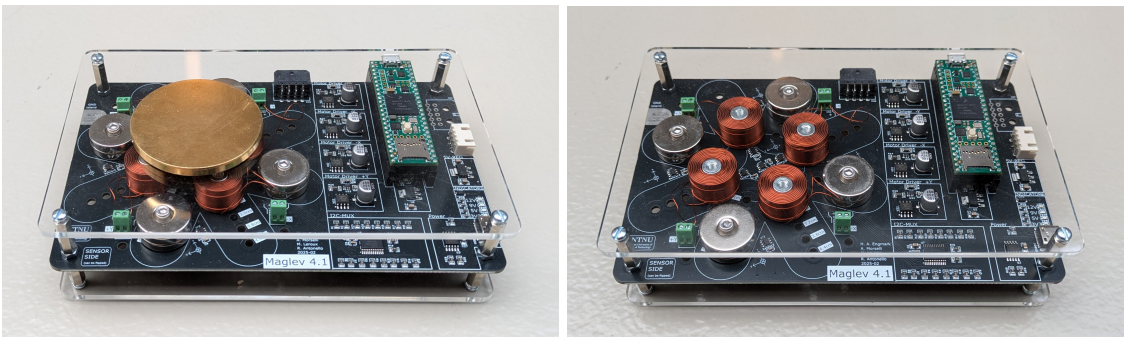
Figure 1: Illustration of maglev system

In this configuration, the four permanent magnets generate the vertical lifting force needed to counteract gravity, while the solenoids are responsible for horizontal stabilization, allowing precise

control of the levitating magnet's position within the plane.

The maglev platform is constantly under development while writing this thesis, with several hardware revisions being developed. The system is designed for use as a take-home laboratory assignment, a novel educational concept that enables students to work with real hardware outside of the classroom. The latest version of the maglev system, which is also used in this thesis, is shown in Figure 2. There has been significant design improvements throughout the different versions of the maglev system. The rectangular form on the 4.1 version was just changed from a circular board shape. Furthermore, the power connector was upgraded to USB-C, for the current version, to improve usability. With the rectangular board shape the possibility of connecting multiple maglev boards side-by-side is improved, which is a part of the maglev project's future goals of enabling modular interaction between units.

To maintain stable levitation of the magnet a feedback control system is essential. The current onboard control consists of a basic Proportional-Derivative (PD) controller. While this setup provides a simple means of stabilization, it is not essential in more advanced control problems which might occur. This thesis aims to enhance the control system by implementing a Model Predictive Controller (MPC), which is an advanced control strategy that uses a system model to predict future states and compute optimal control inputs over a finite horizon. The MPC also explicitly handles constraints on states and inputs. By incorporating MPC, the system can achieve more robust and precise control, particularly under disturbances or when operating near physical limits. This opens the door to improved performance and greater flexibility compared to traditional control methods.



(a) Maglev version 4.1 with magnet

(b) Maglev version 4.1 without magnet

Figure 2: The current version 4.1 of the maglev

1.3 Previous Work

As stated earlier, several student have been contributing to the maglev project throughout the years. Pål Ivar Delphin Kværnø Fosmo, Sverre Graffer and Sacit Ali Senkaya collaborated on an educational book that introduced the Maggy platform [2]. The book covers a range of foundational topics relevant to the system, such as software architecture, stabilization techniques, control strategies, etc. This book, along with all previous work performed by other students, builds upon a shared GitHub repository initiated by co-supervisor Hans Alvar Engmark [accessed January 2025]. The repository serves as a central hub for the maglev project, containing code, simulation tools, and all relevant documentation.

In addition, Hans Alvar Engmark and Kiet Tuan Hoang have published a research paper that presents the physical modeling and control principles of the maglev system [3]. This work established a solid foundation for the development of the simulation tools and has guided much of the control system designed that followed.

1.4 Thesis Contribution

This thesis contributes to the maglev project by implementing a advanced controller, Model Predictive Controller, in MATLAB. This will enable advanced, constraint-aware control of the levitating magnet. While previous versions of the system relied on traditional control strategies, this marks the implementation of an advanced controller, enhancing control capabilities and precision.

This thesis builds upon the author's project thesis [4], which focused on developing a foundational understanding of MPC through simulation-based experiments. That earlier work explored the formulation and implementation of MPC for simplified 2D systems, laying the groundword for the more complex implementation presented in this thesis. The MPC is applied to the full 3D model of the maglev system, with the ultimate goal of achieving full closed-loop control with collaboration with Damsgaard's project.

1.5 United Nations Sustainable Development Goals

This section covers this thesis' relevance to the UN's Sustainable Development Goals, which is adapted from the author's project thesis [4].

The United Nations (UN) has established 17 Sustainable Development Goals (SDGs) aimed at reducing poverty, improving health and education, decreasing inequality, and fostering sustainable economic growth [accessed April 2025]. This thesis contributes to the following SDGs: 4, 9, 11, 17.

Disclaimer: The content of this publication has not been approved by the United Nations and does not reflect the views of the United Nations or its officials or Member States.

This use of the SDG icons follows the UN's official guidelines for non-UN entities and is for informational purposes only. It does not imply endorsement by the United Nations. For full guidelines, see UN SDG Guidelines [accessed April 2025].

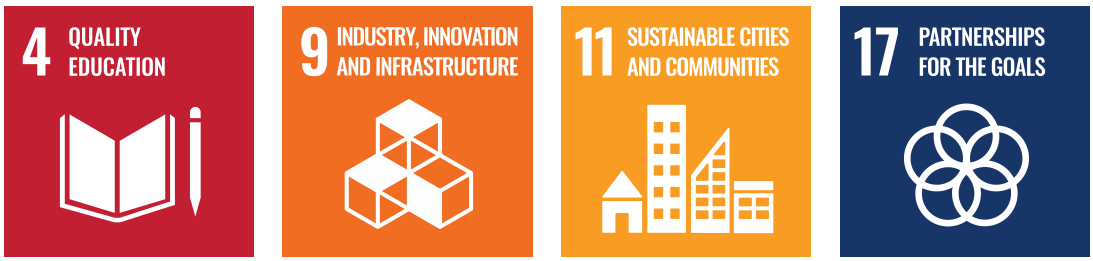


Figure 3: UN SDGs: 4, 9, 11, and 17.

1.5.1 SDG 4 - Quality Education

The primary goal of the development of the maglev system is to be used in take-home laboratory assignments for educational purposes, which forms the foundation of this thesis. By providing a portable and modular maglev board, the system enables student studying control engineering to apply theoretical concepts to a real-world application. This hands-on experience enhances learning outcomes and contributes to improving the overall quality of education.

1.5.2 SDG 9 - Industry, Innovation and Infrastructure

Maglev technology has been successfully applied in various real-world contexts, most notably in high-speed trains, explained in [5]. Continued research related to maglev systems in general, like in this thesis, may contribute to future advancements in maglev-based transportation and other innovative infrastructure solutions.

1.5.3 SDG 11 - Sustainable Cities and Communities

This thesis' relevance to the SDG 11 lines with the reasoning for the SDG 9. Development of maglev-based infrastructure will improve sustainability. Maglev systems operate using electricity and permanent magnetic materials, making them cleaner alternatives to traditional mechanical systems. Furthermore, the absence of physical contact between moving parts eliminates friction, aside from air resistance, resulting in significantly higher energy efficiency.

1.5.4 SDG 17 - Partnerships for the goals

As stated earlier, this thesis is written as a part of a larger project where several students contribute to different parts of the control system for the maglev. This introduce a partnership between several students to achieve the main goal.

1.6 Declaration of generative AI and AI-assisted technologies during this project

During the preparation of this work the thesis author used OpenAI's large language model ChatGPT for the purposes listed below. After using this service, the author reviewed and edited the content received as needed and takes full responsibility for the content of the published thesis.

- Clarify and rephrase theoretical material for presentation during the thesis, as well as finding reliable articles or papers as external sources of information
- Suggest structure improvements of the thesis, rephrase text and grammatical corrections
- Help to develop and debug MATLAB code for the MPC and full control system implementation used during this thesis
- Interpret results from experiments done during this thesis

1.7 Outline of the Thesis

The structure of this thesis is listed below, following a standard structure of a technical report.

- **Scope**: Problem statement, goals, assumptions
- **Literature review**: A review of previous work of MPC and MHE
- **Theory**: All theoretical background related to the thesis
- **Methodology**: Practical implementation of the control system
- **Results**: Experiment results
- **Discussion**: Discussion of implementations, results, goals and future work
- **Conclusion**

2 Scope

Due to time limitation throughout the project, this thesis requires some limitations. During this section, the problem statement of this thesis, goals, limitations and assumptions made will be presented.

2.1 Problem Statement

This thesis investigates the effectiveness of implementing Model Predictive Control (MPC) for the maglev system, with a particular focus on evaluating the role of robustness in practical control applications. It aims to assess the performance of a standard MPC approach in comparison to a robust tube-based MPC, which explicitly accounts for model uncertainties and external disturbances. The study seeks to determine how much robustness contributes to maintaining stable and accurate control under real-world conditions, and whether the added complexity of robust MPC is justified for the maglev system. Ultimately, the goal is to explore how these advance control strategies can improve the existing on-board control framework with a advanced closed-loop strategy with a Moving-Horizon Estimator MHE.

2.2 Goals

The main goals for this project and thesis is listed below.

- To gain a deeper understanding of advanced control techniques, particularly MPC, including its theoretical foundations, control problems, and implementations.
- To evaluate and compare the performance of a standard MPC implementation with a tube-based Robust MPC in a real time closed-loop control with MHE estimates.
- To obtain greater experience in technical writing and academic research methodology.

2.3 Limitations and Assumptions

This thesis is written as part of the Maglev project in collaboration with student Ørnulf Damsgaard, who writes a separate thesis on the same project presented in this thesis. While this thesis focuses on the control aspect of the project, Damsgaards thesis focuses on the state estimation aspect.

Due to time limitations the scope required some limitations in this thesis. Therefore, during this thesis the standard implementation of MPC is explored and compared to a tube-based Robust MPC. The implementation of the MPC is following a standard linear quadratic programming approach to simplify the implementations, due to time limitations. With more time, another approach would be researched, like implementation of a nonlinear MPC to increase accuracy and generality. The maglev system is assumed to be linear, time-invariant and near the equilibrium point where the magnet levitates. This assumption allows the controllers to be implemented efficiently using the quadratic programming formulations with fixed model dynamics.

MATLAB was chosen as the development environment for this thesis because of its accessible interface and strong support for control system design and optimization. There are alternatives, like Python, which may have improved performance, but MATLAB's built in solvers and simulation tools made it a practical choice for this thesis.

3 Literature Review

This section will cover some literature studies done during preparation for this thesis. When deciding which approaches to follow, and how to implement the controllers, we took inspiration from several articles and papers which are discussed during this section.

3.1 Model Predictive Control: Theory and Practice - a Survey

The foundations of MPC emerged in the late 1970s, marking a significant shift in the approach to process control. One of the earliest implementations was described by Richalet et al. in 1978 [6] under the name Model Predictive Heuristic Control (MPHC), which demonstrated promising industrial results. The paper from Richalet about MPHC is reviewed in Section 3.2. The paper reviewed in this section is a paper about the history, theory and development of MPC written of Carlos E. Garcia, David M. Prett, and Manfred Morari [7]. Shortly after the introduction of MPHC from Richalet, in 1979, the concept of Dynamic Matrix Control (DMC) was introduced independently by Cutler and Ramakar, and by Prett and Gillette at Shell, which is reviewed in Section 3.3. These early developments shared a common principle which was the use of an explicit dynamic model of the plant to predict future system behavior and optimize control inputs accordingly.

The key idea in MPC, according to this paper [7], is to solve an optimization problem at each time step, using updated measurements to compute a sequence of future control inputs. Only the first input is applied before the process repeats. This receding horizon strategy enables systematic handling of multivariable interactions, constraints, and changing conditions. MPC gained widespread use in the process industries due to its ability to operate close to constraint boundaries while maintaining safety, efficiency, and product quality. Various formulations such as DMC, Model Algorithmic Control (MAC), and Internal Model Control (IMC) have beed developed, all following the same core principle of prediction-based optimization [7]. Today, MPC remains one of the most widely used advanced control strategies in industry, valued for its flexibility, constraint handling, and strong performance in complex multivariable systems.

3.2 MPHC - Applications to Industrial Processes

Model Predictive Heuristic Control (MPHC), introduced by Richalet, Rault, Testud, and Papon in 1978, was one of the fist successful implementations of the predictive control concept in industry [6]. MPHC was developed to address the complexity of industrial multivariable (MIMO) systems where disturbances often impact unmeasured variables, making traditional strategies insufficient. The MPHC framework is built on three key principles. First, the internal model of the plant is represented by its impulse response, allowing the algorithm to predict future output behavior based on past inputs. Second, a reference trajectory defines the desired closed-loop system behavior over time, guiding the system from its current state toward the target setpoint. Finally, control inputs are computed iteratively using a heuristic optimization optimization procedure rather than a single-step solution. This allows MPHC to flexibly adapt its control actions at each time step using rule-based reasoning, making it practical and robust for industrial applications [6].

3.3 DMC - Dynamic Matrix Control - A Computer Control Algorithm

Dynamic Matrix Control (DMC) was developed by engineers at Shell to address complex multivariable control problems in the oil and chemical industries, where traditional single-loop controllers were inadequate [8]. The DMC algorithm directly incorporates a dynamic model of the process, using a step response representation to predict future outputs and optimal control moves over a finite horizon.

Unlike conventional PID-based schemes, DMC formulates the control problem as an optimization task, often using a least square approach to minimize the deviation between predicted outputs

and reference values. Importantly, DMC explicitly accounts for process dead time and system constraints, enabling it to handle slow dynamics, time delays, oscillations, and nonlinear behaviors more efficiently. [9]. The method assumes the system can be approximated by linear dynamics, making it broadly applicable to systems modeled by linear differential equations. According to Cutler and Ramaker, the success of DMC relies on preserving two key properties: linear superposition and scale invariance. These assumptions allow the model-based optimization to deliver robust control performance across wide range of operating conditions.

3.4 Real-Time Optimization and Nonlinear Model Predictive Control of Processes Governed by Differential-Algebraic Equations

In 2002, Diehl, Bock, Schöder, Findeisen, Nagy and Allgöwer released an article about real-time opimization and nonlinear MPC of systems descibed by differential-algebraic equations (DAE), with a focus on applications in chemical process control [10]. The method is based on direct multiple shooting, a simultaneous strategy that discretizes both the control inputs and system dynamics, transforming the infinite-dimensional optimal control problem into a structured nonlinear programming (NLP). A key advantage of direct multiple shooting is its ability to decouple the integration of the system dynamics over subintervals, making it well-suited for parallel computation. The method naturally handles constraints, offers accurate sensitivity information through efficient DAE solvers, and provides good initial guesses for iterative solvers by incorporating prior trajectory information.

To enable real-time applicability, the authors developed a real-time embedding strategy where precomputable components, such as model linearizations, gradients, and Hessian approximations, are prepared in advance. When disturbance occurs, only a single Sequential Quadratic Programming (SQP) step is performed online, allowing for immediate control action. The real-time iteration approach yields fast feedback responses while maintaining near-optimal performances, even under disturbances and model-plant mismatch. The approach is validated through nonlinear MPC of a high-purity distillation column model with 164 states, where the control goal is to maintain product quality despite feed disturbances. The simulations demonstrate excellent closed-loop performance with CPU times well below the ten second sampling interval, even for long prediction horizons and large numbers of control intervals.

Overall, the work showcases that advance nonlinear MPC schemes based on direct multiple shooting can be executed in real-time for complex industrial systems, achieving high control performance and robustness to disturbances [10].

3.5 Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation

In 2009, Diehl, Ferreau, and Haverbeke released an article about numerical methods for solution of optimal control problems in real-time for nonlinear MPC and MHE. The focus is on discrete-time optimal control problems that are typically constrained, nonlinear, and high-dimensional [11]. The authors consider a general discrete-time system with algebraic constraints, leading to difference-algebraic formulations common in multiple shooting and collocation methods. NMPC and MHE problems are formulated as structured nonlinear programs (NLPs) involving equality constraints, path constraints, and terminal constraints.

The two main classes of Newton-type optimzation methods are compared, SQP and Interior Point (IP) methods. The authors mention the different types of SQP, mainly Classical SQP with exact or approximated Hessians (BFGS), Constrained Gauss-Newton method for least-squares problems, and Adjoint-based SQP using efficient gradient computations via reverse-mode AD. The IP methods replace inequality constraints with barrier terms. Smooth approximations allow Newton iterations on the modified KKT system. These methods are well-suited to large sparse systems but struggle with active-set changes. The two methods, Simultaneous and Sequential Methods, are compared. The sequential methods eliminate state variables via system simulation, optimizing

only over controls. They are lower-dimensional but lose problem structure. The simultaneous methods retain all variables and constraints, which enables structured linear algebra and better convergence for unstable and/or nonlinear systems.

Special techniques like condensing and band structure exploitation are presented for efficiently solving the linearized QPs that arise in each iteration. Techniques like Leineweber's partial reduction and Schlöder's trick help reduce computation when dealing with algebraic variables or large state spaces. To meet real-time constraints, several online nonlinear MPC and MHE schemes are discussed. Like the Shift Initialization and Tangential Predictors which leverage previous solutions for faster convergence. Continuation/GMRES which uses IP and GMRES for fast single-iteration updates. Real-Time Iteration which uses a single SQP step with preparation and feedback phases. Andvance Step NMPC solves the full IP problem ahead of time and applies a sensitivity-based correction. The Multi-level Real-time Iterations progressively improve optimality with increasing computational effort. The last mentioned scheme is the Online Active Set Strategies, which use parametric QP solutions for rapid updates across critial regions.

3.6 Successive Linearization Based Model Predictive Control of Variable Stiffness Actuated Robots

An article written by Zhakatayev, Rakhim, Adiyatov, Baimyshev and Varol, presents the technique of Successive Linearization Based MPC (SLMPC) for variable stiffness actuated robots [12]. The Variable Stiffness Actuated (VSA) robots have gained attention due to their promise for safety, energy efficiency, and adaptability in robotic systems. However, their nonlinear dynamics and complex mechanical structures pose significant challenges for real-time control. Nonlinear MPC has been a popular solution for trajectory tracking in VSA systems due to its ability to handle nonlinearities and constraints. Nevertheless, the high computational cost of NMPC limits its practicality for high-speed, multi-degree-of-freedom systems.

To address this, the concept of SLMPC has been proposed. This method linearizes the nonlinear system dynamics around the current state at every sampling instant and uses a discretized linear model within a traditional MPC framework. Such an approach reduces computational effort by solving a QP problem instead of a NLP, making it more suitable for systems with fast dynamics. In the article, the authors mention various domains for which Successive linearization techniques has been explored, such as: Chemical process control, Centrifugal compressors, Autonomous vehicles, Wind turbines, and Distributed parameter systems.

In robotics, prior benchmarks comparing NMPC and linearized MPC for systems such as autonomous surface vehicles and tank systems have demonstrated that linearized approaches can perform comparably to NMPC under certain conditions, especially when fast response is prioritized over model accuracy. This, at least, is mentioned in the article [12]. The discussed paper presents a detailed implementation of SLMPC for a reaction wheel-augmented sigle-link VSA robot, comparing it with an NMPC solution generated using the ACADO toolkit. The key aspects in the arcticle includes: A linearization and discretization process at 10 ms sampling intervals, QP problem formulation using the Eigen library and solved with qpOASES, cost function formulated using control input increments to improve smoothness and avoid steady-state errors, and constraints and bounds applied to both states and control inputs.

3.7 Real-time MHE-based nonliner MPC of a Pendubot system

An article written by Gulan, Salaj, Abdollahpouri, and Rohal-Ilkiv presents a real-time MHE-based NMPC scheme of a pendubot system [13]. The authors combine a NMPC with MHE which has become a compelling framework for the optimal control of underactuated, nonlinear systems. The Pendubot, a two-link robotic arm with only one actuated joint, presents a canonical benchmark for such studies due to its inherent instability, underactuation, and fast dynamics. Earlier work by Spong and Block [14] introduced the Pendubot as a suitable testbed for advance control strategies, motivating research in both swing-up and stabilization tasks.

Traditional control methods often rely on linearization and switching strategies, but these may underperform when dealing with nonlienar behaviors and constraints. With advancements in real-time optimization techniques, NMPC and MHE have emerged as powerful tools for solving constrained optimal control and estimation problems in real-time. Notably, Diehl [10], [11] introduced the Real-Time Iteration scheme, which allows solving nonlinear optimal control problems efficiently, as discussed earlier. Diehl's approach is particularly useful for fast systems like the Pendubot.

The use of the ACADO Toolkit [15] has facilitated the deployment of RTI-based NMPC and MHE on embedded hardware by providing auto-generated, efficient C-code solvers. Recent work by Gulan et al., according to the article [13], has demonstrated the real-time feasibility of such a combined NMPC and MHE scheme applied to the Pendubot, showing robust tracking of unstable equilibrium points using a full nonlinear model. The study emphasizes the importance of accurate velocity and parameter estimation using MHE to enhance NMPC performance. To handle the computational challenges, the authors exploit different QP solvers, like qpOASES for MHE and qpDUNES for NMPC [accessed March 2025], achieving sub-5 ms iteration times within a 10 ms control loop. These results highlight the maturity of real-time nonlinear control and estimation frameworks and their potential for broader application in robotics and mechatronics.

3.8 Implementation of Linear Model Predictive Control - Tutorial

In 2021, Michael Fink released an article about the implementation of a linear MPC [16]. The tutorial begins with an overview of MPC applied to discrete-time linear systems, where the MPC solves an optimization problem at each time-step, where the MPC predicts future system behavior over a finite horizon and computes the optimal control inputs accordingly. A key contribution of Fink's tutorial is the emphasis on recursive feasibility and stability, two critial properties for ensuring reliable performance. Recursive feasibility ensures that if the optimization problem is solvable at one time step, it remains solvable in future steps according to Fink [16]. This is achieved by incorporating a terminal constraint that defines a set of states from which a feasible solution always exists. For stability, the tutorial demonstrates the use of terminal cost term, designed to ensure that the predicted cost decreases over time, effectively stabilizing the closed-loop system.

The tutorial also includes a detailed discussion on how to reformulate the MPC optimization problem as a QP, which can be solved efficiently with QP solvers. To enable this, the system dynamics and constraints are lifted over the entire prediction horizon and expressed in matrix form. This reformulation is critical for real-time control applications, where computational efficiency is paramount.

3.9 A Simple and Efficient Tube-based Robust Output Feedback Model Predictive Control Scheme

An article by Lorenzetti and Pavone presents a tube-based robust output feedback MPC scheme aimed at controlling constrained linear systems in presence of bounded disturbances and measurements noise [17]. Ulike traditional MPC approaches that assume full-state availability and no disturbances, this method is designed for more realistic scenarios where only output measurements are available and both model and sensor noise are present. The proposed scheme improves upon existing robust feedback MPC methods by maintaining the same online computational complexity as nominal full-state MPC. This is achieved by constructing a single tube with a constant cross-section around the nominal trajectory using a Luenberger observer for state estimation. The control input is then defined as a sum of the nominal input and a correction term based on the estimation error, with feedback gain that ensures system stability.

A central contribution of the work is a novel method for computing robust positively invariant (RPI) sets used to tighten constraints and ensure robust feasibility. The authors combine ideas from existing techniques to compute RPI sets efficiently using a single linear program, resulting in smaller and less conservative sets compared to prior approaches. The method also avoids the need for

complex operations such as Minkowski sums or sequential constraint tightening. The effectiveness of the approach is demonstrated through simulations on both a simple synthetic system and a more complex wind energy conversion system with ten states. In both cases, the proposed method yields less conservative constraint tightening, improved performance, and reduced computational demands compared to established methods. The results support the method's suitability for real-time control in high-dimensional and fast evolving systems. The authors suggest future work could include extensions to time-varying tubes and further analysis of the theoretical properties of the proposed RPI computation method [17].

4 Theory

During this chapter the theoretical parts of the concepts relevant during this thesis will be covered, specifically the concept of Model Predictive Control (MPC). The goal is to give the reader vital information to better understand the implementations during the thesis. Firstly, the theory behind the magnetic levitation (Maglev) system will be introduced, before fully covering the theory behind MPC.

4.1 Optimization

MPC is fundamentally based on solving an optimization problem at each time step. Therefore, it is important to introduce optimization and methods used to solve optimization problems. The content in this section is mostly adapted from [18] and [19].

4.1.1 The Optimization Problem

Optimization is described as the minimization or maximization of an objective function with respect to its decision variables subject to constraints. The objective function is also referred to as the cost function in the context of MPC. This section will mainly cover the minimization problem as it is most relevant for MPC, and therefore this thesis. The notation used are adapted from [18] and [19].

An example is shown in Equation (1) to illustrate a minimization problem.

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad f(\mathbf{x}), \quad \text{s.t.} \quad \begin{cases} c_i(\mathbf{x}) = 0, & i \in \mathcal{E}, \\ c_i(\mathbf{x}) \geq 0, & i \in \mathcal{I} \end{cases} \tag{1}$$

The minimization problem in Equation (1) is also known as a *nonlinear programming problem* (NLP). In the example, $f(\mathbf{x})$ is the objective function, or cost function, $\mathbf{x}$ is the vector of decision variables, and $c_i(\mathbf{x})$ represents the constraints. The constraints are divided into equality and inequality constraints, distinguished by the index i, where $\mathcal{E}$ and $\mathcal{I}$ denote the sets of equality and inequality constraints, respectively. These constraints are functions of $\mathbf{x}$ that define some equations or inequalities which the decision variable must satisfy, and due to these constraints the decision variable must be selected from a subset of $\mathbb{R}^n$. This subset is called the feasible set or the feasible region, which is defined by the constraints c_i. This feasible set is defined as

$$\Omega = \left\{ \ \mathbf{x} \in \mathbb{R}^n \mid (c_i(\mathbf{x}) = 0, \ i \in \mathcal{E}) \ \wedge \ (c_i(\mathbf{x}) \geq 0, \ i \in \mathcal{I}) \ \right\}. \tag{2}$$

To further explain how to solve a minimization problem on the form (1) an example is shown in Equation (3).

$$\min_{\mathbf{x}\in\mathbb{R}^2} \quad f(\mathbf{x}) = x_1^2 + 3x_2, \quad \text{s.t.} \quad \begin{cases} c_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0, \\ c_2(\mathbf{x}) = x_1 + x_2 \geq 0 \end{cases} \tag{3}$$

From this example, we have that $\mathcal{E} = \{1\}$ and $\mathcal{I} = \{2\}$, which means that the feasible set is defined as

$$\Omega = \left\{ \ \mathbf{x} \in \mathbb{R}^2 \mid x_1^2 + x_2^2 = 1 \ \wedge \ x_1 + x_2 \geq 0 \ \right\}. \tag{4}$$

A geographical representation of this example is shown on Figure 4. Here, we see the feasible region Ω in the color red. The feasible region is the half circle when c_2 is satisfied. The feasible region in this example is not a half-disc because c_1 is an equality constraint and not an inequality constraint, which means that the optimal solution has to be on the circle c_1.

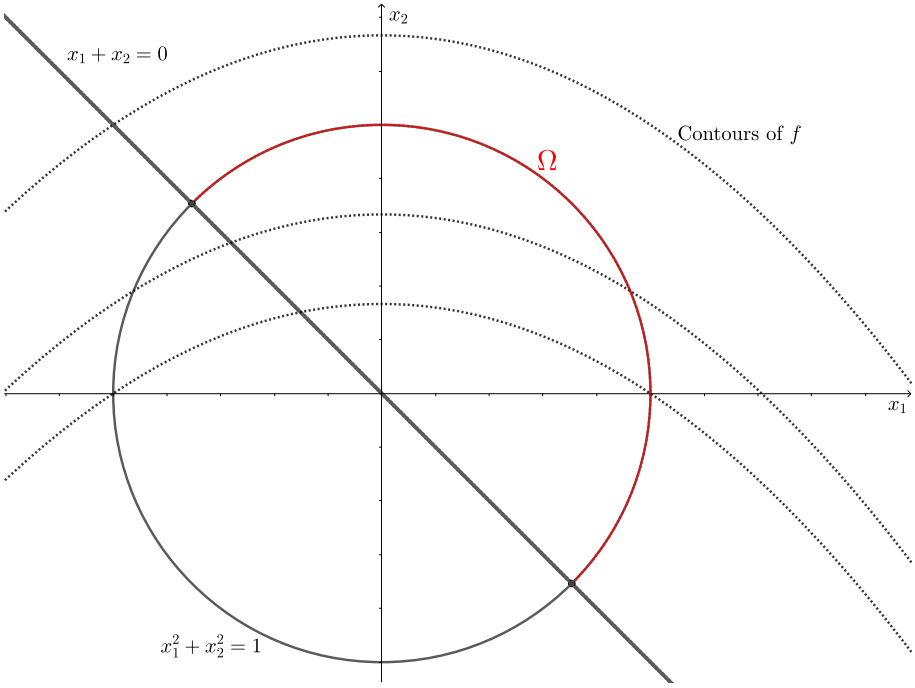


Figure 4: Geographical representation of the minimization problem example in Equation (3) adapted from [18, p. 5]

.

The feasible set for an optimization problem may be empty, $\Omega = \emptyset$, which means there is no solution, or no feasible points. The goal for the optimization problem is to find values for x_1 and x_2, in this case, that minimize the objective function f. A point $x^* \in \Omega$ is the solution if

$$f(x^*) \leq f(x), \quad \text{for all } x \in \Omega \tag{5}$$

A point that satisfies Equation (5) is called a global minimizer, and there may exist several such points. For cases where there only exists one global minimizer, it is called a strict global minimizer. A solution is called a local minimizer if the solution is a minimum within a neighborhood instead of the whole feasible set. A local minimizer is defined by

$$f(x^*) \leq f(x), \quad \text{for all } x \in ||x - x^*|| < \epsilon \tag{6}$$

where epsilon defines the neighborhood in the feasible set. In this case, a strict local minimizer is defined by the use of $<$ instead of $\leq$ in Equation (6).

The main focus during this thesis is nonlinear continuous optimization. There are several other types of optimization, like integer programming, which will not be covered during this thesis. An optimization problem is considered a nonlinear optimization problem (NLP) if either the objective function, f, or the constraints, c_i, are nonlinear. The most relevant type of NLP used in this thesis is quadratic programming (QP), where the objective function is quadratic and the constraints are linear. A typical QP is formulated as

$$\min_{x \in \mathbb{R}^n} \quad q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T G\mathbf{x} + \mathbf{x}^T c, \quad \text{s.t.} \quad \begin{cases} a_i^T \mathbf{x} = b_i, & i \in \mathcal{E} \\ a_i^T \mathbf{x} \geq b_i, & i \in \mathcal{I} \end{cases} \tag{7}$$

where the Hessian matrix, G, is a symmetric $n \times n$ matrix, $\frac{1}{2}\mathbf{x}^T G\mathbf{x}$ is the quadratic term, and $\mathbf{x}^T c$ is the linear term. When the Hessian G is positive semidefinite, $G \geq 0$, the QP can be said to be a convex QP. In cases where the Hessian is positive definite, $G > 0$, the QP is a strictly convex QP. A convex QP is easy to solve and most QP solvers assume convexity. The term convex will be described in more detail below, but first a general case of QP is whenever there are only equality constraints present. Such a case for QP is called an equality-constrained QP and can be formulated as

$$\min_{\mathbf{x} \in \mathbb{R}^n} \quad q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T G\mathbf{x} + \mathbf{x}^T c, \quad \text{s.t.} \quad A\mathbf{x} = b \tag{8}$$

where A is the $m \times n$ Jacobian matrix of the constraints with $m \leq n$, whose rows are $a_i^T, i \in \mathcal{E}$. The vector b is a vector in $\mathbb{R}^m$ whose components are $b_i, i \in \mathcal{E}$.

The term convex, where the Hessian is positive semidefinite for QPs, can be said for both functions and sets. The definition for convexity is obtained from [19, p. 8]. A set $\mathbf{S} \in \mathbb{R}^n$ is a convex set if for any two points, $x, y \in \mathbf{S}$, we have $\alpha x + (1-\alpha)y \in \mathbf{S}$ for all $\alpha \in [0,1]$. The function f is a convex function if its domain $\mathbf{S}$ is a convex set and if the two points $x, y \in \mathbf{S}$ the following property is satisfied

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \quad \text{for all } \alpha \in [0,1] \tag{9}$$

The function f is strictly convex if the inequality is strict whenever $x \neq y$. Whenever the objective function in Equation (1) and the feasible region (2) are both convex, then any local solution to the optimization problem is a global solution. A special case of the optimization problem in Equation (1) is whenever

- the objective function is convex
- the equality constraints functions $c_i, i \in \mathcal{E}$, are linear
- the inequality constraints functions $c_i, i \in \mathcal{I}$, are concave

The term convex programming is used to describe this special case. The term concave can be used for a function f if $-f$ is convex.

4.1.2 Unconstrained and Constrained Optimization - Optimality Conditions

When an optimization problem, such as Equation (1), is unconstrained the optimization problem is on the form

$$\min_{\mathbf{x} \in \mathbb{R}^n} \quad f(\mathbf{x}) \tag{10}$$

which is a minimization problem where the objective function f is minimized without any restrictions. The first and second order necessary conditions for optimality for an unconstrained

optimization problem is explained in detail in [19, p. 14-16]. The First-Order Necessary Conditions is defined as If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.

The first condition yields x^* as a stationary point, but we don't know if it's a minimum, maximum or a saddle point. This is where the Second-Order Necessary Conditions come in, which is defined as: If x^* is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

These conditions are necessary conditions for optimality by assuming x^* is a local minimizer and further proving the facts about $\nabla f(x^*)$ and $\nabla^2 f(x^*)$. These conditions become sufficient if $\nabla^2 f(x^*)$ is positive definite, which yields that x^* is a strict local minimizer.

When it comes to constrained optimization problems, on the other hand, we use the Karush-Kuhn-Tucker (KKT) conditions. The KKT conditions are a set of first order conditions for which a solution x^* must satisfy in order to be a minimizer, and they take use of a Lagrangian function. The Lagrangian function for a general minimization problem as in Equation (1) is defined as

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(\mathbf{x}). \tag{11}$$

In order for the KKT-conditions to hold, it is a necessity that the Linear Independence Constraint Qualification (LICQ) holds, which is the case if $\mathbf{x}$ is a feasible point and the set of active constraint gradients $\{\nabla c_i(\mathbf{x}), i \in \mathcal{A}(\mathbf{x})\}$ is linearly independent, where the active set $\mathcal{A}(\mathbf{x}) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(\mathbf{x}) = 0\}$ The First-Order Necessary Conditions for a constrained optimization problem is obtained from [19, p. 321] and is defined as follows: Suppose that x^* is a local solution of Equation (1), that the functions f and c_i are continuously differentiable, and that the LICQ holds at x^*. Then there is a Lagrange multiplier vector λ^*, with components $\lambda_i^*, i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*)

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \tag{12a}$$
$$c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E}, \tag{12b}$$
$$c_i(x^*) \geq 0, \quad \text{for all } i \in \mathcal{I}, \tag{12c}$$
$$\lambda_i^* \geq 0, \quad \text{for all } i \in \mathcal{I}, \tag{12d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E} \cup \mathcal{I} \tag{12e}$$

.

The Second-Order Necessary conditions for constrained optimization problems uses the critical cone $\mathcal{C}(x^*, \lambda^*)$. Before defining the Second-Order conditions we obtain a set of linearized feasible directions $\mathcal{F}(\mathbf{x})$, which is defined as

$$\mathcal{F}(\mathbf{x}) = \left\{ \mathbf{d} \;\middle|\; \begin{array}{ll} \mathbf{d}^\intercal \nabla c_i(\mathbf{x}) = 0, & \text{for all } i \in \mathcal{E}, \\ \mathbf{d}^\intercal \nabla c_i(\mathbf{x}) \geq 0, & \text{for all } i \in \mathcal{A}(\mathbf{x}) \cap \mathcal{I} \end{array} \right\} \tag{13}$$

given the active constraint set $\mathcal{A}(\mathbf{x})$ and that $\mathbf{x}$ is a feasible point. Now, given the set of linearized feasible directions $\mathcal{F}(x^*)$ and the Lagrange multiplier λ^* satisfying the KKT-conditions the critical cone $\mathcal{C}(x^*, \lambda^*)$ is defined as

$$\mathcal{C}(x^*, \lambda^*) = \{\omega \in \mathcal{F}(x^*) \mid \nabla c_i(x^*)^T \omega = 0, \text{ all } i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0\} \tag{14}$$

where ω is the directions in the critical cone which follows the inequality and equality constraints even if there would be made some changes to the objective. The Second-Order Sufficient Conditions

ensures that x^* is a strict local minimizer for the optimization problem, such as in Equation (1), and is defined as

$$\omega^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)\omega \geq 0, \quad \text{for all } \omega \in \mathcal{C}(x^*, \lambda^*) \tag{15}$$

4.1.3 Duality

The duality theory shows how we can define an optimization problem as an alternative problem. These two perspectives of an optimization problem is called the dual-problem and the primal-problem. A typical primal-problem is the optimization problem presented in Equation (1). The reason for the use of the duality theory is that in some cases the dual-problem is easier to solve than the primal-problem. When defining the duality problem in this section it will be as in [19, p. 343] where there are only inequality constraints, meaning $\mathcal{E} = \emptyset$. Given the optimization problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \quad f(\mathbf{x}) \quad \text{s.t. } c_i(\mathbf{x}) \geq 0, \tag{16}$$

, with the Lagrangian function

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda^T c_i(\mathbf{x}) \tag{17}$$

we can define the dual objective function as

$$q(\lambda) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) \tag{18}$$

which further defines the dual problem as

$$\max_{\lambda \in \mathbb{R}^n} \quad q(\lambda) \quad \text{s.t. } \lambda \geq 0 \tag{19}$$

4.1.4 Solving Optimization Problems

Solving optimization problems vary from explicit solutions to complex iterative schemes. The explicit solutions are relevant for simple problems, such as the linear quadratic control (LQ) problem [18]. This thesis will mainly focus on iterative schemes which is relevant for MPC. Until now, the necessary and sufficient conditions for a solution x^* to be a minimizer has been covered. The vectors $\mathbf{d}$ and ω in Equations (13) and (14) are feasible directions which satisfy the constraints for the optimization problem. This section will cover how to find the optimal point of the problem, x^*. For NLPs, which will be focused on in this thesis, there are several practical strategies for solving the optimization problem. Such strategies are *Line Search Methods*, *Trust-Region Methods* and *Sequential Quadratic Programming (SQP)*. During this section the gradients of the given objective function, $\nabla f(x_k)$ and $\nabla^2 f(x_k)$, will be shortened to ∇f_k and $\nabla^2 f_k$ for simplicity.

Firstly, we introduce the Line Search Method, which is an iterative method for solving a NLP. The material used to explain the line search methods is adapted from Chapter 3 in [19]. In a line search method we compute a search direction $\mathbf{p}_k$ and a positive scalar α_k at each iteration. The iteration for a such method is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{20}$$

where α_k is called the step length. This type of strategy is a solver for an optimization problem such as Equation (1), which means that the goal is to minimize the objective function f. To guarantee a reduction in the objective function at each iteration the search direction is required to be a *descent direction*, for which $\mathbf{p}_k^T \nabla f_k < 0$. This property guarantee reduction in the objective function along this direction. The general form of the search direction has the form

$$\mathbf{p}_k = -\mathbf{B}_k^{-1}\nabla f_k \tag{21}$$

where $\mathbf{B}_k$ is a symmetric and nonsingular matrix. There are several versions of the line search methods, and what categorizes the different versions is how $\mathbf{B_k}$ is chosen. In the steepest descent method, for instance, the matrix $\mathbf{B}_k$ is simply the identity matrix. Another typical line search method is the Newton's method, where $\mathbf{B}_k$ is the exact Hessian matrix, $\nabla^2 f_k$. The third example of a line search method is the so called quasi-Newton methods. Here the matrix $\mathbf{B}_k$ is an approximation of the Hessian, which is updated at each iteration. If the search direction is given by the Equation (21), and the used matrix $\mathbf{B}_k$ is positive definite, we can confirm that $\mathbf{p}_k$ is a descent direction by

$$\mathbf{p}_k^T\nabla f_k = -\nabla f_k^T\mathbf{B}_k^{-1}\nabla f_k < 0 \tag{22}$$

A challenging part of these line search methods is to chose the step length α_k. There are several factors in play for the choice of the step length. Two factors is that we want the step length to guarantee a reduction in the objective function f, but as fast as possible. The line search strategies, introduced above, find the step length α_k by trying out a sequence of step length candidates. The acceptable step length is decided when certain conditions are met. This method happens in two stages. In the first phase an interval of desirable step lengths are found, where a good step length is computed within this interval in the second phase.

As stated earlier, we want the step length to give a reduction in the objective function f,

$$f(\mathbf{x}_k + \alpha_k\mathbf{p}_k) < f(\mathbf{x}_k) \tag{23}$$

However, to guarantee convergence to the optimal solution x^* this requirement is not enough. To enforce a sufficient decrease in the objective function f there are some conditions which the line search method must satisfy. These conditions are called the *Armijo condition* and the *Curvature condition*. The Armijo condition is given by the following inequality

$$f(\mathbf{x}_k + \alpha\mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1\alpha\nabla f_k^T\mathbf{p}_k \tag{24}$$

where $c_1 \in (0, 1)$ is a constant. This inequality states that the reduction in f should be proportional to both the directional derivative $\nabla f_k^T\mathbf{p}_k$ and the step length α_k. The Armijo condition is also not enough by itself, which is why the Curvature condition is introduced. This condition requires the step length α_k to satisfy the following inequality

$$\nabla f(\mathbf{x}_k + \alpha_k\mathbf{p}_k)^T\mathbf{p}_k \geq c_2\nabla f_k^T\mathbf{p}_k \tag{25}$$

where c_2 is also a constant $0 < c_1 < c_2 < 1$.

4.1.5 SQP

One of the most effective methods for solving NLPs is called *Sequential Quadratic Programming* (SQP), where the methods essence is solving quadratic subproblems at each iteration. This method can be used for both line search methods and trust-region methods. The SQP method is explained in depth in Chapter 18 in [19], which is where the material for this section is adapted from.

SQP is based on approximating the original nonlinear problem at each iteration by a QP subproblem, which models the objective function with a quadratic function with a quadratic approximation and the constraints with linear approximations. The idea is to obtain the step $\mathbf{p}_k$ in each iteration which will bring the problem closer to the solution. The QP subproblem constructed minimizes the second-order Taylor series approximation of the original NLP's Lagrangian [19, p. 532]

The QP subproblem is obtained by linearizing the constraints around the current timestep k, which results in a general QP problem as shown in Equation (26).

$$\min_{\mathbf{p}} \quad f_k + \nabla f_k^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \nabla_{xx}^2 \mathcal{L}_k \mathbf{p}_k \tag{26a}$$

$$\text{s.t.} \quad \nabla c_i(\mathbf{x}_k)^T \mathbf{p}_k + c_i(\mathbf{x}_k) = 0, \;\; i \in \mathcal{E} \tag{26b}$$

$$\nabla c_i(\mathbf{x}_k)^T \mathbf{p}_k + c_i(\mathbf{x}_k) \geq 0, i \in \mathcal{I} \tag{26c}$$

where $\mathcal{L}_k$ is defined as in Equation (11). The next iterate from solving the QP subproblem is given by $(\mathbf{x}_k + \mathbf{p}_k, \lambda_{k+1})$, where $\mathbf{p}_k$ and λ_{k+1} are the solution and the corresponding Lagrange multipliers from the QP in Equation (26). This QP subproblem can be solved as a standard QP problem. The method therefore identifies an active set of constraints, which is treated as equality-constrained in subsequent iterations. The main approaches for SQP are IQP and EQP. IQP is based on solving the full inequality constrained QP at each step, while the EQP solves the equality constrained QPs using a working set of active constraints.

4.2 Dynamic Optimization

Dynamic optimization concerns the process of making optimal decisions over time for systems whose behavior evolves, typically as a result of both current conditions and past actions. Unlike static optimization, which addresses decisions at a single point in time, dynamic optimization explicitly consider temporal dynamics and the interdependence between states and control inputs across time steps. This is crucial in many engineering applications, such as chemical process control, energy systems, robotics, and trajectory planning, where system states and decisions unfold sequentially. Most of this content is adapted from Chapter 3 [18], unless otherwise specified.

In modeling dynamic systems, discrete-time representations are often used, especially in digital control and optimization contexts. A discrete-time system can be represented by the state transition equation in Equation (27).

$$\mathbf{x}_{k+1} = g(\mathbf{x}_k, \mathbf{u}_k) \tag{27}$$

Here, $\mathbf{x}_k$ denotes the state at timestep k, and $\mathbf{u}_k$ is the control input at time step k. g is a function which defines how the states evolves. This function can be nonlinear in general, although linear approximations are frequently used. If g is linear and time-invariant, the system is described by Equation (28).

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{28}$$

Here, $\mathbf{A}$ and $\mathbf{B}$ are constant matrices. In some cases, the system may vary over time, yielding a time-varying model with $\mathbf{A}_k$ and $\mathbf{B}_k$. The objective in dynamic optimization is to determine a sequence of control inputs that minimizes a performance criterion over a finite horizon. A commonly used objective function is given by Equation (29).

$$\mathbf{J} = \sum_{k=0}^{N-1} f_k(\mathbf{x}_{k+1}, \mathbf{u}_k) \tag{29}$$

Here, the objective function $\mathbf{J}$ captures the cumulative costs such as deviations from a desired state, energy usage, economic costs, or combinations of all of them. In practice, quadratic forms are often adopted because they are computationally convenient and capture the trade-off between state error and control effort effectively. A typical quadratic cost function is given in Equation (30).

$$f_k(\mathbf{x}_{k+1}, \mathbf{u}_k) = \frac{1}{2}\mathbf{x}_{k+1}^T \mathbf{Q}\mathbf{x}_{k+1} + \frac{1}{2}\mathbf{u}_k^T \mathbf{R}\mathbf{u}_k \tag{30}$$

Here, f_k in Equation (29) is substituted with the quadratic term in Equation (30). $\mathbf{Q}$ and $\mathbf{R}$ are two positive semidefinite weighting matrices that define the importance of minimizing state deviation and control usage respectively. To ensure realistic solutions, constraints are typically imposed on states and control inputs. These can include bounds on the allowable state values, limits on control inputs, and bounds on rate of change of control inputs to avoid excessive wear or actuator saturation. The dynamic model itself is also enforced as an equality constraint in the optimization problem. The full dynamic optimization problem can thus be expressed as minimizing the objective function subject to the system dynamics and various constraints. If the dynamics are linear and the cost is quadratic, the resulting problem is a convex QP, which can be solved efficiently. However, when the system dynamics are nonlinear, the problem becomes a NLP, which is generally non-convex and computationally more demanding. The focus in this thesis is QP formulation as will be discussed further in Section 4.3. From the objective function given in Equation (30), a given finite horizon N, and constraints added to the dynamic optimization problem, a typical dynamic quadratic optimization problem is given by Equation (31).

$$\min_{\mathbf{z}\in\mathbb{R}^n} \quad f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2}x_{k+1}^T Q x_{k+1} + \frac{1}{2}u_k^T R u_k \tag{31a}$$

$$\tag{31b}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k, \quad k = 0, ..., N-1 \tag{31c}$$

$$x^{low} \leq x_k \leq x^{high}, \quad k = 1, ..., N \tag{31d}$$

$$u^{low} \leq u_k \leq u^{high}, \quad k = 0, ..., N-1 \tag{31e}$$

$$x_0 = \text{given} \tag{31f}$$

A fundamental concept in dynamic optimization is Bellman's Principle of Optimality, introduced by Richard Bellman in his seminal 1954 paper *The Theory of Dynamic Programming* [20]. The principle states that an optimal policy has the property that whatever the initial state and initial decision are, the remaining decision must constitute an optimal policy with regard to the state resulting from the first decision" [20, p. 504].

This principle provides the basis for the recursive structure of many optimal control formulations. From this principle, Bellman derived the now-famous Bellman equation, which expresses the value of a decision problem at a given time in terms of the value at the next step. For a discrete-time system, this is given by Equation (32).

$$\mathbf{V}_k(\mathbf{x}_k) = \min_{\mathbf{u}_k} \left[f_k(\mathbf{x}_{k+1}, \mathbf{u}_k) + \mathbf{V}_{k+1}(\mathbf{x}_{k+1} \right] \tag{32}$$

Here, $\mathbf{V_k}(\mathbf{x_k})$ is the optimal cost-to-go function from state $\mathbf{x}_k$. The recursion begins from the terminal cost $\mathbf{V}_N(\mathbf{x}_N)$, which is typically defined as zero or a penalty on the final state. Bellman's equation forms the theoretical foundation for dynamic programming, as discussed earlier in this section. It provides a systematic method for breaking down complex, multi-stage decision problems into simpler subproblems. This recursive structure has since been adapted in many areas of optimal control and planning, including Model Predictive Control which will be discussed in Section 4.3. While the exact solution to Bellman's equation becomes computationally infeasible for high-dimensional systems its conceptual framework remain vital.

4.2.1 Control

To integrate control with dynamic optimization, it is essential to introduce foundational concepts from control theory, namely *controllability* and *observability*. These properties determine whether

a system's state can be manipulated or reconstructed over time and are fundamental to the design of both state-feedback and output-feedback controllers. Starting with controllability, consider a continuous-time linear time-invariant (LTI) system defined by Equation (33).

$$\dot{x} = Ax + Bu \tag{33}$$

Here, the state matrices $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ are time-invariant, ergo LTI. Controllability for a given system, as in Equation (33), refers to the ability to move the systems states to a desired setpoint in finite time. In mathematical terms, a system is controllable if a setpoint $\mathbf{z}$ can be reached from any pair of states x [21, p. 23]. An LTI system, given as Equation (33), is therefore said to be controllable if there exists a finite time n and a control sequence $\mathbf{U} = (u_0, u_1, ..., u_{n-1})$ that can reach any z from any x. With the controllability matrix given as

$$\mathcal{C} = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \tag{34}$$

the system (A, B) is said to be controllable if and only if the rows of $\mathcal{C}$ are linearly independent. In other words, (A, B) is controllable if and only if $\text{rank}(\mathcal{C}) = n$ [21, p. 23].

The next concept for control systems are observability. In observability the control inputs are irrelevant, and we therefore consider the following LTI with zero input

$$\dot{x} = Ax \tag{35}$$
$$y = Cx \tag{36}$$

An LTI, as in Equation (35), is said to be observable if there exists a finite n, such that for every $\mathbf{x}$, n measurements $(y_0, y_1, ..., y_{n-1})$ distinguish uniquely $\mathbf{x}$. In other words, observability means the systems ability to observe all states during a finite time n. With $A \in \mathbb{R}^{n \times n}$ and $C \in \mathbb{R}^{p \times n}$ the observability matrix is given by

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{37}$$

and by the observability matrix $\mathcal{O}$ we have that the system is observable if and only if the columns are linearly independent. In other words, the system (A, C) is observable if and only if $\text{rank}(\mathcal{O}) = n$ [21, p. 42]. This property is essential for designing state estimators like Kalman filters or MHE, which is discussed in Section **??**.

4.2.2 Linear Quadratic Control

A Linear Quadratic Controller or, as we will call it in this thesis, Linear Quadratic Regulator (LQR), can in a lot of ways be compared to a linear MPC which will be discussed in Section 4.3. A LQR is a solution to a open-loop optimization problem even though the controller works as a state feedback controller. The material during this section is mostly adapted from [18] with adapted notation to align with this thesis.

The main aspect of a LQR is to calculate a constant gain $\mathbf{K}$ which is used for off-line state-feedback control. The control law used by a LQR is given by Equation (38).

$$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k \tag{38}$$

where $\mathbf{x}_k$ is the current state. In most cases the current state is unknown, which requires a state estimator. A LQR combined with a state estimator is also known as a Linear Quadratic Gaussian (LQG) [21]. We will use the term LQR during this thesis, even the focus will be on a LQR combined with a state estimator, and the new control law with state estimation is given by Equation (39).

$$\mathbf{u}_k = -\mathbf{K}\hat{\mathbf{x}}_k \tag{39}$$

where $\hat{\mathbf{x}}_k$ is the state estimation provided by a MHE for example. This results in a off-line output-feedback controller, which will be the focus from now on. A LQR control law will stabilize the given system if the system model is accurate and $(\mathbf{A}, \mathbf{B})$ is controllable [18, p. 65]. LQR can be used in an infinite horizon case and finite horizon. An infinite-horizon LQR is based on solving an infinite-horizon open-loop optimization problem as in Equation (40)

$$\min_{\mathbf{z}} \quad \sum_{k=0}^{\infty} \frac{1}{2}\mathbf{x}_{k+1}^T \mathbf{Q}\mathbf{x}_{k+1} + \frac{1}{2}\mathbf{u}_k^T \mathbf{R}\mathbf{u}_k \tag{40a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{40b}$$

$$\mathbf{x}_0 = \text{given} \tag{40c}$$

where $\mathbf{Q} \geq 0$, $\mathbf{R} \geq 0$ and $\mathbf{z}^T = (\mathbf{u}_0^T, ..., \mathbf{u}_\infty^T, \mathbf{x}_1^T, ..., \mathbf{x}_\infty^T)$ [18, p. 64]. $\mathbf{Q}$, $\mathbf{R}$ are weight matrices assigned to penalize the states and control inputs. The solution of Equation (40) is the control law given by Equation (39), where the gain matrix $\mathbf{K}$ is derived by

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}(\mathbf{I} + \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P})^{-1}\mathbf{A} \tag{41a}$$

$$\mathbf{P} = \mathbf{Q} + \mathbf{A}^T\mathbf{P}(\mathbf{I} + \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P})^{-1}\mathbf{A} \tag{41b}$$

$$\mathbf{P} = \mathbf{P}^T \geq 0 \tag{41c}$$

where Equation (42b) is known as the *algebraic Riccati equation* and $\mathbf{P}$ is the *Riccati matrix*. Note that the formulation of the optimization problem in Equation (40) is the infinite horizon LQR. In cases with a finite horizon the formulation is almost the same, except for taking the sum over a horizon N instead of ∞. In the case of a finite-horizon LQR the gain matrix $\mathbf{K}_k$ and the Riccati matrix $\mathbf{P}_k$ are both time-variant. The Equations in (41), in case of a finite horizon LQR, is formulated as

$$\mathbf{K}_k = \mathbf{R}_k^{-1}\mathbf{B}_k^T\mathbf{P}_{k+1}(\mathbf{I} + \mathbf{B}_k\mathbf{R}_k^{-1}\mathbf{B}_k^T\mathbf{P}_{k+1})^{-1}\mathbf{A}_k \tag{42a}$$

$$\mathbf{P}_k = \mathbf{Q}_k + \mathbf{A}_k^T\mathbf{P}_{k+1}(\mathbf{I} + \mathbf{B}_k\mathbf{R}_k^{-1}\mathbf{B}_k^T\mathbf{P}_{k+1})^{-1}\mathbf{A}_k \tag{42b}$$

$$\mathbf{P}_N = \mathbf{Q}_N \tag{42c}$$

where all matrices are time-variant in this general formulation. $\mathbf{Q_N}$ is the terminal state weight matrix for the terminal state $\mathbf{x}_N$.

4.3 Model Predictive Control

Having introduced the foundations of optimization and dynamic systems, we now focus on the central topic of this thesis, Model Predictive Control (MPC). MPC is a control strategy that computes optimal control actions by solving a finite-horizon dynamic optimization problem at each timestep, incorporating current state information and system constraints. In other words, the MPC uses the current state to predict the future behavior of the system at each timestep, and computing the optimal control input for that optimal predicted solution.

In Section 4.2, we formulated an open-loop optimization problem as a quadratic program (QP), given by Equation (31). This formulation corresponds to an open-loop optimization problem, where the optimal control sequence is computed once at time $k = 0$ and applied over the entire prediction horizon N without re-evaluation. In contrast, MPC operates in closed-loop mode, which means that the optimization problem is solved recurrently at every timestep with the use of the most recent state or output measurements to update predictions and control decisions. This leads to a feedback-based strategy that enhances robustness and adaptability. A standard state-feedback MPC procedure is summarized in Algorithm 1 adapted from [p. 39][18]. Most of the material adapted for this section is based on [18], unless otherwise specified.

Algorithm 1 State Feedback MPC Procedure

```
for k = 0, 1, 2, ... do
    Get the current state x_k
    Solve a dynamic optimization problem over the prediction horizon N
     with x_k as the initial condition
    Apply the first control input u_k from the optimal sequence
end for
```

As illustrated in the algorithm, the current state $\mathbf{x}_k$ is used as the initial condition to solve the optimization problem, as in Equation (31), and only the first control input $\mathbf{u}_k$ is applied to the system. The procedure is repeated at the next step with updated state. It is important to note that the constraints, as defined in Equation (31), are aligned with the current timestep k and shift forward in time as the algorithm progresses. Specifically, the constraints are applied over the intervals $k, ..., k + N - 1$ for control inputs and $k + 1, ..., k + N$ for predicted states. Similarly, the prediction horizon is re-centered around the current timestep at each iteration [18, p. 40].

The procedure in Algorithm 1 assumes full access to the system's current state. However, in many practical applications, only output measurements are available. In such cases, output-feedback MPC is used, where the optimization is based on estimated states $\hat{\mathbf{x}}_k$, obtained from past measurements. The corresponding procedure is shown in Algorithm 2, adapted from [18, p. 41].

Algorithm 2 Output Feedback MPC Procedure

```
for k = 0, 1, 2, ... do
    Compute an estimate of the current state x̂_k based on
      measured data from the past until k
    Solve a dynamic optimization problem over the prediction horizon N
      with the estimate x̂_k as initial condition
    Apply the first control input u_k from the optimal sequence
end for
```

The essential difference between the two approaches is that state-feedback MPC assumes perfect state measurements, while output-feedback MPC relies on an observer or estimator, such as a Kalman Filter or MHE, to reconstruct the system state.

MPC has become a widely adopted control methodology, particularly in process industries, due to its ability to handle multivariable systems and explicitly incorporate constraints on both states and inputs. One of its key advantages is its applicability to complex systems where designing an offline control law is intractable or impractical [22]. MPC provides a flexible and powerful framework for solving such control problems in real-time, with constraint satisfaction and performance optimization built directly into the controller design.

4.3.1 Linear MPC

A linear MPC is an MPC where the dynamic optimization problem solved at each timestep is a QP with linear constraints, such as the one in Equation (**??**). A slight change in the general form of the QP problem for linear MPC is to include Δu_k as a term in the objective function as well as in the constraints. This gives us the dynamic optimization problem formulated as

A linear MPC strategy involves solving a quadratic program (QP) with linear dynamics and linear constraints at each time step. A typical linear MPC formulation includes not only state and control penalties but also penalties on control input changes, denoted as $\Delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$, to promote smoother control actions and avoid actuator saturation. A general form of the linear MPC formulation is given by Equation (43).

$$
\begin{aligned}
\min_{\mathbf{z}\in\mathbb{R}^n} \quad & f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2} x_{k+1}^T Q_{k+1} x_{k+1} + d_{xk+1} x_{k+1} + \frac{1}{2} u_k^T R_k u_k + d_{uk} u_k + \frac{1}{2} \Delta u_k^T R_{\Delta k} \Delta u_k && \text{(43a)}\\
\text{s.t.} \quad & x_{k+1} = A_k x_k + B_k u_k && \text{(43b)}\\
& x_0 = \text{given} && \text{(43c)}\\
& x^{low} \leq x_k \leq x^{high} && \text{(43d)}\\
& u^{low} \leq u_k \leq u^{high} && \text{(43e)}\\
& -\Delta u^{high} \leq \Delta u_k \leq \Delta u^{high} && \text{(43f)}\\
\text{where} \quad & Q_k \geq 0 && \text{(43g)}\\
& R_k \geq 0 && \text{(43h)}\\
& R_{\Delta k} \geq 0 && \text{(43i)}\\
& \Delta u_k = u_k - u_{k-1} && \text{(43j)}\\
& \mathbf{z}^T = (x_1^T, ..., x_N^T, u_0^T, ..., u_{N-1}^T) && \text{(43k)}\\
& n = N(n_x + n_u) && \text{(43l)}
\end{aligned}
$$

A key concern in linear MPC is whether the optimization problem remains feasible under all conditions, particularly in the presence of disturbances, model mismatch, or tight constraints. For instance, if a disturbance causes the predicted state to exceed the defined state bounds at time k, no feasible control input may exist to satisfy the constraints. To address such issues and maintain feasibility, *slack variables* can be introduced. These are non-negative auxiliary variables that relax hard constraints, enabling the optimizer to prioritize feasibility while penalizing constraints violations in the cost function [18, p. 44-45]. These slack variables are applied to the linear MPC model, in Equation (43), by adding them to the state and/or input bounds depending on what constraints we want to soften. The updated linear MPC formulation is given by Equation (44).

$$
\begin{aligned}
\min_{\mathbf{z}\in\mathbb{R}^n} \quad & f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2} x_{k+1}^T Q_{k+1} x_{k+1} + d_{xk+1} x_{k+1} + \frac{1}{2} u_k^T R_k u_k + d_{uk} u_k && \text{(44a)}\\
& + \frac{1}{2} \Delta u_k^T R_{\Delta k} \Delta u_k + \rho^T \epsilon + \frac{1}{2} \epsilon^T \mathcal{S} \epsilon && \text{(44b)}\\
\text{s.t.} \quad & x_{k+1} = A_k x_k + B_k u_k && \text{(44c)}\\
& x_0 = \text{given} && \text{(44d)}\\
& x^{low} - \epsilon \leq x_k \leq x^{high} + \epsilon && \text{(44e)}\\
& u^{low} \leq u_k \leq u^{high} && \text{(44f)}\\
& -\Delta u^{high} \leq \Delta u_k \leq \Delta u^{high} && \text{(44g)}\\
\text{where} \quad & \epsilon \in \mathbb{R}^{n_x} \geq 0 && \text{(44h)}\\
& \rho \in \mathbb{R}^{n_x} \geq 0 && \text{(44i)}\\
& \mathcal{S} \in \text{diag}\{s_1, ..., s_{n_x}\},\ s_i \geq 0,\ i = \{1, ..., n_x\} && \text{(44j)}
\end{aligned}
$$

The slack variables ϵ incur both linear and quadratic penalties, and should ideally remain zero unless constraint violations are unavoidable. In some implementations, slack variables are applied only to the first few steps of the prediction horizon, as violations are more likely to occur early in the trajectory.

A longstanding challenge in MPC is the lack of a general stability theory for constrained formulations [23, p. 13]. Even in unconstrained systems, where no input limits are imposed, instability can arise if the cost function or horizon is poorly chosen. However, asymptotic stability can often be ensured through careful selection of the weight matrix $\mathbf{R}$ or by increasing the prediction horizon length N. This special, unconstrained case is not relevant for this thesis and will not be discussed any further. A central concept for stability in constrained MPC cases however, started its development in the early eighties and late nineties [23, p. 13]. In short terms, this stability concept idea is to reformulate the dynamic optimization problem which will guarantee stability for the system. Such general approach might be to add a *terminal state constraint* and a *terminal state weight* to the optimization problem. In other words, we include an additional constraint or weight on the terminal, final, state $\mathbf{x}_N$ for the optimization problem. A terminal state constraint can be formulated as in Equation (45).

$$x_N \in \mathbb{X}_f \tag{45}$$

Here, the set $\mathbb{X}_f$ is the set where $x_N = 0$ [21, p. 142]. However, adding the constraint (45) in the optimization problem might cause infeasibility, specifically for problems with shorter prediction horizons [18, p. 47].

The most relevant algorithm for this thesis is Algorithm 2 the output feedback MPC. The project in this thesis is in collaboration with another student working with estimation, more precisely Moving Horizon Estimation (MHE) which will be discussed more later.

4.3.2 Reference Tracking

Thus far, we have focused on formulations of linear MPC without an explicit consideration of reference signals. In practice, however, one of the most common control objectives is *reference tracking* that is, ensuring that the system states or outputs follow a desired setpoint or trajectory over time [24]. In context of dynamic optimization and MPC, reference tracking typically involves penalizing the deviation of the system state $\mathbf{x}_k$ and control input $\mathbf{u}_k$ from corresponding reference values $\mathbf{x}_k^{ref}$ and $\mathbf{u}_k^{ref}$. This modifies the cost function to incorporate reference terms directly. A general form of the objective function with reference tracking is given by Equation (46).

$$f(\mathbf{x}, \mathbf{u}) = \|\mathbf{x} - \mathbf{x}^{ref}\|_Q^2 + \|\mathbf{u} - \mathbf{u}^{ref}\|_R^2 \tag{46}$$

This formulation is equivalent to the formulation in Equation (31). However, in the referred equation the norm has been calculated out, also there is no reference point for the state and input. This means that the reference in Equation (46) is zero. The goal in reference tracking problems is to compute control inputs that minimize the deviation from these reference trajectories across a finite prediction horizon [24]. The formulation in Equation (46) is for a reference setpoint, but might also be used for a reference trajectory where the reference point is updated at each timestep.

In many applications, the reference is time-varying, leading to a reference trajectory. Additionally, it is often not the full state that is tracked, but a controlled variable, a function of the state denoted as $\gamma_k = H\mathbf{x}_k.$, where $H \in \mathbb{R}^{p\times n}$ is the output or selection matrix. Suppose we have a linear MPC problem as in Equation (31), and we wish to track a trajectory γ_k^{ref}. The objective function is then modified to penalize the deviation between the predicted output and the reference trajectory. A general objective function in such cases is given by Equation (47).

$$f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2}\|\gamma_{k+1} - \gamma_{k+1}^{ref}\|_Q^2 + \frac{1}{2}\Delta u_k^T R_\Delta u_k \tag{47}$$

Here, Δu_k is defined as in Equation (43), and γ_k is usually called a controlled variable [18, p. 50].

For reference tracking to be meaningful, it is necessary that the reference trajectory is feasible, meaning that it should be consistent with the system dynamics. A common approach to determine

feasible steady-state references for the states and inputs is to require that they satisfy the steady-state condition. In linear case, this is given by Equation (48). As shown in the equation, the reference setpoints must satisfy the respective linear system dynamics in that setpoint. This equation ensures that if the system reaches the reference setpoint, it can remain there under constant input Chapter 5.6 [25].

$$x^{ref} = Ax^{ref} + Bu^{ref} \tag{48}$$

In summary, reference tracking in MPC involves modifying the objective function to account for desired behavior and ensuring that the reference values are dynamically feasible. This approach enables MPC to effectively control systems toward both constant setpoints and time-varying trajectories.

4.3.3 Robust MPC

Robust Model Predictive Control (RMPC) extends classical MPC by explicitly accounting for model uncertainties and bounded disturbances. Unlike a standard MPC, which assumes a perfect model and no external influence, RMPC guarantees constraint satisfaction and stability even when the system is affected by additive disturbances. To formulate a typical RMPC problem an adapted formulation of the system in Equation (33) is considered. A discrete-time linear system on the form

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \omega_k, \quad \omega_k \in \mathcal{W} \subset \mathbb{R}^n \tag{49}$$

A typical RMPC formulation is based on formulating the optimization problem such that constraints are satisfied robustly over a set of possible disturbance realizations [26]. During this section we will focus on formulating the RMPC problem in the form of a general QP MPC, as the formulation in Equation (31). A typical such RMPC problem is formulated as

$$\min_{\mathbf{z}} \quad \sum_{k=0}^{N-1} \frac{1}{2}\|\mathbf{x}_{k+1}\|_Q^2 + \|\mathbf{u}_k\|_R^2 \tag{50a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \omega_k \tag{50b}$$

$$\mathbf{x}_k \in \mathcal{X} \tag{50c}$$

$$\mathbf{u}_k \in \mathcal{U} \tag{50d}$$

$$\forall \omega_k \in \mathcal{W} \tag{50e}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are the positive semidefinite weight matrices. The constraints $\mathcal{X}$ and $\mathcal{U}$ are the bounds on the states and inputs, which must be satisfied for all possible disturbances $\omega_k \in \mathcal{W}$. A direct solution of such a general RMPC problem may be computationally expensive which opens the introduction of a structured approach known as Tube-based MPC.

Tube-based MPC simplifies the robust control problem by decoupling nominal planning from disturbance rejection. Instead of solving the full robust optimization problem online, the controller splits the task in to two parts. The first part is based on planning the nominal trajectory, which is the system without disturbances. The second part is based on applying the actual control input by adding a feedback correction to the nominal input. The formulation of such a tube-based MPC is adapted from [26] and [17]. The first part, nominal part, is based on the nominal system

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\hat{\mathbf{u}}_k \tag{51a}$$

$$\text{s.t.} \quad \hat{\mathbf{x}}_k \in \mathcal{X} \ominus \mathcal{E}, \ \ \hat{\mathbf{u}}_k \in \mathcal{U} \ominus \mathbf{K}\mathcal{E} \tag{51b}$$

where the nominal states and inputs are defined as $\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k$. $\mathcal{E}$ is a robust positively invariant set for the error dynamics $\mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$. $\mathbf{K}$ is the stabilizing gain for the feedback correction. In the constraints the symbol $\ominus$ represents the Pontryagin set difference. The Pontryagin difference between two sets $\mathcal{A}, \mathcal{B}$ is the set of all points, $\mathcal{C}$, such that adding $\mathcal{B}$ to any of those points is still contained in the set $\mathcal{A}$ [27, p. 1], $\mathcal{A} \ominus \mathcal{B} = \{x \in \mathcal{A} : x + z \in \mathcal{A} \forall z \in \mathcal{B}\}$. In the case of the formulation in Equation (51), $\hat{\mathbf{x}}_k \in \mathcal{X} \ominus \mathcal{E}$ means that the nominal state must leave enough space for the actual state to deviate and still stay inside $\mathcal{X}$. $\hat{\mathbf{u}}_k \in \mathcal{U} \ominus \mathcal{X}$ ensures that the total input, after the second part of the formulation is added, stays within $\mathcal{U}$. The second part is based on adding the feedback correction control input to the control law, which result in the following control law [17]

$$\mathbf{u}_k = \hat{\mathbf{u}}_k + \mathbf{K}(\mathbf{x}_k - \hat{\mathbf{x}}_k) \tag{52}$$

which is applied to ensure that the system trajectory stays inside a tube around the nominal trajectory. With the error dynamics defined as $\mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$, its evolution can be defined as

$$\mathbf{e}_{k+1} = (\mathbf{A} - \mathbf{BK})\mathbf{e}_k + \omega_k \tag{53}$$

where the gain matrix $\mathbf{K}$ is selected to ensure that $\mathbf{A} - \mathbf{BK}$ is Schur stable [17], which means that all eigenvalues lie inside the open unit disk [28]. An appropriate design of the gain matrix $\mathbf{K}$, along with a appropriate choice of terminal cost matrix $\mathbf{Q_N}$, can ensure nominal system stability by the tube-based MPC. In the paper by Lorenzetti and Pavone they explain that the gain $\mathbf{K}$ and Riccati matrix $\mathbf{P}$ can be used for such purpose [17]. These are obtained by the Riccati equation in Equation (41).

4.3.4 Nonlinear MPC

Compared to a linear MPC in Section 4.3.1, a nonlinear MPC solves a similar quadratic dynamic optimization problem, however with nonlinear constraints. The linear MPC formulation in Equation (43) is reformulated into a similar QP formulation only with nonlinear constraints, as shown in Equation (54).

$$\min_{\mathbf{z} \in \mathbb{R}^n} \quad f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2} x_{k+1}^T Q_{k+1} x_{k+1} + d_{xk+1} x_{k+1} + \frac{1}{2} u_k^T R_k u_k + d_{uk} u_k + \frac{1}{2} \Delta u_k^T R_{\Delta k} \Delta u_k \tag{54a}$$

$$\text{s.t.} \quad x_{k+1} = g(x_k, u_k) \tag{54b}$$

$$x_0 = \text{given} \tag{54c}$$

$$x^{low} \leq x_k \leq x^{high} \tag{54d}$$

$$u^{low} \leq u_k \leq u^{high} \tag{54e}$$

$$-\Delta u^{high} \leq \Delta u_k \leq \Delta u^{high} \tag{54f}$$

$$\text{where} \quad Q_k \geq 0 \tag{54g}$$

$$R_k \geq 0 \tag{54h}$$

$$R_{\Delta k} \geq 0 \tag{54i}$$

$$\Delta u_k = u_k - u_{k-1} \tag{54j}$$

$$\mathbf{z}^T = (x_1^T, ..., x_N^T, u_0^T, ..., u_{N-1}^T) \tag{54k}$$

$$n = N(n_x + n_u) \tag{54l}$$

where $g(x_k, u_k)$ is a nonlinear function. As in the LMPC case, the dynamic nonlinear optimization problem must be transcribed into a finite-dimensional nonlinear program (NLP) in order to be numerically tractable. In other words, the problem must first be discretized and then optimized with the use of several transcription schemes. The ones explained in this section is the *Direct Single Shooting*, *Direct Multiple Shooting*, *Direct Collocation* and *Successive Linearization*.

In the Direct Single Shooting, or sequential, approach only the control input $\mathbf{u}_k$ is the decision variable in the optimization problem. The state trajectory for the system is further generated by forward integration from an initial state, described in Section 4.5. With the control input as the only decision variable the decision vector $\mathbf{z}$ is simply $(\mathbf{u_0}, ..., \mathbf{u}_{N-1})$. Some key notes for Single Shooting is that it has few degrees of freedom, even for large ODEs and only initialization for the control degrees of freedom are needed. However, due to the fact that the control input is the only decision variable the knowledge of the state trajectory cannot be used, as for reference tracking problems discussed in Section 4.3.2 Chapter 2.2 [29].

Direct Multiple Shooting, on the other hand, uses the same approach as for Single Shooting where the controls are piecewise discretized. Further, the each state $\mathbf{x}_k$ is used as a *shooting node* inside the discrete-time control horizon to solve an ODE as in Equation (55) [30]. To solve such ODEs the reader can refer to Section 4.5.

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{55}$$

The decision vector in Multiple Shooting is augmented with knot variables at the boundaries of each integration interval, and the dynamics is integrated separately on each segment. The continuity is enforced through equality constraints, which produces a large and highly sparse NLP, with extra degrees of freedom, which improves convergence and robustness [29].

In Collocation, both states and control inputs are discretized at a set of collocation nodes within each interval. The system dynamics are enforced as algebraic constraints at each of these nodes, and continuity between intervals is imposed via matching constraints. Collocation yields sparse NLPs and often converges faster, especially for stiff or highly nonlinear dynamics, at the expense of more decision variables per interval [29].

The method of Successive Linearization is based on approximating a nonlinear system by a sequence of linear time-varying (LTV) models around a reference trajectory. With the nonlinear dynamics given in Equation (55) the Jacobian matrices of f with respect to the states and control inputs are computed at each timestep. Concretely, denoting the reference state and input at timestep k as $\bar{x}_k$ and $\bar{u}_k$ the Jacobians are given in Equation (56).

$$\hat{\mathbf{A}}_k = \left.\frac{\partial f}{\partial x}\right|_{(x,u)=(\bar{x}_k,\bar{u}_k)} \qquad \hat{\mathbf{B}}_k = \left.\frac{\partial f}{\partial u}\right|_{(x,u)=(\bar{x}_k,\bar{u}_k)} \tag{56}$$

These expressions capture the local linearization, which is further discretized and formulated as the discrete-time LTV approximation given by Equation (**??**).

$$\mathbf{x}_{k+1} = \hat{\mathbf{A}}_k \mathbf{x}_k + \hat{\mathbf{B}}_k \mathbf{u}_k \tag{57}$$

At each timestep this LTV is substituted into the nonlinear MPC optimization problem which results in a QP formulation [30].

4.4 State Estimation

In this section state estimation will briefly be discussed. This thesis is focusing on the control aspect of the project, which is in collaboration with another student focusing on the estimation aspect. For a more detailed read about the estimation aspect of the project the reader can refer to Damsgaard's thesis [31]. During this section, the basic aspect of state estimation will be discussed, following by introducing the two state estimators relevant for the project, *Kalman Filter* and *Moving Horizon Estimation*. In most control applications, the true state $x(t)$ is not exactly known, meaning it is not directly measurable. The state must be inferred from inputs u and outputs y. Modern state-estimation cast this as a *Bayesian inference problem*. Bayesian Inference provides a systematic framework for estimating the states with prior knowledge, system dynamics and noisy

measurements, by recursively updating a belief over the state based on all available measurements up to time k, denoted $\mathbf{y}_{1:k}$. The material for this Section is adapted from [32], with notations altered to fit the formulations in this thesis.

A key assumption in Bayesian filtering is the Markov property, which asserts that the current state $\mathbf{x}_k$ is conditionally independent of all previous states $\mathbf{x}_0, ..., \mathbf{x}_{k-1}$ given the most recent state $\mathbf{x}_{k-1}$. This ensures that all relevant past information is encapsulated in x_{k-1}. The Bayes filter uses two probabilistic models: *Process model* and *Measurement model (likelihood)*. The process model is defined as $\mathbf{p}(\mathbf{x}_k|\mathbf{x}_{k-1})$ and describes how the state evolves. The measurement model is defined as $\mathbf{p}(\mathbf{y}_k|\mathbf{x}_k)$ and describes how observations relate to the state. With these two models, the Bayes filter performs two recursive steps: *Prediction step* and *Update step*. The prediction step is based on using the process model and the posterior from the previous step, the prior at time k is computed via the Chapman-Kolmogorov equations given in Equation (58).

$$\mathbf{p}(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \int \mathbf{p}(\mathbf{x}_k|\mathbf{x}_{k-1})\mathbf{p}(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})\, d_{k-1} \tag{58}$$

Here, the predicted distribution of $\mathbf{x}_k$ is represented before incorporating the new measurements $\mathbf{y}_k$. The other step, the update step, is performed with the new measurements $\mathbf{y}_k$ and is shown in Equation (59).

$$\mathbf{p}(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \frac{\mathbf{p}(\mathbf{y}_k|\mathbf{x}_k)\mathbf{p}(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{\mathbf{p}(\mathbf{y}_k|\mathbf{y}_{1:k-1})} \tag{59}$$

The resulting posterior $\mathbf{p}(\mathbf{x}_k|\mathbf{y}_{1:k})$ combines the prediction with the likelihood of the new measurement. Once the posterior is known, point estimates of the state can be extracted. Two common Bayesian estimators are *Maximum a Posteriori (MAP) estimator* and *Minimum Mean Square (MMSE) estimator*. The MAP estimator selects the most probable state given the data and the MMSE estimator minimizes the expected squared error. The equations for these filters are given in Equation (60).

$$\hat{\mathbf{x}_k}^{MAP} = \arg\max_{\mathbf{x}_k} \mathbf{p}(\mathbf{x}_k|\mathbf{y}_{1:k}), \quad \hat{\mathbf{x}}_k^{MMSE} = \mathbb{E}\left[\mathbf{x}_k|\mathbf{y}_{1:k}\right] = \int \mathbf{x}_k \mathbf{p}(\mathbf{x}_k|\mathbf{y}_{1:k})\, d\mathbf{x}_k \tag{60}$$

The MMSE estimate is unbiased under ideal conditions, while the MAP can be biased if the posterior is asymmetric or multimodal. The recursive Bayesian framework forms the basis of many practical filters. In the case of linear systems with Gaussian noise, the Bayes filter simplifies to the Kalman Filter, a closed-form MMSE estimator. Another estimator is the optimization-based Moving Horizon Estimation (MHE). These two are discussed briefly in the next sections, where the MHE is the most relevant for this thesis.

4.4.1 Kalman Filter

The Kalman Filter (KF) is a recursive Bayesian estimator that provides the MMSE estimate for linear systems with Gaussian noise. It consists of a prediction step and an update step, where state and covariance estimates are recursively updated based on system dynamics and measurements. The material for this section is adapted from [33] and [34] unless otherwise specified.

Lets consider a linear system model given by Equation (61).

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}, \quad \mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{61a}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \tag{61b}$$

Here, $\mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the process noise at timestep $k-1$, modeled as a zero-mean Gaussian with covariance $\mathbf{Q}$, which captures model inaccuracies or unmodeled disturbances. $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$

is the measurement noise, also modeled as zero-mean Gaussian with covariance $\mathbf{R}$ and captures sensor inaccuracies or noise in the measurements. The KF provides an optimal, in MMSE sense, and recursive solution for linear-Gaussian system with the Equations (62).

$$\text{Prediction Step:} \quad \hat{\mathbf{x}}_{k|k-1} = \mathbf{A}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}\mathbf{u}_{k-1} \tag{62a}$$
$$\mathbf{P}_{k|k-1} = \mathbf{A}\mathbf{P}_{k-1|k-1}\mathbf{A}^T + \mathbf{Q} \tag{62b}$$

$$\text{Update Step:} \quad \mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{C}^T(\mathbf{C}\mathbf{P}_{k|k-1}\mathbf{C}^T + \mathbf{R})^{-1} \tag{62c}$$
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_{k|k-1}) \tag{62d}$$
$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K_k}\mathbf{C})\mathbf{P}_{k|k-1} \tag{62e}$$

Here, the notation $\hat{\mathbf{x}}_{k|k-1}$ in Equation (62a) represents the predicted state estimate at timestep k based on the state at time $k-1$, which in other words means the state is predicted to be without incorporating the current measurements. $\mathbf{P}_{k|k-1}$ represents the predicted error covariance matrix at timestep k, which is the uncertainty in our prediction. The $\mathbf{K}_k$ in Equation (59) represents the Kalman gain, which determines how much the measurements should influence the new estimate. A high gain means the measurements is trused more, and a low gain means the model is trusted more. $\hat{\mathbf{x}}_{k|k}$ represents the updated state estimate at k after incorporating the measurement $\mathbf{y}_k$. $\mathbf{P}_{k|k}$ is the updated error covariance matrix at time k, which represents how uncertain we are in the corrected estimate. $\mathbf{I}$ is the identity matrix, used to adjust the error covariance during the update.

4.4.2 Moving Horizon Estimation

Moving Horizon Estimation (MHE) is an optimization-based technique for state estimation that is well-suited for nonlinear systems, and/or systems with constraints. It solves a constrained optimization problem, as for MPC, over a moving horizon. This is the estimation method used during this thesis, while KF is discussed because it was used to compare the MHE estimates to KF estimates. MHE will only be briefly discussed during this section, as Damsgaard's thesis [31] focuses on MHE and KF and will go in detail of the two, and state estimation in general. The material used for this section is adapted from Chapter 4.3 [21] unless otherwise specified. The formulation of the MHE problems is adapted to fit the MPC formulation during this thesis.

MHE is an optimization based state estimation method that utilizes a finite window of past measurements and system dynamics to estimate the current state. It is particularly advantageous for systems with constraints or nonlinearities. The formulation used in this thesis is formulated, like the MPC, as a linear QP problem. Both this formulation, and the nonlinear formulation is presented below. For the linear case we consider the linear system in Equation (63).

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \tag{63a}$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{v}_k \tag{63b}$$

With this linear system model, and the two noise variables $\mathbf{w}_k, \mathbf{v}_k$ defined as in Equation (61), the QP formulation is given in Equation (64).

$$\min_{\mathbf{x}_{k-N+1|k}} \|\mathbf{x}_{k-N+1} - \hat{\mathbf{x}}_{k-N+1}\|^2_{\mathbf{P}^{-1}} + \sum_{i=k-N+1}^{k} \left(\|\mathbf{y}_i - \mathbf{C}\mathbf{x}_i\|^2_{\mathbf{R}} + \|\mathbf{x}_{i+1} - \mathbf{A}\mathbf{x}_i - \mathbf{B}\mathbf{u}_i\|^2_{\mathbf{Q}^{-1}} \right) \tag{64}$$

Here, $\hat{\mathbf{x}}_{k-N+1}$ is the prior estimate of the state at the beginning of the horizon. $\mathbf{P}, \mathbf{Q}$ and $\mathbf{R}$ are weighting matrices, or inverse covariance matrices, for arrival cost, process noise and measurement

noise, respectively. This is a convex QP problem and can be solved efficiently using standard QP solvers. This apprach is especially practical for embedded applications and real-time estimation due to the predictable structure and speed of the optimization. The nonlinear MHE formulation in similar style as the linear formulation is shown in Equation (66). For the nonlinear MHE formulation we consider a standard discrete nonlinear system given by Equation (65).

$$\mathbf{x}_{k+1} = f(\mathbf{x_k}, \mathbf{u}_k) + \mathbf{w}_k \tag{65a}$$
$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{65b}$$

Here, the noise variables are defined as in Equation (63). At each timestep, the MHE estimates the current state by solving the following optimization problem over a moving horizon N.

$$\min_{\mathbf{x}_{k-N+1}} \|\mathbf{x}_{k-N+1} - \hat{\mathbf{x}}_{k-N+1}\|^2_{\mathbf{P}^{-1}} + \sum_{i=k-N+1}^{k} \left(\|\mathbf{y}_i - h(\mathbf{x}_i\|^2_{\mathbf{R}^{-1}} + \|\mathbf{x}_{i+1} - f(\mathbf{x}_i, \mathbf{u}_i\|^2_{\mathbf{Q}^{-1}} \right) \tag{66}$$

Constraints on state and input can, as in MPC, be included directly in the optimization problem. This will not be discussed during this section, because the concept is quite similar as to MPC which was discussed in Section 4.3.

4.5 Numerical Simulation

The task of numerical simulation is to solve initial-value problems, which is characterized by an initial state and a differential equation. Numerical simulation is based on finding an approximate of the solution of the initial-value problem. The focus during this thesis is numerical simulation of ordinary differential equations (ODE), which is also known as *Numerical Integration*. Most of the material for this section is based on Chapter 8.2 [21]. A general initial-value problem is given by Equation (67)

$$\mathbf{x}(0) = \mathbf{x}_0, \quad \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), t), \quad \forall t \in [0, T] \tag{67}$$

where T is the simulation time. A unique solution to the initial-value problem is guaranteed if the function f is continuous with respect to the time t, and *Lipschitz* continuous with respect to state x. The Lipschitz continuity requires the existence of a constant $L > 0$ such that the inequality in Equation (68) holds.

$$|f(x, t) - f(y, t)| \leq L|x - y| \tag{68}$$

During numerical integration the simulation time T is divided into smaller equal intervals, $\Delta t = \frac{T}{N}$. These intervals can be referred to as stepsize or timestep. As stated earlier, numerical integration is based on producing an approximation, $\tilde{x}(t)$, of the exact solution, $x(t)$, at each timestep by the use of several different methods. Two really common methods are *Euler integrator* and *Runge-Kutta 4*. Both these numerical integration methods are one-step integration methods. All one-step integration methods create a discrete-time system as in Equation (69)

$$\tilde{\mathbf{x}}(t + \Delta t) = \tilde{\mathbf{x}}(t) + \mathbf{\Phi}(t, \tilde{\mathbf{x}}(t), \Delta t) \tag{69}$$

where the Euler integrator, or forward Euler, approzimates the integral by $\mathbf{\Phi}(t, \mathbf{x}, \Delta t) = \Delta t f(t, \mathbf{x}(t))$. The Runge-Kutta 4 method (RK4) used four functions to map $\mathbf{\Phi}$ which makes it a fourth order method. RK4 approximates the integral by the solution of the equations in Equation (70)

$$k_1 = f\left(t, \mathbf{x}\right) \tag{70a}$$
$$k_2 = f\left(t + \frac{\Delta t}{2}, \mathbf{x} + \frac{\Delta t}{2} k_1\right) \tag{70b}$$
$$k_3 = f\left(t + \frac{\Delta t}{2}, \mathbf{x} + \frac{\Delta t}{2} k_2\right) \tag{70c}$$
$$k_4 = f\left(t + \Delta t, \mathbf{x} + \Delta t k_3\right) \tag{70d}$$
$$\mathbf{\Phi} = \frac{\Delta t}{6} k_1 + \frac{\Delta t}{3} k_2 + \frac{\Delta t}{3} k_3 + \frac{\Delta t}{6} k_4 \tag{70e}$$

Real physical systems are typically modeled in continuous time using differential equations, yet practical implementations, such as MPC or MHE, operate in discrete steps, processing sampled measurements and running discrete-time numerical routines. To bridge this gap, we discretize the continuous dynamics. We use the notations $\mathbf{A}_c, \mathbf{B}_c$ for continuous dynamics, and $\mathbf{A}_d, \mathbf{B}_d$ for discrete dynamics. With the continuous-time system given by Equation (71), a method for discretization is the *exact discretization* method [35] given by Equation (72).

$$\dot{\mathbf{x}}(t) = \mathbf{A}_c \mathbf{x}(t) + \mathbf{B}_c \mathbf{u}(t) \tag{71}$$

$$\mathbf{A}_d = e^{\mathbf{A}_c \Delta t}, \quad \mathbf{B}_d = \left(\int_0^{\Delta t} e^{\mathbf{A}_c \tau} d\tau\right) \mathbf{B}_c, \quad \mathbf{C}_d = \mathbf{C}, \quad \mathbf{D_d} = \mathbf{D} \tag{72}$$

If $\mathbf{A}_c$ is nonsingular, the formulation for $\mathbf{B}_d$ can be rewritten as $\mathbf{B}_d = \mathbf{A}_c^{-1}(\mathbf{A}_d - \mathbf{I})\mathbf{B}_c$. In practise, one often resorts to simpler approximations, like forward Euler, by replacing the exact discretization formulation in Equation (72) by $\mathbf{A}_d \approx \mathbf{I} + \mathbf{A}_c \Delta t$, and $\mathbf{B}_d \approx \mathbf{B}_c \Delta t$. Then, the continuous-time system in Equation (71) is given by the discrete-time system in Equation (73).

$$\mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d \mathbf{u}_k \tag{73a}$$
$$\mathbf{y}_k = \mathbf{C}_d \mathbf{x}_k + \mathbf{D}_d \mathbf{u}_k \tag{73b}$$

However, such approximations may fail to preserve stability for the discrete-time dynamics, which is an important property when dealing with discretization. For a continuous-time system to be asymptotically stable all eigenvalues of $\mathbf{A}_c$ must have negative real parts. For its discretized equivalent is requires all eigenvalues of $\mathbf{A}_c$ to lie within the unit circle [36, p. 45]. The approximations, by for example forward Euler, may fail to keep the asymptotical stability property for $\mathbf{A}_d$ if the discretization step is too large. The exact discretization method in Equation (72) guarantees the stability property in continuous-time carries over to discrete-time for any Δt [36, p. 53].

Another important note is that of stiff systems. When the Jacobian $\mathbf{A} = \frac{\partial f}{\partial x}$ has eigenvalues with very large negative real parts, or widely varying magnitudes, a system is considered to be stiff which may pose challenges [21, p. 506]. Their rapid modes can cause explicit, fixed-step sovlers to become unstable or inefficient, and can impair controllers (like MPC) and estimators (like MHE) when some states changes much faster than others. In such cases, adaptive implicit integrators, like MATLAB's ode15s[accessed May 2025], and careful time-step selection are often necessary to ensure accurate, robust, performance.

4.6 Practical Tools

During this thesis some practical tools were used for the implementation of the full control system. These tools will be briefly discussed during this section.

4.6.1 Finite Difference Linearization

Finite difference linearization is a common tool, which is used to approximate the derivatives of given functions. In the context of this thesis, it is used to approximate the derivatives of the nonlinear differential equations to retrieve the linearized system matrices. Given a function f, the derivative is defined as Equation (74).

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{74}$$

Given a small but finite choice of h, the left hand side of Equation (74) can be used to approximate the derivative. In Equation (75), three different methods for approximating the derivative is shown. From these three approximations the central difference is typically more accurate, as it combines the forward and backward difference [37].

$$f'(x) \approx \begin{cases} \frac{f(x+h)-f(x)}{h}, & \text{Forward difference,} \\ \frac{f(x)-f(x-h)}{h}, & \text{Backward difference,} \\ \frac{f(x+h)-f(x-h)}{2h}, & \text{Central difference} \end{cases} \tag{75}$$

4.6.2 Root Mean Squared Error

During this thesis, to obtain the optimal choice for prediction horizon N for the MPC, a quick simulation test were conducted, explained in Section 5.4.6. Root Mean Squared Error (RMSE) is a widely used metric for evaluating the accuracy of model predictions. Given a set of n observed values y_i and corresponding predicted values $\hat{y}_i$, the RMSE is defined as in Equation (76) [38].

$$\texttt{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{76}$$

In the context of this thesis, specifically MPC, this can be used to calculate the how well the MPC solution track a given reference for example.

5 Methodology

We now move on to the Methodology section, where the though process throughout the MPC implementations will be presented. During this section the whole layout for the control system is explained, which includes the maglev model in detail, system setup, tuning, control logic, and all associated code.

Throughout the entire project MATLAB has been used as the coding environment for the complete control system. MATLAB is a useful tool in the engineering world due to its native support for performing linear algebra.

5.1 System Model

The paper *Model and Control of a Magnetic Levitation Platform* is written by co-supervisor Hans Alvar Engmark and Kiet Tuan Hoang, and is an in depth derivation of the system model equation. We will not go in depth on the formulation of the system model used during this thesis, and for a deeper read the reader can refer to the said paper [3]. The magnetic field at the base was modeled using the Biot-Savart's law. Further, by the use of this magnetic field the force and torque on the levitating magnet was derived by the use of Lorentz force law. Lastly, the translation and rotation were modeled using the Newton-Euler Equations of Motion. A complete model presentation, containing the stated equations, will now be presented.

The state and input vectors of the full 3D model consist the position of the levitating magnet's center of gravity given in Cartesian coordinates, the corresponding Euler rotation angles, and their respective translation and velocities. The control input vector contains all the solenoid currents, I_i. These vectors are given by

$$\mathbf{x} = [x, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}], \quad \mathbf{u} \in \mathbb{R}^{n_s \times 1} \tag{77}$$

The equations for the force and torque are given by the two integrals

$$\mathbf{F} = \int_{\mathbf{S}} \mathbf{K}(\alpha, \beta, \gamma) \times \mathbf{B}_{base}(x, y, z)\, d\mathbf{S} \tag{78a}$$

$$\tau = \int_{\mathbf{S}} \frac{1}{\mu_0} \mathbf{J}(\alpha, \beta, \gamma) \times \mathbf{B}_{base}(x, y, z)\, d\mathbf{S} \tag{78b}$$

where $\mathbf{K}$ is the magnetization surface current, $\mathbf{J}$ is the magnetic polarization and μ_0 is the permeability of free space. The translation and rotational dynamics of the model is given by

$$\begin{bmatrix} \mathbf{F} - \mathbf{F}_g \\ \tau \end{bmatrix} = \begin{bmatrix} m\mathbf{I}_3 & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathcal{I} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \alpha \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3\times3} \\ \omega \times \mathcal{I}\omega \end{bmatrix} \tag{79a}$$

$$\text{where} \quad \mathcal{I} = \text{diag}(\left[\tfrac{1}{4}mR^2, \tfrac{1}{4}mR^2, \tfrac{1}{2}mR^2\right]) \tag{79b}$$

$$\mathbf{F}_g = -mg\hat{\mathbf{e}}_z \tag{79c}$$

where $\mathbf{a} = [\ddot{x}, \ddot{y}, \ddot{z}]$ and $\alpha = \dot{\omega} = [\ddot{\alpha}, \ddot{\beta}, \ddot{\gamma}]$ are the linear and angular acceleration, respectively. In Equations (79a) - (79c) $\mathcal{I}$ denote the inertia matrix of a solid cylinder and $\mathbf{F}_g$ denote the force of gravity, where R is the cylinder radius and g is the acceleration due to gravity.

Further, these dynamics equations can be combined into a complete model representation given as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}f(\mathbf{x}, \mathbf{u}) \tag{80a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} \tag{80b}$$

$$\text{where} \quad \mathbf{A} = \begin{bmatrix} \mathbf{0}_{6\times6} & \mathbf{I}_{6\times6} \\ \mathbf{0}_{6\times6} & \mathbf{0}_{6\times6} \end{bmatrix} \tag{80c}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{6\times6} \\ \mathbf{I}_{6\times6} \end{bmatrix} \tag{80d}$$

$$\text{and} \quad f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} m\mathbf{I}_3 & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathcal{I} \end{bmatrix}^{-1} \left(\begin{bmatrix} \mathbf{F}(\mathbf{x}, \mathbf{u}) \\ \tau(\mathbf{x}, \mathbf{u}) \end{bmatrix} - \begin{bmatrix} \mathbf{F}_g \\ \omega \times \mathcal{I}\omega \end{bmatrix} \right) \tag{80e}$$

As stated earlier, the model itself is not that relevant for this thesis. However, some background of how the model were applied for the MPC is in order. Inside the code, the system model is given by a set of nonlinear functions given as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \text{params}) \tag{81a}$$

$$\mathbf{y} = h(\mathbf{x}, \text{params}) \tag{81b}$$

where f is the function for the model dynamics, h is the function the model measurements, and the variable `params` is a struct variable containing all physical parameters for the system. From the state vector in Equation (77) we know we have twelve states, six states and their respective velocity. Due to observability and controllability considerations a reduced-order model were utilized instead of the full model. This reduced-order model removes the angle γ and it's velocity $\dot{\gamma}$, which results in a ten state vector instead.

For simplicity considering the implementations, we assumed the system to be an LTI system close to the equilibrium, which is the operating point at which we want the stable levitation. The exact equilibrium was found by the root of the function f, in Equation (81a), when the control inputs and all states, except the height z, were equal to zero. In other words, this gave the height of the levitating magnet for which the repulsion of the magnets cancels out the force of gravity. The model functions, in Equation (81a) - (81b), were further linearized in this equilibrium to get the linearized model of the system. This LTI system consists of the state matrices $\mathbf{A}_c$, $\mathbf{B}_c$ and $\mathbf{C}_c$ obtained from linearization. There is no control matrix $\mathbf{D}$ from the measurement function h since the function doesn't depend on the control inputs $\mathbf{u}$.

As stated previously in this section, the `params` variable consist of all physical parameters for the system. A table listing the key physical parameters in `params` is shown in Table 1.

Table 1: Key Physical Parameters in `params`

Parameter	**Value**	**Description**
`g`	$9.81\ \mathrm{m/s^2}$	Acceleration due to gravity
`mu_0`	$4\pi \cdot 10^{-7}\ \mathrm{H/m}$	Permeability of free space, (μ_0)
`M_lev`	$0.075\ \mathrm{kg}$	Mass of levitating magnet
`I_lev`	(82a)	Moment of inertia
`m_lev`	$3.6\ \mathrm{A\ m^2}$	Magnetic dipole moment of levitating magnet
`m_base`	$2.0\ \mathrm{A\ m^2}$	Magnetic dipole moment of the base magnets
`r_base`	Radial distance: $0.21\ \mathrm{m}$	Circular positions of base magnets
`r_sensor_array`	(82b)	Sensors positions

$$\texttt{I_lev} = 10^{-5} \begin{bmatrix} 0.709 & 0 \\ 0 & 0.709 \end{bmatrix} \text{kg m}^2 \tag{82a}$$

$$\texttt{r_sensor_array} = \begin{bmatrix} 0 & -0.032 & 0.015 \\ 0 & 0.014 & 0.032 \\ 0 & 0 & 0 \end{bmatrix} \texttt{m} \tag{82b}$$

5.2 MPC problem formulation

The formulation of both the MPC and MHE components was initially inspired by an example provided by Prof. Mohamed W. Mehrez [24], where both are implemented using nonlinear optimization with CasADi. CasADi is an open-source toolbox for algorithmic differentiation and nonlinear optimization which is well suited for SQP approaches for example [39].

However, due to the focus on levitating the manget at a stable equilibrium point and the desire for a more transparant and computationally efficient implementation, a QP formulation of the MPC and MHE was chosen. The QP formulation allows for a straightforward debugging and more predictable solver behavior compared to the complexity of general SQP formulations.

For this thesis a linear MPC was chosen, a quadratic dynamic optimization problem with linear constraints, explained in Section 4.3.1. The dynamic optimization problem for the MPC is defined as in Equation (31), however it is formulated on matrix form. The MPC problem was therefore defined by the objective function

$$f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2} \mathbf{z}^T \mathbf{G} \mathbf{z} \tag{83}$$

where $\mathbf{z} = (\mathbf{x}_0, ..., \mathbf{x}_N, \mathbf{u}_0, ..., \mathbf{u}_{N-1})$. The equality constraint and bounds must also be formulated in the same manner with respect to the decision variable $\mathbf{z}$ as

$$\mathbf{A}_{eq}\mathbf{z} = \mathbf{b}_{eq} \tag{84a}$$

$$\mathbf{z}^{low} \leq \mathbf{z} \leq \mathbf{z}^{high} \tag{84b}$$

With the reduced-order model, derived in 5.1, we have ten states and four control inputs. Throughout the project, the number of states, control inputs and measurements were defined as `nStates`, `nControls` and `nMeasurements`, respectively. These names will therefore be used when representing the dimensions of weight matrices, decision variable, etc. from now on.

To construct the objective function on matrix form as in Equation (83), the matrix $\mathbf{G}$ were constructed as a *block diagonal* matrix with the blocks containing the following terms:

- $(N+1) \cdot \mathbf{Q}$, where $\mathbf{Q} \in \texttt{nStates} \times \texttt{nStates}$
- $N \cdot \mathbf{R}$, where $\mathbf{R} \in \texttt{nControls} \times \texttt{nControls}$

The block diagonal matrix $\mathbf{G}$ was therefore constructed as

$$\mathbf{G} = \begin{pmatrix} \underbrace{\begin{pmatrix} \mathbf{Q} & & & \\ & \mathbf{Q} & & \\ & & \ddots & \\ & & & \mathbf{Q} \end{pmatrix}}_{N+1} & \\ & \underbrace{\begin{pmatrix} \mathbf{R} & & & \\ & \mathbf{R} & & \\ & & \ddots & \\ & & & \mathbf{R} \end{pmatrix}}_{N} \end{pmatrix} \tag{85}$$

To get the complete MPC problem on the matrix form the equality constraint in Equations (84a) - (84b) the $\mathbf{A}_{eq}$ matrix and $\mathbf{b}_{eq}$ vector had to be constructed to match the form of the decision variable $\mathbf{z}$. The $\mathbf{A}_{eq}$ matrix and $\mathbf{b}_{eq}$ were therefore constructed as in Equation (86). $\hat{\mathbf{x}}_k$ represents the current estimate from the MHE, if run in full-closed loop. If the MPC was run without estimates this would be the current optimal state.

$$\mathbf{A}_{eq} = \left[\begin{array}{ccccc|ccccc} I & 0 & \cdots & \cdots & 0 & -B & 0 & \cdots & \cdots & 0 \\ -A & I & \ddots & & \vdots & 0 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -A & I & 0 & \cdots & \cdots & 0 & -B \end{array}\right], \quad b_{eq} = \begin{bmatrix} \hat{\mathbf{x}}_k \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{86}$$

The bounds on the decision variables was formulated as

$$\begin{bmatrix} \mathbf{x}^{low} \\ \vdots \\ \mathbf{x}^{low} \\ \mathbf{u}^{low} \\ \vdots \\ \mathbf{u}^{low} \end{bmatrix} \leq \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \\ \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix} \leq \begin{bmatrix} \mathbf{x}^{low} \\ \vdots \\ \mathbf{x}^{high} \\ \mathbf{u}^{high} \\ \vdots \\ \mathbf{u}^{high} \end{bmatrix} \tag{87}$$

The MPC is constructed as described in Algorithm 2 as a output feedback MPC, which is explained in Section 4.3. In other words, the MPC uses the estimates of the states from the MHE at each iteration and minimizes the objective function in Equation (83).

5.2.1 Reference Tracking Formulation

The concept of reference tracking for the MPC is explained in Section 4.3.2, which could be either a non-zero constant setpoint or a trajectory for which the MPC should track. The objective function in Equation (46) was formulated on matrix form to match the decision variable $\mathbf{z}$, as in Equation (83). The objective function for the reference tracking problem, when the norm in Equation (46) is calculated, gives the following form

$$f(\mathbf{z}) = \sum_{k=0}^{N-1} \frac{1}{2}\mathbf{z}^T\mathbf{G}\mathbf{z} - 2\mathbf{z}_{ref}^T\mathbf{G}\mathbf{z} + \underbrace{\mathbf{z}_{ref}^T\mathbf{G}\mathbf{z}_{ref}}_{\text{Constant term}} \tag{88}$$

where the last term is constant and doesn't affect the optimization problem. The term $-2\mathbf{z}_{ref}^T\mathbf{G}$ was replaced by a vector $\mathbf{f}$, which was constructed as

$$\mathbf{f} = -2 \begin{bmatrix} \mathbf{x}^{ref} \\ \vdots \\ \mathbf{x}^{\mathbf{ref}} \\ \mathbf{u}^{ref} \\ \vdots \\ \mathbf{u}^{ref} \end{bmatrix} \mathbf{G} \tag{89}$$

with $\mathbf{G}$ given as in Equation (85).

5.2.2 Robust MPC Formulation

Throughout the project, an implementation of a more robust MPC method was decided. More specifically a tube-based RMPC, which is explained in Section 4.3.3. The goal for such an implementation was to make the MPC more robust for sudden disturbances or model mismatches, and compare the affect. This section will cover the problem formulation which focuses only on the tube-based RMPC implementation part.

As Section 4.3.3 explains, the only key part about implementing a tube-based RMPC is to divide the system formulation into two parts. The first part, nominal part, stays the same as in Section 5.2. The second part is the new implementation where a feedback correction is added to the full control input. The full control input updated at each timestep is given by Equation (90).

$$\mathbf{u}_k = \bar{\mathbf{u}}_k + \mathbf{K}(\hat{\mathbf{x}}_k - \bar{\mathbf{x}}_k) \tag{90}$$

where $\bar{\mathbf{u}}_k$ is the nominal control sequence calculated by the MPC, $\bar{\mathbf{x}}_k$ is the current real state from the integration step, and $\hat{\mathbf{x}}_k$ is the MHE's estimate of the current state. As explained in Section 4.3.3, a good choice for the gain matrix $\mathbf{K}$ is simply the LQR gain matrix. The LQR gain is calculated by the Riccati equation (41), which is done in MATLAB by the function `dlqr`. From the Riccati equation we also obtain the Riccati matrix $\mathbf{P}$, which is a good choice for the terminal weight matrix $\mathbf{Q}_N$. The gain matrix K is not the same used as for the LQR in the simulation, where in the RMPC case we calculated the gain with the MPC weight matrices $\mathbf{Q}$ and $\mathbf{R}$. One important note for the RMPC case was that the tube width in the tube-based RMPC, which represents the maximum deviation between the nominal and real trajectories, was not explicitly computed. Instead, we assumed that the feedback gain $\mathbf{K}$ is sufficient to keep the trajectory within a bounded region around the nominal one, which would implicitly handle the tube-width problem. As a result, no formal constraint tightening or robust invariant set computation is applied, which simplifies the implementation but assumes that disturbances and estimation errors remain small enough to avoid constraint violations.

One important note considering the RMPC formulation, compared to the tube-based RMPC in Section 4.3.3, is that we do not have tube widths included in the formulation. In otherr words, the invariant set $\mathcal{E}$ from Equation (51b) was not included in this thesis. We rather used the MHE for the disturbance estimation instead, which means that the RMPC formulation is not a classical tube-based RMPC as in the articles mentioned earlier. The RMPC formulation is based on these formulations, with some alterations because of time-limitations.

5.2.3 Slack Variable Formulation

When experimenting with the MPC we wanted to make the bounds on the control input as strict as possible and still obtain stability for the magnet. Therefore, we decided to add slack variables for the control inputs, which is explained in Section 4.3.1. The problem formulation when adding slack variables is based on the formulation in Equation (44), but the difference being that the slack variables only is applied to the control inputs. The goal by adding slack variables is to see how

strict the bounds can be set, still obtaining stability, and letting the system violate these bounds by some degree by adding these variables.

The new MPC formulation, when adding the slack variables, is given by Equation (91).

$$\min_{\mathbf{z}} \quad \sum_{k=0}^{N-1} \frac{1}{2}\mathbf{x_{k+1}}^T\mathbf{Q}\mathbf{x}_{k+1} + \frac{1}{2}\mathbf{u}_k\mathbf{R}\mathbf{u}_k + \rho^T\epsilon + \frac{1}{2}\epsilon^T\mathbf{S}\epsilon \tag{91a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{91b}$$

$$\mathbf{x}^{low} \leq \mathbf{x}_k \leq \mathbf{x}^{high} \tag{91c}$$

$$\mathbf{u}^{low} - \epsilon \leq \mathbf{u}_k \leq \mathbf{u}^{high} + \epsilon \tag{91d}$$

where ϵ, ρ and $\mathbf{S}$ are defined as in Equation (44). Similar to the standard MPC formulation in Section 5.2, we wanted to formulate this problem on matrix form. The same formulation is used, where ϵ is added as a decision variable in the vector $\mathbf{z}$. In Equations (92a) - (92d) the new formulation is shown.

$$\min_{\mathbf{z}} \quad \sum_{k=0}^{N-1} \frac{1}{2}\mathbf{z}^T\mathbf{G}\mathbf{z} \tag{92a}$$

$$\text{s.t.} \quad \mathbf{A}_{eq}\mathbf{z} = \mathbf{b}_{eq} \tag{92b}$$

$$\mathbf{G}_{ineq}\mathbf{z} \leq \mathbf{h}_{ineq} \tag{92c}$$

$$\mathbf{z}^{low} \leq \mathbf{z}_k \leq \mathbf{z}^{high} \tag{92d}$$

where $\mathbf{z}$ is the decision vector containing all states, control inputs and slack variables $(\epsilon_0, ..., \epsilon_{N-1})$. The equality matrix $\mathbf{A}_{eq}$ is the same as in Equation (**??**), but a third block is added with only zeros to match the number of decision variables. The bounds on the decision vector containts all bounds which is defined as

$$\mathbf{z}^{low} = \begin{bmatrix} \mathbf{x}^{low} \\ \vdots \\ \mathbf{x}^{low} \\ \mathbf{u}^{low} \\ \vdots \\ \mathbf{u}^{low} \\ \epsilon^{low} \\ \vdots \\ \epsilon^{low} \end{bmatrix}, \quad \mathbf{z}^{high} = \begin{bmatrix} \mathbf{x}^{high} \\ \vdots \\ \mathbf{x}^{high} \\ \mathbf{u}^{high} \\ \vdots \\ \mathbf{u}^{high} \\ \epsilon^{high} \\ \vdots \\ \epsilon^{high} \end{bmatrix} \tag{93}$$

where the bounds on states and inputs are defined as normal, with bounds on ϵ being $0 \leq \epsilon \leq \infty$. The new $\mathbf{G}$ matrix is defined as earlier with another block diagonal matrix containing $\mathbf{S}$ matching the number of ϵ. $\mathbf{S}$ is a diagonal weight matrix, as $\mathbf{Q}$ and $\mathbf{R}$, penalizing the slack variable ϵ. ρ for the linear term in Equation (91a) is the weight on the linear term of ϵ, which is defined as a scalar.

The inequality constraint in Equation (92c) contains the bounds on $\mathbf{u}_k$ and ϵ to set up the bounds as in Equation (91) containing the slack variables. The inequality matrix $\mathbf{G}_{ineq}$ and inequality vector $\mathbf{h}_{ineq}$ was defined as

$$\mathbf{G}_{ineq} = \begin{bmatrix} \mathbf{0} & \mathbf{I} & -\mathbf{I} \\ \mathbf{0} & -\mathbf{I} & -\mathbf{I} \end{bmatrix}, \quad \mathbf{h}_{ineq} = \begin{bmatrix} \mathbf{u}^{\mathbf{high}} \\ \vdots \\ \mathbf{u}^{high} \\ -\mathbf{u}^{low} \\ \vdots \\ -\mathbf{u}^{low} \end{bmatrix} \tag{94}$$

where the dimension of $\mathbf{0}$ matches the number of states in the objective function $(\mathtt{N}+1)\cdot n_x$, and $\mathbf{I}$ matches the number of control inputs and slack variables $\mathtt{N}\cdot n_u$. When we calculate the inequality constraint in Equation (92c), with the formulation in Equation (94), we get the formulation in Equation (91).

5.3 Closed-loop Setup

The ultimate goal was to achieve on-line closed-loop control of the maglev system with the use of a combination of MPC, LQR and MHE. The two controllers, MPC and LQR, would get state estimates from the MHE. A block diagram illustrating the full control loop is shown on Figure 5. The controllers, MPC or LQR, receive an estimate of the current state, $\hat{\mathbf{x}}_k$, and compute a control action which is applied to the system which moves one step. The sensors obtain the value of the new states and transfer a measurement $\mathbf{y}_k$ of the current states to the MHE. The MHE receive both the measurements from the sensors, and the control input from the controllers, which is used to make an estimate of the states.

The block *System* is either the simulated model or the real physical system depending on the context. In the real physical system the sensors obtain measurement based on the position of the levitating magnet, but in simulation these measurements are calculated by the measurement model h in Equation (81b).

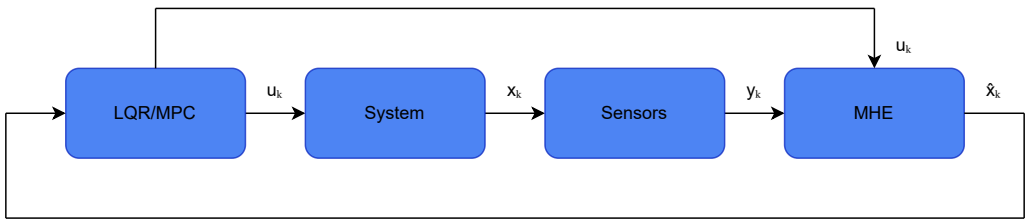


Figure 5: Block diagram of the closed-loop control system. *System* is either the simulated model or the real physical maglev system.

For the LQR in this context the control setup is constant by only applying the constant gain matrix $\mathbf{K}$ to the current state estimate $\hat{\mathbf{x}}_k$. In the MPC case there are some parameters which changes every iteration. When the tube-based RMPC, explained in Section 4.3.3, is applied the constant gain matrix $\mathbf{K}$ is applied as in Equation (90). Inside the MPC problem the $\mathbf{b_e q}$ is constantly updated each timestep, where the first element is switched out with the current estimate $\hat{\mathbf{x}}_k$. Based on what type of problem we want to solve, the rest is pretty much constant. As for the reference tracking problem, explained in Section 5.2.1, it depended on what type of reference the MPC was going to track. If the reference point is a constant setpoint, the vector $\mathbf{f}$ containing the reference would be constant. However, if the MPC were to track a reference trajectory, this would change at each timestep depending on the trajectory state at timestep k.

5.4 Code

The whole code consists of multiple files which collaborate into one large control system. All files related to the Maglev system were provided from co-supervisor Engmark, which is the file `3D model reduced order_fixed`, which included all necessary functions and parameters. During the project the model continuously developed, where co-supervisor Engmark continuously updated the model directory with model fixes, provided us the linearized model directly, etc. Below the main directory for the control system is listed, where only the smaller files related to MPC is included. All files related to MHE will be discussed in Damsgaard's thesis [31]. The code implemented for the full control system can be found on our GitHub repository where all files involved in the project is shared.

```
/
├── 3D model reduced order_fixed
│   ├── parameters.m
│   └── functions
│       ├── f.m
│       ├── h.m
│       └── utils
│           ├── rotRateEuler.m
│           ├── rotEuler.m
│           ├── linearizeModel.m
│           └── physics
│               ├── torque.m
│               ├── magnetization.m
│               ├── force.m
│               └── field.m
├── ABC_high_fidelity_model.mat
├── main.m
├── MHEclass.m
├── MPCclass.m
├── mpc_bounds.m
├── RK4Step.m
└── openloopMain.m
```

Both the MPC and MHE were constructed in separate files as classes to isolate all logic related to each of them in separate class files. This would make it easier to debug and expand each part of the whole code, additionally the main code with the control loop would consisting only of the common parts of MPC and MHE during the project. This gave the resulting file `main.m` consisting of the following main parts:

- System Model initialization
- MPC, MHE, LQR tuning
- Control loop
- Plots of results

This section will cover the code implementation for the separate parts related to MPC, and also the main code. For starters, we review the system model initialization and tuning in the main loop. Then, we explain the structure and methods used in the class file for the MPC, before reviewing the main control loop. Throughout the presentation of the code, we will only show the MPC setup in general. The RMPC, as shown in Section 5.2.2, only requires the single term in Equation (90) which will be seen in the code during the presentation of the main loop. Except for the correction term in the RMPC case, the RMPC has the same setup as MPC.

5.4.1 System Model Initialization

The system model initialization is implemented in the first part of `main.m`.

Listing 1: Finding the equilibrium

```
addpath(genpath('3D model reduced order'))

% Define system parameters
params = parameters;

% Find equilibrium point
index = @(A,i) A(i);
fz = @(z) index(f([0,0,z,zeros(1,7)]', [0,0,0,0]', params), 8);
zeq =  fzero(fz,0.1);

xeq = [0, 0, zeq, zeros(1,7)]';
ueq = [0,0,0,0]';
```

Firstly, we added a path to the model directory. This file includes all functions, parameters, etc. relevant for the system. The struct variable `params` was loaded from this model directory from the file `parameters.m`. When loading the directory this way, we could do changes in the parameters file which would automatically update when running `main.m`. Such changes could be the positions of the sensors, as in Table 1.

As stated in Section 5.1, we linearize the system around the equilibrium, which was found by the root of function f, in Equation (81a). In the code, the root was found by defining a anonymous function, `index`, which returns the $i-$`th` element of a given array `A`. Further, another anonymous function, `fz`(z), was defined. `fz`(z) uses `index`, given an array where all elements except from z is zero, and returns the $8-$`th` element which refers to the velocity of z, $\dot{z}$. The non-zero equilibrium `zeq` is then found by the root of $f(z)$, meaning the value of z which makes $\dot{z} = 0$. To find this root we used the MATLAB function `fzero`, with the initial guess of 0.1, which is the second argument in `fzero`. This initial guess was based on the assumption that the equilibrium was to lie in the order of magnitude. Further, the equilibrium was defined, as the two last lines of the code 5.4.1 shows, as `xeq` and `ueq`.

The equilibrium found with the code 5.4.1 was further used as the linearization point for the functions in Equation (81a)-(81b). This linearization was performed by a utility function, `linearizeModel.m` from the `3D reduced order` file. This function is shown in the code 5.4.1, where we only show part of the the code since the for loops for B and C is constructed exactly the same as for A, with only the variable names being different.

Listing 2: The linearization function

```
function [A,B,C] = linearizeModel(f,h,xlp,ulp,params)
    delta = 1e-10;
    nx = length(xlp);
    nu = params.n_u;
    ny = params.n_y;

    A = zeros(nx);
    for i = 1:nx
        xp = xlp;
        xn = xlp;

        xp(i) = xp(i) + delta;
        xn(i) = xn(i) - delta;

        dxp = f(xp, ulp, params);
        dxn = f(xn, ulp, params);

        A(:, i) = (dxp - dxn)/(2*delta);
```

```
    end

    B = zeros(nx, nu);
    for i = 1:nu
        ...
    end

    C = zeros(ny, nx);
    for i 0 1:nx
        ...
    end
end
```

The state matrices obtained from `linearizeModel.m` was defined as `Ac`, `Bc` and `C` which we used to define the dimensions `nStates`, `nControls` and `nMeasurements`. This is shown in the code 5.4.1 below.

Listing 3: System setup

```
    % Linearize model
    xlp = xeq;
    ulp = ueq;

    % States: [ x y z phi theta xdot ydot zdot phidot thetadot ]
    [Ac, Bc, C] = linearizeModel(@f, @h, xlp, ulp, params);

    nStates = size(Ac, 1);
    nControls = size(Bc, 2);
    nMeasurements = size(C, 1);
```

5.4.2 MPC Tuning

This thesis is focusing on the control part of the project, which means that the tuning of the MHE will not be review during this thesis. However, this will be discussed in detail in Damsgaard's thesis [31].

The control loop was run for `NT` iterations with $\Delta t =$ `dt`. The simulation time `dt` was experimented with a lot because we used exact discretization of the model, which makes the stability properties of the system unaffected by the choice of `dt`. A higher `dt` would result in faster dynamics, meaning that the dynamics evolve in fewer iterations. A lower `dt` however, would increase the temporal resolution. This would give a finer sampling, which is generally preferable when it comes to control performance because the controller will react to faster dynamics quickly. Also, the estimator for a given control system will get more frequent measurements. The most frequent choice of simulation time was `dt` $= 0.003$s, ± 0.002. `NT` was defined as `t/dt`, where `t` was a fixed simulation time. The goal being to stabilize the levitating magnet in the equilibrium, the choice of the initial condition `X0` was set close to the equilibrium.

The tuning part of the main code involved assigning values for all weight matrices for MPC, MHE and the LQR. Focusing primarily on the control aspect of the project, the tuning of the MPC and LQR will be reviewed. Both the MPC and LQR need the two weight matrices Q and R, which is defined as `Q_MPC`, `R_MPC`, `Q_LQR` and `R_LQR`. One important tuning parameter for the MPC is the horizon length N, defined as `N_MPC`, which was chosen based on a RMSE test presented in Section 5.4.6. The LQR doesn't require a horizon length as explained in Section 4.2.2. The weight matrix `Q_MPC` was weighted due to which state required more penalization, and how fast we wanted the system to converge. A higher value in the elements of `Q_MPC` would result in faster convergence and less oscillation. The weight matrix for the control inputs `R_MPC` was experimented with to see the affection of higher or lower values. A lower value would result in a more aggressive control input but a faster response. A higher value would result in slower movement. The bounds on the states and inputs were defined in the separate file `mpc_bounds.m`. The bounds on the states was chosen

due to the physical dimensions of the Maglev system, which is listed in Table 1, and are presented in Table 3.

When we used the slack variables there were two weights which required tuning, presented in Section 5.2.3, $\mathbf{S}$ and ρ. $\mathbf{S}$ is a weight matrix for the quadratic term of the slack variable, which was tuned based on how frequent we would allow the controller to violate the control input bounds. A higher value in $\mathbf{S}$ would result in less violations but a too high weight would cause infeasibility. If the tuning was too low the controller would violate the constraints more frequently which is not desired. The same goes for the linear term of the slack variable, ρ. There were tuned based on how strict we wanted the control bounds to be. As for the reference tracking case, with the problem formulation presented in Section 5.2.1, The weight matrix `Q_MPC` was tuned based on penalizing the deviation from the actual states and its reference point. A higher weight would result in a quicker convergence to the reference, but a too high weight could cause instability for the system.

5.4.3 MPC Class

As stated earlier, all code related to MPC and MHE individually, were defined in separate files as classes. The main reason for the use of separate class files was to separate all code related to MPC and MHE individually from the main code. A second important reason was is the model would change during the project, which it did, or if we would test on other systems, it wouldn't be too comprehensive to change the system. The class files allow the MPC and MHE code to be totally general code which takes variable inputs in the main code. The file `MPCclass.m` is the class file for the MPC and we will now review how this file was constructed.

The MPC class takes a set of input arguments, defined in `main.m`, which is listed in Table 2.

Table 2: `MPCclass.m` Input Arguments

Input	**Description**
`N_MPC`	Prediction Horizon
`Ac`	Linearized continuous-time transition matrix, $\mathbf{A}_c$
`Bc`	Linearized continuous-time control matrix, $\mathbf{B}_c$
`X0`	Initial condition
`dt`	Discretization time
`nStates`	Number of states
`nControls`	Number of control inputs
`lb`	Lower bound vector
`ub`	Higher bound vector
`Q_MPC`	State weight matrix
`R_MPC`	Control weight matrix
`xRef`	The reference setpoint
`MPC_options`	Options related to the solver `quadprog`

These input arguments are defined in the main code where the MPC class is called with the code 5.4.3.

Listing 4: The MPCclass call function

```
mpc = MPCclass(N_MPC, Ac, Bc, X0, dt, lb, ub, Q_MPC, R_MPC, nStates,
    nControls, MPC_options, xRef);
```

The class file includes several functions for which the most important parts are:

- `Initialize`
- `SetupOptimizationProblem`
- `runMPC`
- `runOpenLoop`
- `computeReferenceInput`

These functions will now be reviewed, and the most important code will be listed and discussed. Throughout the class code, the variable `obj` is referred to as the class object containing all related variables and parameters in the class file. When calling the MPC class with the code 5.4.3, the two functions `initialize` and `setupOptimizationProblem` is run at once. The `initialize` function's purpose is to calculate the exact discretization matrices of `Ac` and `Bc` with `dt`, and calculate a terminal cost matrix `QN` by the MATLAB function `dlqr`, where the matrix `P` from the Riccati equations (see Section 4.2.2) is used as the terminal cost matrix. The choice of the Riccati matrix as terminal cost was made as explained in Section 4.3.3. The `initialize` function is shown in the code 5.4.3.

Listing 5: The initialize function in the MPC class

```
    % Function for discretezise A and B matrices
        function obj = initialize(obj)
            obj.A = expm(obj.Ac * obj.dt);
            obj.B = (expm(obj.Ac * obj.dt) - eye(size(obj.Ac))) * (obj.Ac \
                 obj.Bc);

            [~,P,~] = dlqr(obj.A, obj.B, obj.Q, obj.R);
            obj.QN = P;

            obj = obj.setupOptimizationProblem();
        end
```

The next function in the MPC class is the `setupOptimizationProblem`, where the weight matrices $\mathbf{Q}$ and $\mathbf{R}$, the block diagonal matrix $\mathbf{G}$ and the vector $\mathbf{f}$ (see Section 5.2.1), as well as the $\mathbf{A}_{eq}$ matrix were constructed. The weight matrices were constructed in the main code and provided to the MPC class as seen in Listing 5.4.3. However, these matrices need to match the prediction horizon in dimension when constructing the $\mathbf{G}$ matrix. We start by taking the Kronecker product of an identity matrix, with dimension equal to the prediction horizon, and the weight matrix $\mathbf{Q}$. Since both are diagonal, this gives us a diagonal matrix with dimension equal to the prediction horizon and with the elements of $\mathbf{Q}$ on the diagonal. Finally, we use the MATLAB function `blkdiag`, which construct a block diagonal matrix with the given arguments. We use the Kronecker matrix and the terminal weight matrix `QN` as the arguments which creates a block diagonal matrix with elements of $\mathbf{Q}$ on the diagonal and the last block in bottom right corner is the elements of `QN`. The same purpose was for the $\mathbf{R}$ matrix. Further, both the `costQ` and `costR` were used as arguments for `blkdiag` resulting in the block diagonal $\mathbf{Q}$ matrix. The vector f is here set to zero, because there is no current reference tracking objective done. If we wanted to have reference tracking we just commented out the code below the f vector.

In Section 4.3.2 we discussed the optimization problem formulated as a reference tracking problem. From Equation (48) we know that if we flip the equation solving for $\mathbf{u}^{ref}$ we get the equation seen in the Listing 5.4.3. Here we calculate `uRef` by taking the pseudoinvers of `B` times $(I - A)x^{ref}$. This gives us a constant value of u^{ref} which forces the system to a given setpoint x^{ref}. Then the vector `ref` containing the reference for state and control input is constructed to match the dimension of `G`. Lastly, the vector f is defined as $-2\mathbf{G}\mathbf{ref}$. The `Aeq` matrix was constructed to match the form presented in Equation (86). In the code, this is done by a `for` loop which was used to calculate the index position in the matrix and get the correct position for $\mathbf{A}$ and $\mathbf{B}$.

Listing 6: The setupOptimizationProblem function

```
function obj = setupOptimizationProblem(obj)

    % Setting up G and f matrices
    costQ = blkdiag(kron(eye(obj.N_MPC), obj.Q), obj.QN);
    costR = kron(eye(obj.N_MPC), obj.R);
    obj.G = blkdiag(costQ, costR);
    obj.f = zeros(size(obj.G, 1), 1);
    %
    % uRef = pinv(obj.B) * ((eye(obj.nStates) - obj.A)*obj.xRef);
    %
    % Xref_full = repmat(obj.xRef, obj.N_MPC+1, 1);
    % Uref_full = repmat(uRef, obj.N_MPC, 1);
    % ref = [Xref_full; Uref_full];
    % obj.f = -2 * obj.G * ref;

    % Constructing the Aeq matricx
    rowAeq = (obj.N_MPC+1) * obj.nStates;
    colAeq = (obj.N_MPC+1) * obj.nStates + obj.N_MPC * obj.nControls;
    obj.Aeq = zeros(rowAeq, colAeq);
    obj.beq = zeros(rowAeq, 1);
    obj.beq(1:obj.nStates) = obj.X0;

     for k = 1:obj.N_MPC+1
        % Row indices for the k-th constraint (each constraint has
        % obj.nStates rows)
        % Placement of Identity
        rowIdx = (k-1)*obj.nStates + 1 : k*obj.nStates;
        colIdx_state = (k-1)*obj.nStates + 1 : k*obj.nStates;
        obj.Aeq(rowIdx, colIdx_state) = eye(obj.nStates);

        % Placement of A
        if k > 1
            colIdx_prevState = (k-2)*obj.nStates + 1 : (k-1)*obj.
                nStates;
            obj.Aeq(rowIdx, colIdx_prevState) = -obj.A;
        end

         if k > 1
            % Placement of B
            colIdx_control = (obj.N_MPC+1)*obj.nStates + (k-2)...
            *obj.nControls + 1 : (obj.N_MPC+1)*obj.nStates + ...
            (k-1)*obj.nControls;
            obj.Aeq(rowIdx, colIdx_control) = -obj.B;
         end
     end
end
```

The next function in the MPC class file is the `runMPC` function presented in Listing 5.4.3. This function was called in the main control loop in the main code, which will be discussed later, at each iteration with input argument `xCurrent`. `xCurrent` was the estimate done by the MHE of the current state, which was fed to the MPC as the initial condition. Firstly, the `beq` was constructed with the first argument as `xCurrent` (see Equation (86)). `beq` was included in this function because we wanted to update the vector at each iteration to update the equality constraint for each current state. Next, the optimization problem was solved with the MATLAB QP solver `quadprog`. The input arguments are seen in the Listing 5.4.3, where the two first is the block diagonal matrix $\mathbf{G}$ and vector $\mathbf{f}$. We multiplied the matrix $\mathbf{G}$ with two to account for the fraction in the objective function as in Equation (83). We had no inequality constraints, except for the bounds, which is why these two are empty. The rest of the arguments are the equality constraint, bounds, and the `MPC_options`. The optimal solution for the QP was stored in each iteration as `Zopt`, where we extracted the first control input sequence from this vector. This `Zopt` vector has the dimension $(\texttt{N_MPC} + 1) \cdot \texttt{nStates} + \texttt{N_MPC} \cdot \texttt{nControls}$, where the respective solutions for optimal state and control inptus were separated into two the variables `U0` and `X1`. The variable `Uopt` extracted

the first control sequence, and `xNextTrue` extracted the optimal state for the next step $\mathbf{x_{k+1}}$. `xNextTrue` was only used to compare the MHE estimates `xCurrent` to the optimal state evolution, so this is not relevant for the main loop.

The output arguments from the solver `quadprog` had each separate purposes, where the most important one was `exitflag`, which gave an output of a number, where 1 was the desired result. If the output was 1 the optimization problem had been solved successively, if any other output was printed it meant that there was some issues which needed debugging.

Listing 7: The runMPC function

```
function [xNextTrue, Uopt] = runMPC(obj,xCurrent)

    obj.beq(1:obj.nStates) = xCurrent;


    [Zopt, fval, exitflag, output, lambda] = ...
    quadprog(2*obj.G, obj.f, [],[],obj.Aeq, obj.beq,obj.lb,obj.ub, [],
        obj.options);

    U0 = Zopt((obj.N_MPC+1)*obj.nStates+1 : ...
    (obj.N_MPC+1)*obj.nStates + obj.nControls);
    Uopt = U0(1:obj.nControls);

    X1 = Zopt(1:(obj.N_MPC+1)*obj.nStates);
    xNextTrue=X1(obj.nStates+1:2*obj.nStates);

end
```

The last important function within the MPC class is the `runOpenLoop` function presented in Listening 5.4.3. Here, the open-loop solution of the dynamic optimization problem was solved once when calling the function in the main file. The purpose of this function was to get an optimal open-loop trajectory which would be compared with the closed loop solution which will be presented in Section 5.5. Also, the open-loop trajectory could further be used as a reference trajectory in the control loop. The only difference in this function, compared to `runMPC`, was that the whole optimal solution vector `Zopt` was separated into the two trajectory vectors `xTraj` and `uTraj`.

Listing 8: The runOpenLoop function

```
function [xTraj, uTraj] = runOpenLoop(obj)

    obj.beq(1:obj.nStates) = obj.A*obj.X0;

    [Zopt, fval, exitflag, output, lambda] = ....
    quadprog(2*obj.G, obj.f, [],[],obj.Aeq, obj.beq,obj.lb,obj.ub, [],
        obj.options);

    uTraj = reshape(Zopt((obj.N_MPC+1)*obj.nStates+1 : end), ...
    obj.nControls, obj.N_MPC);
    xTraj = reshape(Zopt(1:(obj.N_MPC+1)*obj.nStates), ...
    obj.nStates, obj.N_MPC+1);

end
```

5.4.4 MPC Bounds

Now, the file `mpc_bounds` will be reviewed. The code itself is pretty straight forward and will not be listed here. In the file we only list all higher and lower bounds on states and inputs and store them in two variables as vectors `lb` and `ub`, which is further used as input arguments for the MPC class as shown in Table 2. Instead of showing the code we list the values of these bounds in Table

3 below. The lower and upper bound on states x and y is set to $\pm 0.03\,\mathtt{m}$. This was decided from the parameters of the maglev model, in Table 1, where `r_base` represents the radial distance for the four magnets placed in a circular pattern around the z-axis. From that radial distance, $0.021\,\mathtt{m}$, we simply used Pythagoras which resulted in $\approx 0.03\,\mathtt{m}$ as lower and upper bound for both the x and y position.

Table 3: Bounds selected for states and inputs

State/Input	**Value**
$\mathbf{x}$	$\pm 0.03\,\mathtt{m}$
$\mathbf{y}$	$\pm 0.03\,\mathtt{m}$
$\mathbf{z}$	$\pm 0.01 + \mathtt{zeq}\,\mathtt{m}$
α	$\pm\frac{\pi}{8}\,\mathtt{rad}$
β	$\pm\frac{\pi}{8}\,\mathtt{rad}$
$\dot{\mathbf{x}}$	$\pm\infty$
$\dot{\mathbf{y}}$	$\pm\infty$
$\dot{\mathbf{z}}$	$\pm\infty$
$\dot{\alpha}$	$\pm\infty$
$\dot{\beta}$	$\pm\infty$
$\mathbf{I}_1$	$\pm 5\,\mathtt{A}$
$\mathbf{I}_2$	$\pm 5\,\mathtt{A}$
$\mathbf{I}_3$	$\pm 5\,\mathtt{A}$
$\mathbf{I}_4$	$\pm 5\,\mathtt{A}$

5.4.5 Main Control Loop

Now a review of the main control loop will be covered. During the main control loop, there are some tests done on the MHE in order to calculate the accuracy and consistency. These will be mentioned and briefly explain what they do and why they are used, for a deeper read about these tests the reader can refer to Damsgaard's thesis about MHE [31].

The main control loop governs the simulation and estimation process, combining both control and state estimation in a real-time iterative structure. The simulation was constructed inside a `while` loop where at each iteration the system applied a control action from either the LQR or MPC, depending on the system's performance as evaluated by the Normalized Innovation Squared (NIS) metric. The NIS, which quantifies the consistency between predicted and measured outputs, was compared to statistically derived bounds to access the accuracy of the MHE. If the NIS remained within these bounds consistently, the switch counter `sxitchCounter` increased, eventually enabling the use of MPC for more advanced control. Conversely, if the NIS exceeded the confidence interval, the counter was reduced and the system defaulted to the LQR to ensure robustness. This switching logic ultimately smoothed the transition between LQR and MPC, avoiding abrupt shifts. As seen in the code in Listening 5.4.5, there are two control setups where one is for the MPC and one is for the RMPC. The RMPC gain was calculated as a LQR gain with the MPC weight matrices, as explained in Section 5.2.2. In this thesis, both nominal MPC and the RMPC were tested. To toggle between them we simply removed the correction term or applied it depending on which controller to be used.

At each iteration, throughout the `while` loop, the next control action was computed and applied to the system dynamics. Inside the last `if` code we checked if the NIS was either inside or outside the bounds and, depending on the conclusion, either the LQR or MPC computed the next control input. This is where `runMPC`, from Listening 5.4.3, is called if the `useAdvancedControl` was set to true. If it was set to false the LQR computes the next control input with the constant gain `K_lqr`

calculated from the Riccati equations, see Equation **??**. With the calculated control sequence the next step for the system dynamics was calculated via the Runge-Kutta integration step function `RK4Step`, which is discussed in Section 4.5. Originally, we used the MATLAB function `ode15s` but switched to Runge-Kutta 4 which eventually gave more satisfying results. The system output, after this integration step, was measured with the use of an exponential moving average (EMA), and passed to the MHE, which updated the state estimation for the next iteration. The state estimation performance was evaluated with the use of both NIS and the Normalized Estimation Error Squared (NEES) to track consistency and accuracy over time. The `while` loop continued as long as the `iterCounter` was less, or equal, to the variable `NT`.

Listing 9: The main control loop

```

    %Calculating the RMPC gain before the loop
    [K_mpc,~,~] = dlqr(mpc.A, mpc.B, Q_MPC, R_MPC);

    % Current switching logic:
    %   If NIS is inside bounds: increase switchCounter, if Nis exits bounds: reduce switchCounter.
    %   Whenever switchCounter > switchThreshold: use advanced control, else use LQR.
    %   This hopefully leads to a smoother transition between control types.
    while RunningFlag == true && iterCounter <= (NT)
        t_start = tic;
        k = iterCounter;
        iterCounter = iterCounter + 1;


        if (NIS_current) >= lowerBound_NIS && (NIS_current <= upperBound_NIS)
            switchCounter = switchCounter + 1;
            if switchCounter > 2 * switchThreshold
                switchCounter = 2 * switchThreshold;
            end
        else
            switchCounter = switchCounter - 3;
            if switchCounter < 0
                switchCounter = 0;
            end
        end

        if switchCounter > switchThreshold
            useAdvancedControl = true;
        else
            useAdvancedControl = false;
        end

        disp(string(k) + ", Running with advanced control: " + ...
                string(useAdvancedControl))


        if iterCounter == mhe.N + 2
            mhe.Q = Qscaling * mhe.Q;
            mhe.G(mhe.nStates * (mhe.N+1) + 1 : mhe.nStates * (mhe.N+1) + ...
                    mhe.nStates * mhe.N, mhe.nStates * (mhe.N+1) + 1 : ...
                    mhe.nStates * (mhe.N+1) + mhe.nStates * mhe.N ) = ...
                    kron(eye(mhe.N), mhe.weightScaling * mhe.Q);
            % Here we increase Q after N_MHE+1 iterations when the MHE has calibrated.
            % This seems to improve performance?
        end
```

```

        %use controllerMode = X_sim(:,k) or xEst for running MHE on true state or MHE estimates
        controllerMode = X_sim(:,k); if controllerMode ==xEst; controllerModePrint="xEst"; elseif controllerMode == X_sim(:,k); controllerModePrint="Xsim";end

        % MPC
        if useAdvancedControl
            [~, Uopt] = mpc.runMPC(controllerMode);
            U = Uopt;
            controllModeVec(k) = 1;
        else
            U_LQR = -K_lqr * controllerMode;
            U = U_LQR;
        end

        % RMPC
        if useAdvancedControl
            [xTrue, Uopt] = mpc.runMPC(controllerMode);
            U = Uopt + K_mpc*(xTrue - controllerMode); %RMPC used
            controllModeVec(k) = 1;
        else
            U_LQR = -K_lqr * controllerMode;
            U = U_LQR;
        end

        %[~, X] = ode15s(@(t, x) f(x, U, params), tspan, X_sim(:,k));
        %X_sim(:, k+1) = X(end, :)';
        X_sim(:, k+1) = RK4Step(@f, X_sim(:,k), U, dt, params);
        U_sim(:, k) = U;
        newU = U_sim(:, k); %For MHE input

        noise = noise_std * randn([nMeasurements, 1]);
        yNext(:, k+1) = C * X_sim(:, k+1) - C * xlp + noise;
        yNext_f(:, k+1) = alpha * yNext(:, k+1) + (1-alpha) * yNext_f(:, k); %EMA
        newY = yNext_f(:, k+1); %For MHE input
        mhe = mhe.runMHE(newY, newU); %Run MHE with newY and newU
        xEst = mhe.xCurrent; % Extract xhat
        MHE_est(:, k+1) = xEst;
        vsol(:, k+1) = mhe.vCurrent;
        wsol(:, k+1) = mhe.vCurrent;


        NIS_current = mhe.currentNIS;
        NIS_traj(k) = NIS_current;
        Innovations_traj(:,k) = mhe.currentInnovation;
        error = xEst - (X_sim(:, k+1) - xlp);
        toterror = toterror + vecnorm(error,2);
        NEES_traj(k) = error' / mhe.currentP * error;

        %profile viewer
        elapsed = toc(t_start)
    end
```

5.4.6 RMSE test

An important part of MPC, and optimization control in general, is the choice of prediction horizon N. To evaluate the impact of the discretization timestep `dt` and choice of N on the performance of the MPC, a grid-based parameter sweep was conducted. A range of `dt` and `N_MPC` was stored

in `dtList` and `NList` which were systematically tested, with each unique combination forming an independent closed-loop MPC simulation. For each setting, the MPC first first computed an open-loop trajectory using the initial condition, then executed the closed-loop control by solving the MPC problem at each timestep and applying the fist control sequence. The resulting closed-loop state trajectory was then compared to the open-loop reference to compute the Root Mean Square Error (RMSE) for each state. These RMSE values were stored in a structured format for analysis in the variable `results`. The mean RMSE across all states served as a scalar performance metric to quantify the deviation from the reference, and was used to populate a grid for performance visualization. This approach enabled a comprehensive evaluation of the controller's sensitivity to different temporal resolutions and horizon lengths, helping to identify optimal parameters for minimizing trajectory error and balancing computational cost. The code used is listed in Listening 5.4.6.

Listing 10: The RMSE test

```
dtList = 0.001:0.002:0.01;
NList = 2:4:100;

[dtGrid, NGrid] = meshgrid(dtList, NList);  % Grid for plotting
RMSEsum = zeros(length(dtList), length(NList));
RMSEmatrix = zeros(length(dtList), length(NList), nStates);
result = struct();

k = 1;

for i = 1:length(dtList)
    for j = 1:length(NList)

        dt = dtList(i);
        N_MPC = NList(j);

        fprintf("Running closed-loop MPC: dt = %.4f, N = %d\n", dt,
            N_MPC);

        mpc = MPCclass(N_MPC, Ac, Bc, X0, dt, NT, [], [], ...
            Q_MPC, R_MPC, nStates, nControls, MPC_options, ...
            xRef, Xref, [], [], []);

        [xOpenLoop, uOpenLoop] = mpc.runOpenLoop();
        tVec = 0:dt:N_MPC*dt;

        X_sim = zeros(nStates, length(tVec));
        U_sim = zeros(nControls, length(tVec)-1);
        X_sim(:, 1) = X0;

        for l = 1:length(tVec)-1
            [Xopt, Uopt] = mpc.runMPC(X_sim(:, l));
            U_sim(:, l) = Uopt;
            X_sim(:, l+1) = mpc.A * X_sim(:, l) + mpc.B * Uopt;
            % X_sim(:, k+1) = RK4Step(@f, X_sim(:,k), Uopt, dt, params)
                ;

        end

        RMSE = sqrt(mean((X_sim - xOpenLoop).^2, 2));
        RMSEscal = mean(RMSE);
        RMSEsum(i, j) = RMSEscal;

        result(k).dt = dt;
        result(k).N = N_MPC;
        result(k).xOpenLoop = xOpenLoop;
        result(k).X = X_sim;
```

```
            result(k).U = U_sim;
            result(k).RMSE = RMSE;
            k = k + 1;

            RMSEmatrix(i, j, :) = RMSE;
        end
    end
```

Since this is not technically an experiment, but rather a useful test for which the result decides the optimal tuning parameters `N_MPC` and `dt`, the result is briefly reviewed in this section for which the choice of parameters was used further for the experiments explained in Section 5.5.

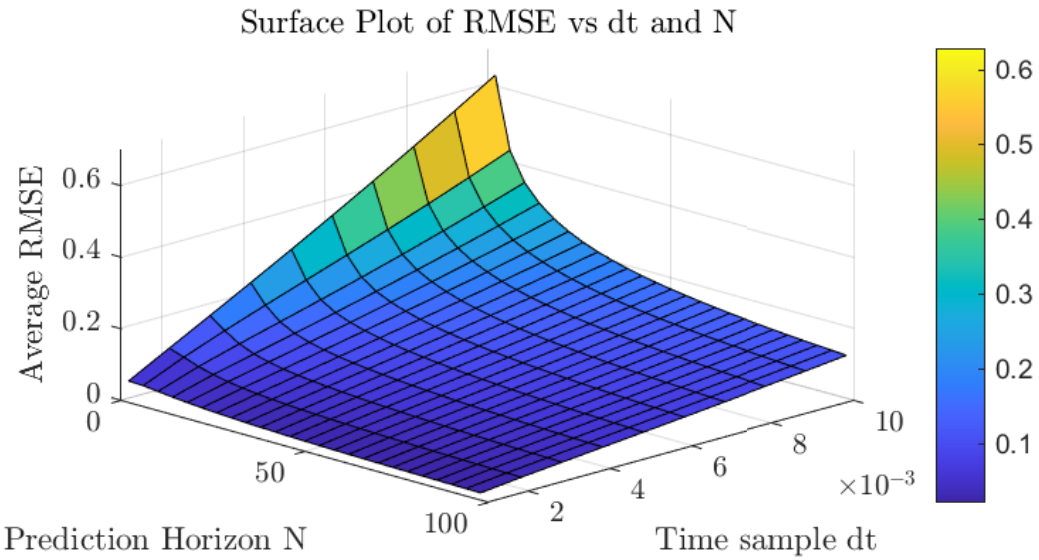


Figure 6: RMSE result with several horizon N and timestep dt, with open-loop vs. closed-loop.

As seen in Figure 6, when timestep is chosen lower than 0.004 the average RMSE isn't significantly high unless using a horizon $N \leq 10$. We want to have as little simulation time as possible while having the average RMSE as low as possible. Based on the heatmap in Figure 6, a timestep of $dt = 0.003$ combined with a horizon of $N = 20$ achieved a favorable trade-off between control accuracy and computational efficiency. At this configuration, the RMSE remained below the threshold of 0.1, indicated by dark blue region, making it the optimal choice for subsequent experiments.

5.5 Experiments

To evaluate the MPC and the control loop for the full control system, we conducted some experiments. Each of these experiments had different purposes to fully investigate the implementations. Each of these experiments are presented in this section, along with their purpose and how we evaluated each of the results. Each experiment is done for both the MPC and MHE combination, for the fully closed loop, and for only the MPC on the true optimal calculated states.

5.5.1 Experiment 1 - Disturbance Rejection

An important feature of MPC in general is the robustness of the controller, which is why the comparison of the MPC and RMPC is made during this thesis. In real-world scenarios, systems are often subject to unpredictable forces, such as vibrations, impacts, or environmental perturbations. For the magnetic levitation system, it is important that the controller can reject such disturbances while obtaining stability without excessive oscillations, etc. The purpose of this experiment was to compare the MPC and the RMPC, while being affected by a sudden disturbance to see if the robustness in the RMPC is an important feature for the maglev system.

To perform this experiment in simulation, an external force was applied as a pulse input to the controllers, to simulate a sudden push on the levitating magnet. There were tested with two different disturbances with different scales to see how the two controllers react, and if the robustness plays an important role when such problems occurs.

5.5.2 Experiment 2 - Model Mismatch

In real world control systems the exact model parameters for the real system is often unknown, and might be estimated incorrectly or changed over time. Therefore, it is important to test how the controller reacts if there is a model mismatch between the real system and the controller. The goal of this experiment was to test both the MPC and RMPC with model mismatch to see if the robustness is an important feature when controlling the magnet with a mismatched model.

In order to perform such an experiment, there were introduced deliberate discrepancies into the internal model used by the MPC and RMPC. Specifically, parameters such as the mass of the levitating magnet, magnetic force, or damping coefficient, etc., were slightly altered in the controllers linearized model. The nonlinear dynamics of the simulated model were unchanged which reflects a realistic scenario where the controllers were designed based on an imperfect system model.

5.5.3 Experiment 3 - Constraint Handling and Feasibility

As stated in Section 4.3, a huge part of MPC in general is the constraint handling. The third experiment was conducted to test the MPC and RMPC's ability to handle strict control bounds. We want the controller to use as low input as possible while still obtaining stability, which was the sole purpose of this experiment. Another huge part of the experiment was to compare how well each of them handle the strict bounds, and see if the robustness part of the MPC is an important part in constraint handling.

To perform the experiment, the lower and upper bounds presented in Table 3 were tightened for the control inputs. The bounds were tightened as much as possible while still maintaining stability. Another test were added for this experiment, slack variables, which is discussed in Section 5.2.3. The goal of the use of slack variables were to see if the MPC and RMPC use less control effort all around, with even stricter bounds but letting them violate the bounds to some degree throughout the simulation.

5.5.4 Experiment 4 - Reference Tracking

The fourth experiment was the reference tracking problem for the MPC and RMPC. This experiment's purpose was to compare the MPC and RMPC's ability to accurately follow a given reference trajectory that deviated from the nominal equilibrium position. Reference trajectory is a fundamental task for any controller which demonstrates whether the controller would reposition the magnet precisely to a new given location within the system's operating range. During this experiment, there was used a circular reference trajectory to move the magnet in a circle during the simulation time.

During the experiment, the reference point is constantly changed at each timestep with the reference trajectory being time-varying. How this was set up is discussed in Section 5.2.1, where at each iteration the reference point for the MPC and RMPC is updated to the current reference step, $\mathbf{x}_k^{ref}$. The purpose of the experiment was to compare the two MPC setups to see how important robustness is for a MPC for a reference tracking problem in case of the magnetic levitation system.

6 Results

In this section, all results from the experiments discussed in Section 5.5 will be presented. Each experiment result has an individual section to get a clear review of each result.

6.1 Experiment 1 - Disturbance Rejection

The experiment in this section was done with the tuning parameters listed in Table 4.

Table 4: Tuning Parameters for Experiment 1

Tuning Parameter	Value	Description
`N_MPC`	20	MPC Prediction horizon
`N_MHE`	15	MHE Prediction horizon
`dt`	0.003	Time Step
`t`	1	Simulation Time
`R_MPC`	$2 \cdot \mathbf{I}_4$	MPC Control weight
`Q_MPC`	$\text{diag}\begin{smallmatrix}(2000, & 2000, & 7500, & 100, & 100, \ldots \\ 1, & 1, & 1000, & 1, & 1)\end{smallmatrix}$	MPC State weights
`R_LQR`	$10^2 \cdot \mathbf{I}_4$	LQR Control weight
`Q_LQR`	$\text{diag}\begin{smallmatrix}(10, & 10, & 10, & 10, & 10, \ldots \\ 10, & 10, & 10, & 10^5, & 10)\end{smallmatrix}$	LQR State weights
`alpha`	0.9	EMA Constant
`noise_std`	10^{-4}	Magnetic sensor noise
`R_MHE`	$(\texttt{noise_std}^2 \cdot \mathbf{I}_9)^{-1}$	MHE Measurement noise weights
`Q_MHE`	$10^6 \cdot \text{diag}\begin{smallmatrix}(1, & 1, & 1, & 1, & 1, \ldots \\ 5, & 5, & 5, & 5, & 5)\end{smallmatrix}$	MHE Process noise weights
`Qscaling`	$5 \cdot 10^3$	Post-scaling for `Q_MHE`
`weightScaling`	10^{-4}	MHE Uniform scaling for weights
`X0`	$(0.003,\ 0.002,\ \text{zeq} + 0.002,\ \mathbf{0}_{1\times 7})^\top$	Initial state
`disturbance_1`	(10, 6, 0, 0)	Low disturbance
`disturbance_2`	(15, 10, 0, 0)	High disturbance

6.1.1 Low Disturbance

In this part of the experiment, a sudden low disturbance, denoted `disturbance_1` in Table 4, was introduced during the simulation to evaluate how the MPC and RMPC responded to external perturbations. The goal was to assess whether the additional robustness of the RMPC improved performance in disturbance rejection. The experiment was conducted using the tuning parameters listed in Table 4.

Figures 7 - 8 show the position, orientation, and their respective velocities for both MPC, in blue dashed line, and RMPC, in solid red. The red-shaded region indicated the period when the disturbance was active, and the light blue shading indicates when the advanced control was active, as discussed in Section 5.4.5. From Figure 7, it is evidant that both controllers react to the disturbance. However, the RMPC demonstrates improved disturbance rejection. Notably, the RMPC maintains a tighter bound around the equilibrium and stabilizes more quickly following the added disturbance. This is especially prominent in the angular states, α and β, where the MPC exhibits significantly larger deviations and longer recovery times.

In Figure 8, further reinforce these observations. The RMPC leads to reduced velocity spikes and faster decay toward the nominal trajectory. While both controllers ultimately stabilize the system,

RMPC achieves this more efficiently and with less fluctuation, indicating improved robustness to the external disturbance input. Figures 9 - 10, show the control sequences applied by the RMPC and MPC, respectively. Both controllers respond aggressively to counteract the disturbance, but the RMPC exhibits slightly more variation in its control signals. This increased variability likely reflects the additional feedback correction term introduced by the formulation in Section 5.2.2, allowing the RMPC to react more adaptively to unexpected deviations.

Overall, the results show that the RMPC provides improved performance in handling external disturbance. Its ability to constrain deviations and stabilize the system faster, particularly for the angular positions, demonstrates the value of robust feedback correction in the control of the maglev system.

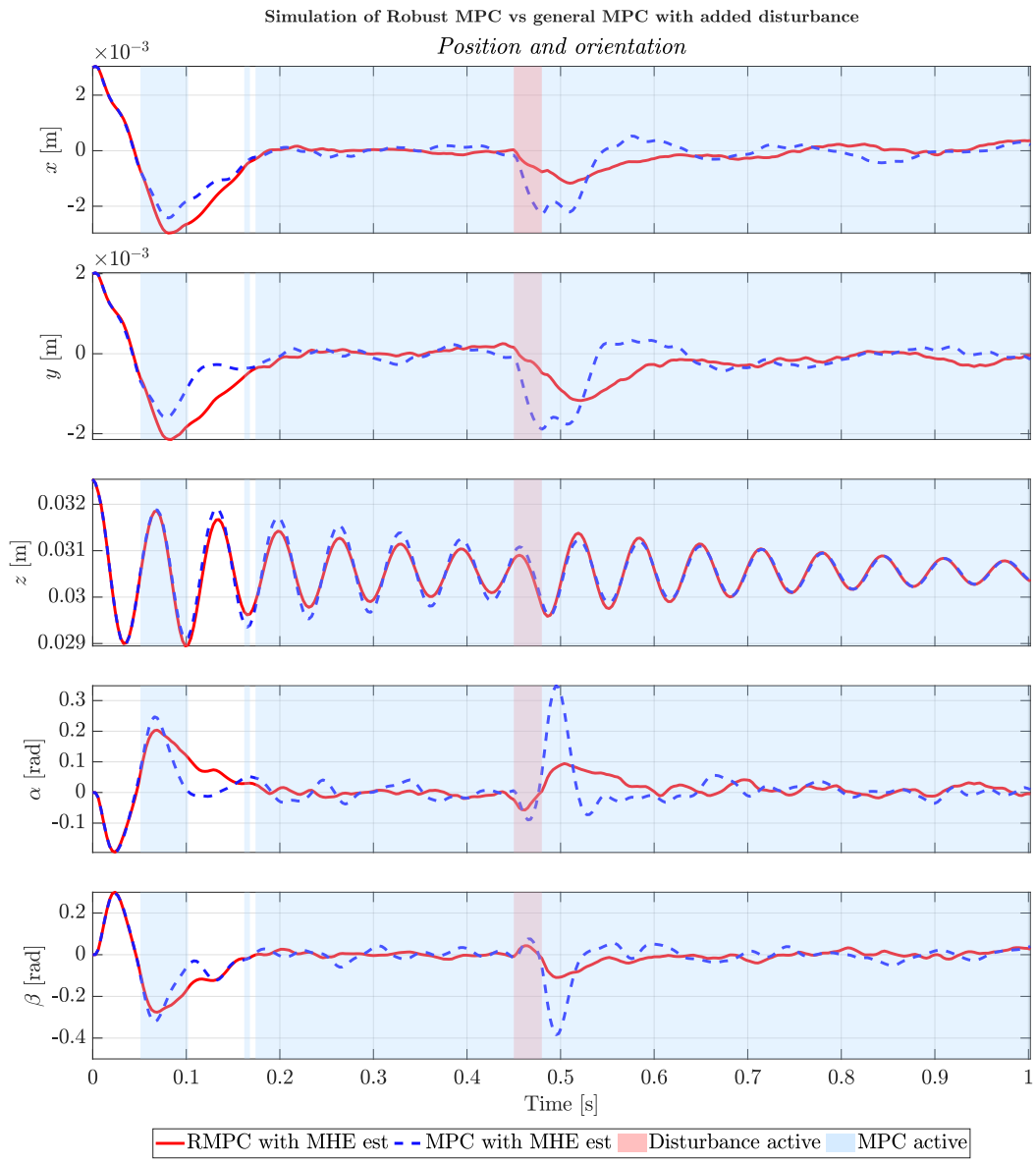


Figure 7: Simulation of state positions for RMPC and MPC with `disturbance_1` (See Table 4) added.

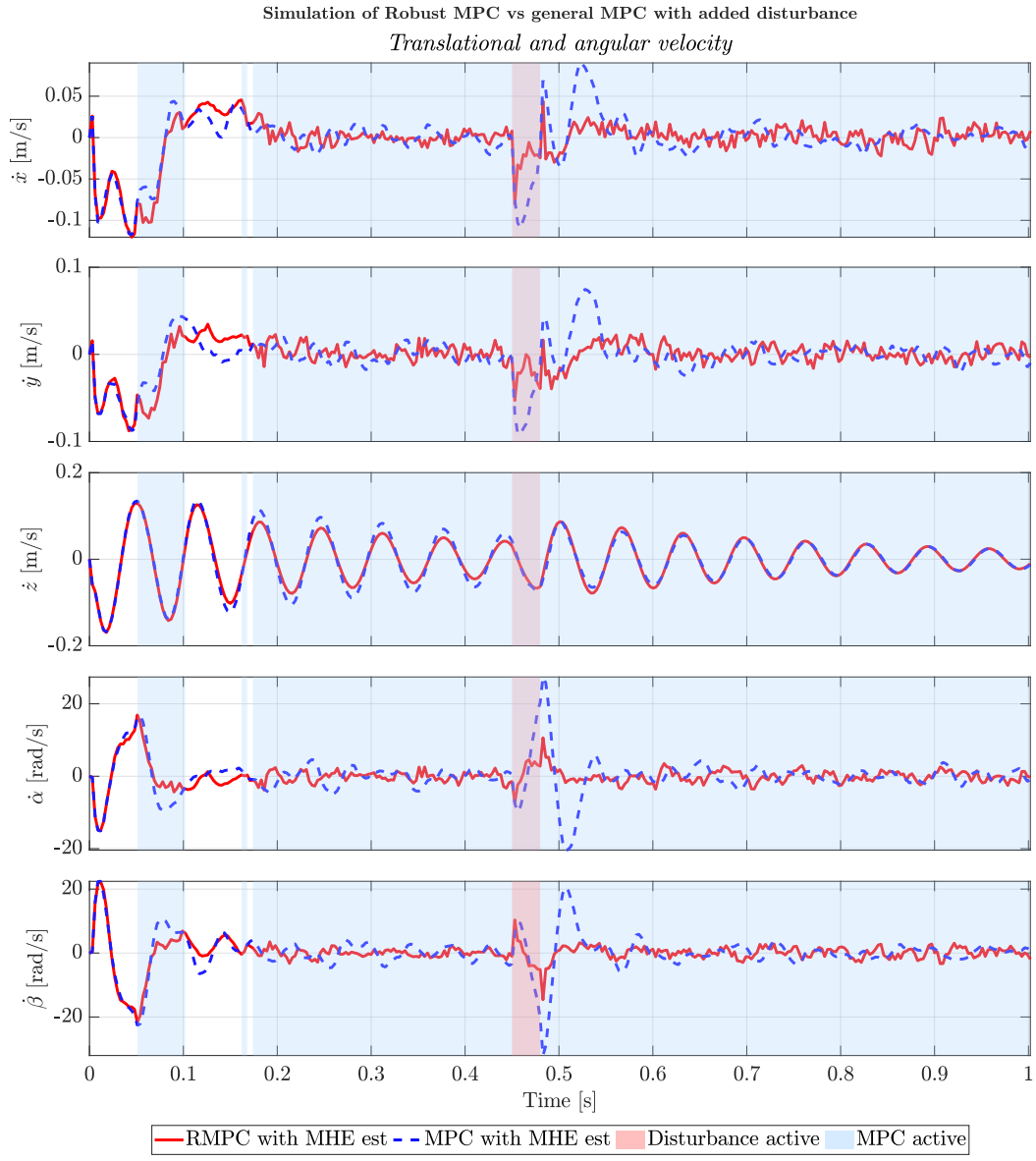


Figure 8: Simulation of state velocity for RMPC and MPC with `disturbance_1` (See Table 4) added.

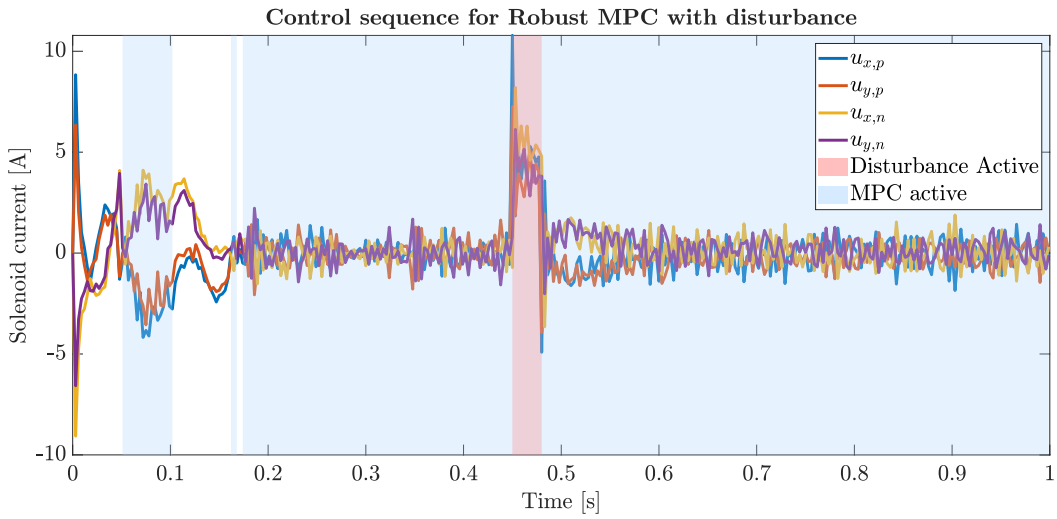


Figure 9: Simulation of control sequence for RMPC with `disturbance_1` (See Table 4) added.

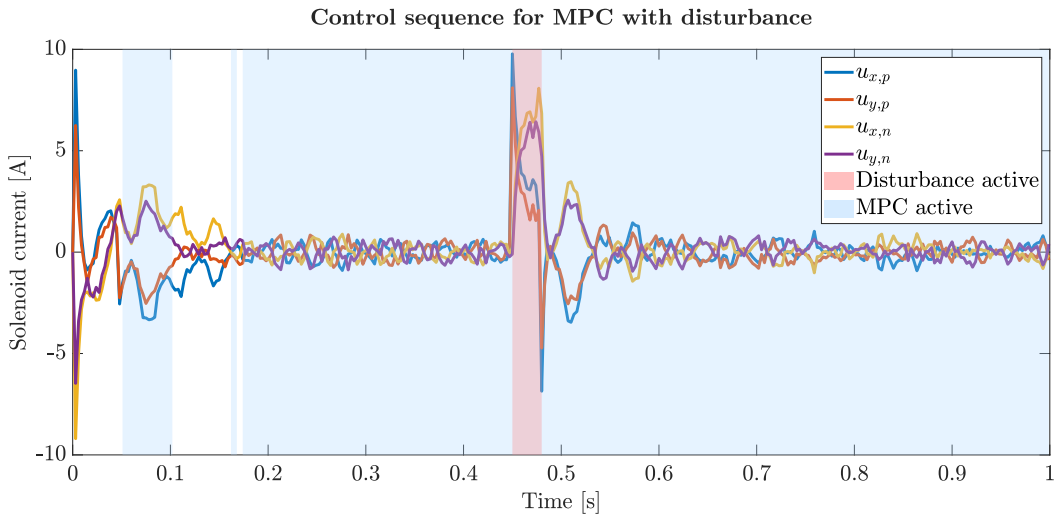


Figure 10: Simulation of control sequence for MPC with `disturbance_1` (See Table 4) added.

6.1.2 High Disturbance

In the second part of the disturbance rejection experiment, a larger external disturbance, referred to as `disturbance_2` in Table 4, was applied to further assess the robustness of the RMPC compared to the standard nominal MPC. This disturbance was applied in the same control channels as in the first part, specifically targeting the two positive solenoids in the control vector.

Figures 11 - 12, show the resulting trajectories for the positions and their respective velocities for both RMPC and MPC with same referred colors as earlier. One key observation from these figures is the controller activity, explained in Section 5.4.5. When the RMPC is active is represented by the same light blue shade as earlier, however, for the MPC the darker shade represents the activation. As observed on the figures, the RMPC stays active throughout the whole simulation. In contrast, the MPC was deactivated shortly after the disturbance occurred, and the system defaulted to LQR control when the MPC and MHE combination failed the internal consistency checks described in Section 5.4.5. This indicates that the MPC was unable to maintain stable operation under this stronger disturbance, while the RMPC remained robust and fully functional. For the RMPC the positions deviated to some degree after the added stronger disturbance but quickly stabilized. For the height of the magnet, z, it oscillated somewhat in response to the disturbance but also stabilized quickly. For the MPC, on the other hand, the controller wasn't activated before the magnet was almost stabilized.

In figure 12, the respective velocities of the positions are shown, with a similar trend. The RMPC quickly damps out the disturbance-induced deviations, particularly in vertical velocity $\dot{z}$, whereas the MPC got deactivated and replaced by the LQR. The superior disturbance rejection of RMPC is further evidenced by the faster return to nominal velocities and fewer oscillations in angular velocity states.

Control input sequences for RMPC and MPC are shown in Figures 13 - 14, respectively. Once again, RMPC exhibits continuous and adaptive control behavior throughout the simulation. The MPC, however, got deactivated as stated earlier which highlights a critical limitation of the standard MPC implementation when faced with unmodeled or large disturbances.

Overall, this experiment clearly demonstrated the added value of robustness in RMPC. While the standard MPC loses feasibility or violates trust conditions under large perturbations, the RMPC maintained closed-loop stability and constraint satisfaction, underscoring its practical benefit in uncertain and unpredictable environments.

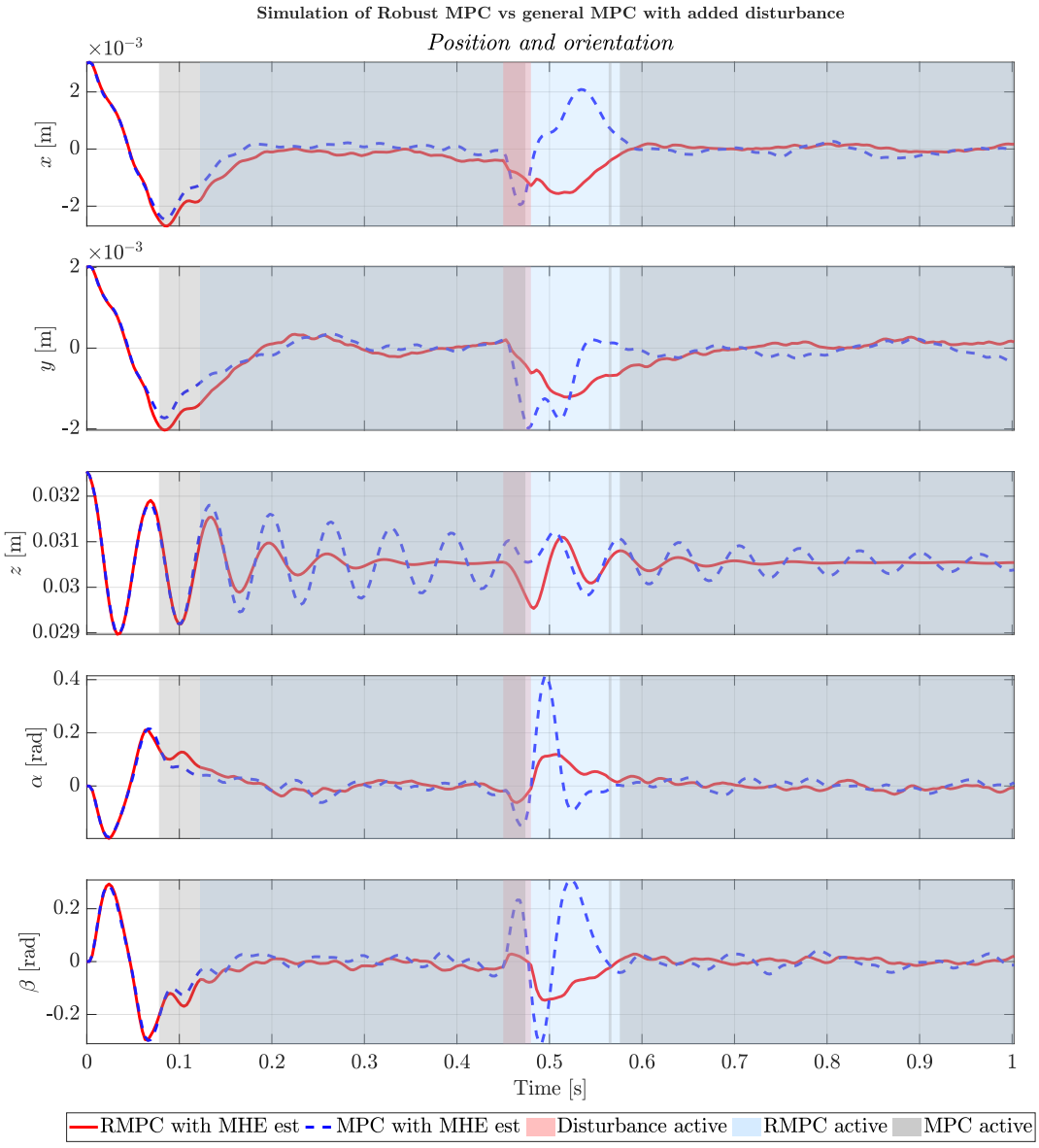


Figure 11: Simulation of state positions for RMPC and MPC with `disturbance_2` (See Table 4) added.

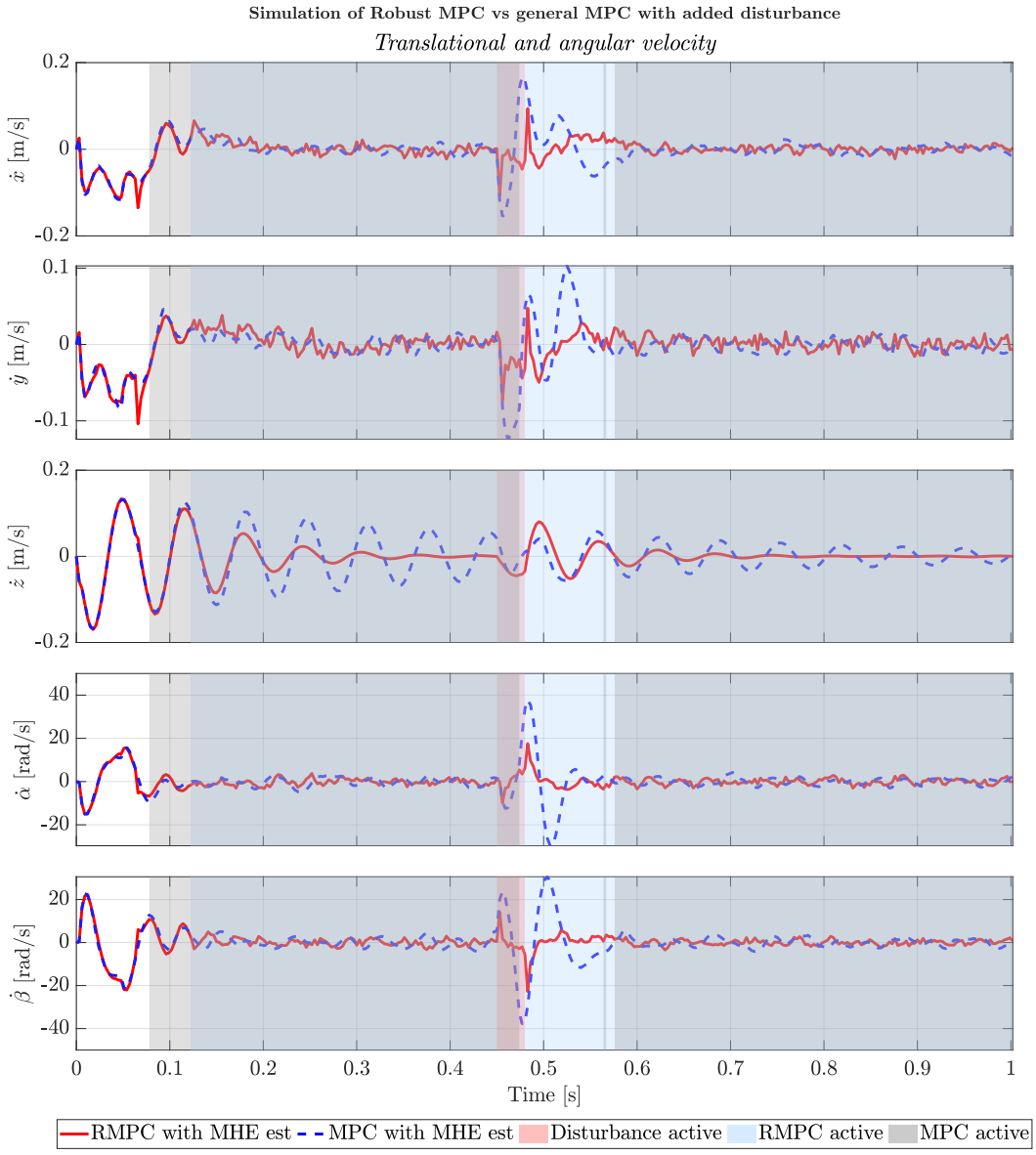


Figure 12: Simulation of state velocity for RMPC and MPC with `disturbance_2` (See Table 4) added.

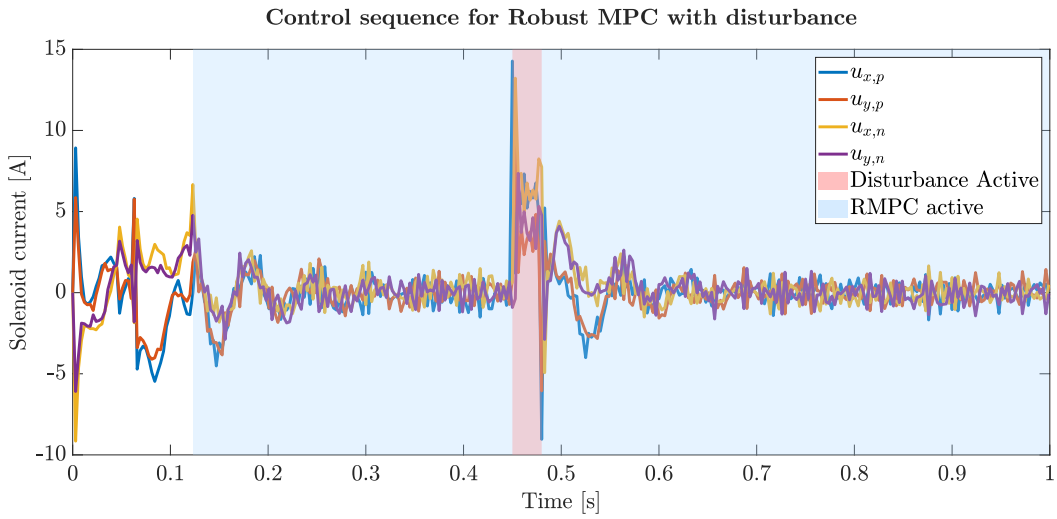


Figure 13: Simulation of control sequence for RMPC with `disturbance_2` (See Table 4) added.

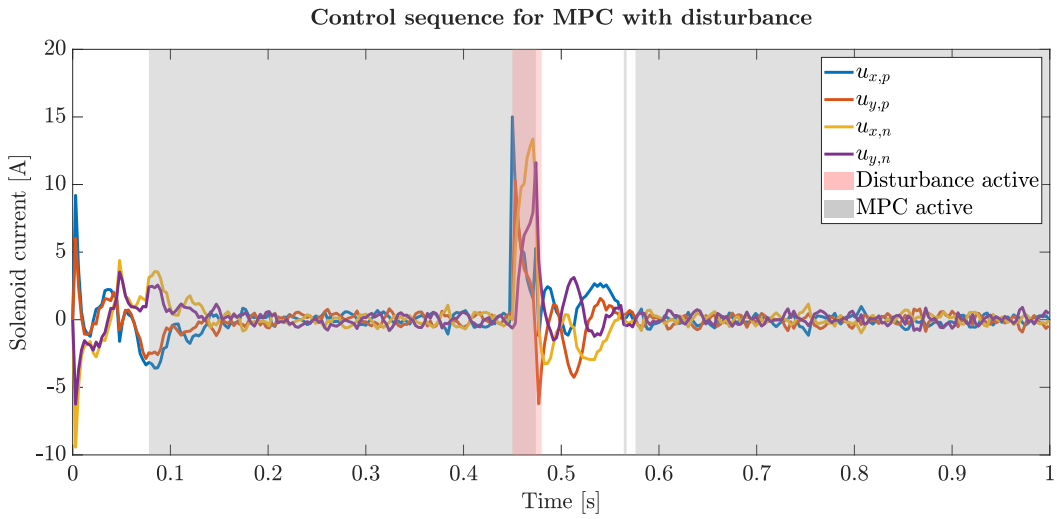


Figure 14: Simulation of control sequence for MPC with `disturbance_2` (See Table 4) added.

6.2 Experiment 2 - Model Mismatch

This experiment was conducted with the same tuing parameters as in Table 4. In this experiment we changed the model for just the MPC and RMPC to see how sensitive they were to model mismatch. The changes done is shown in Table 5. We decided to make the magnet heavier in the model to see if the controllers responded with more aggressive control action. We also made the radius of the levitating magnet higher, also for the base magnets. The last change, and perhaps the most significant one, we did was rearranging the base magnets a little more messier than originally.

Table 5: Model Changes for Simulation

Model Parameter	**Original**	**Adjusted**	`Description`
`M_lev`	0.07 `kg`	0.1 `kg`	Mass of levitating magnet
`r_lev`	0.02 `m`	0.022 `m`	Radius of levitating magnet
`r_base`	0.01 `m`	0.015 `m`	Radius of base magnets
`r_base_array`	(95a)	(95b)	Position array for the base magnets

$$\texttt{r_base_array} = \begin{bmatrix} 0.021 & 0 & -0.021 & 0 \\ 0 & 0.021 & 0 & -0.021 \\ 0.006 & 0.006 & 0.006 & 0.006 \end{bmatrix} \quad (\texttt{Original}) \tag{95a}$$

$$\texttt{r_base_array} = \begin{bmatrix} 0.022 & 0 & -0.019 & 0 \\ 0 & 0.021 & 0 & -0.020 \\ 0.006 & 0.006 & 0.006 & 0.006 \end{bmatrix} \quad (\texttt{Adjusted}) \tag{95b}$$

Figures 15 - 18 show the results of this experiment. Despite the mismatch in the internal plant, both MPC and RMPC remained stable and effective.

The RMPC showed slightly more consistent performance in handling angular deviations, which were more sensitive to the model changes. However, the difference in response between RMPC and MPC was relatively minor in this case. The RMPC appeared marginally better at damping the oscillations earlier and maintaining smoother convergence. This response yields both the positions and their respective velocities. The control inputs, shown in Figures 17 - 18, show a similar pattern for both controllers, but the RMPC exhibited marginally more aggressive actuation. In summary, both RMPC and MPC handled the plant model mismatch without loss of stability.

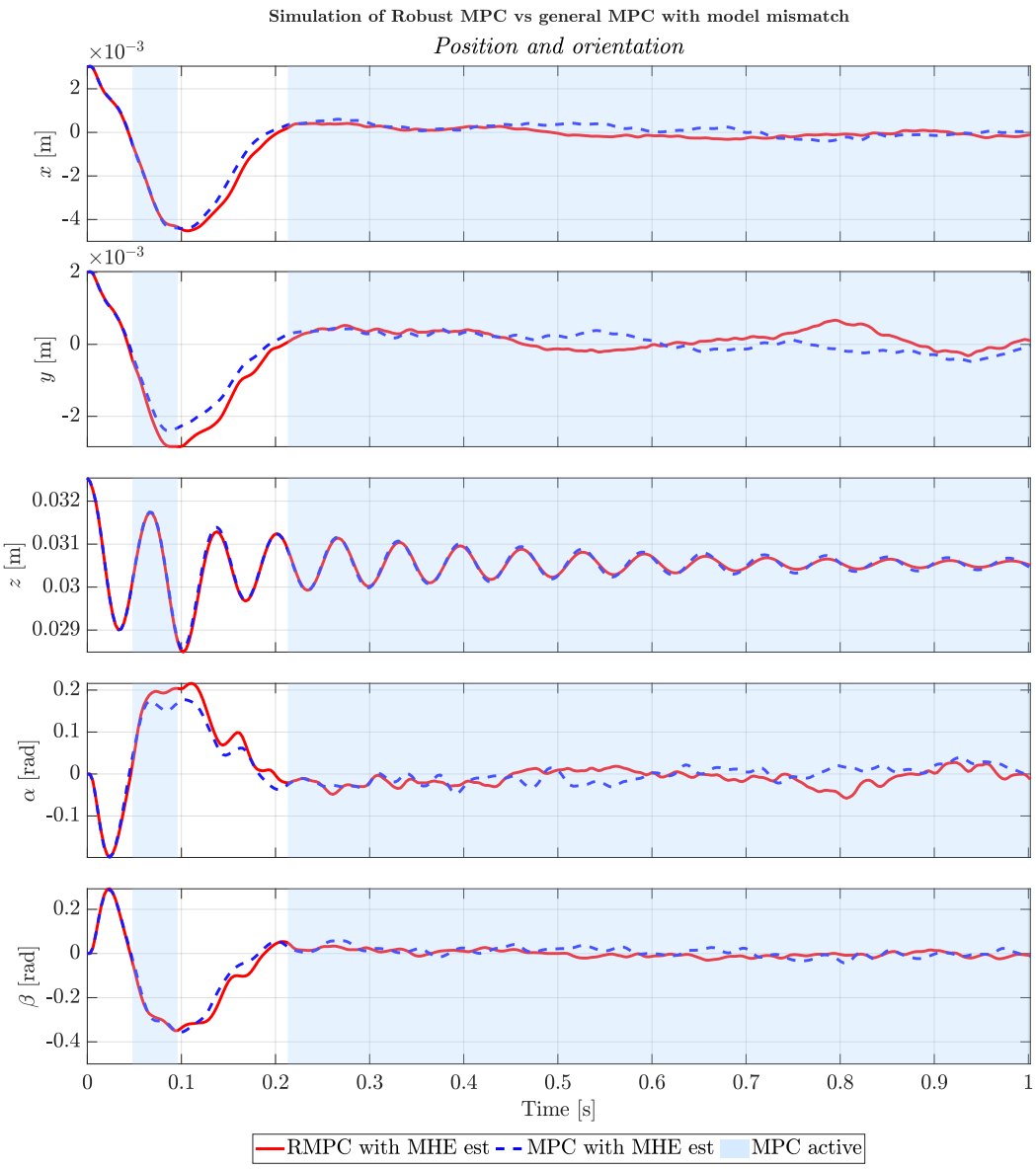


Figure 15: Simulation of RMPC and MPC with MHE estimates and model mismatch.

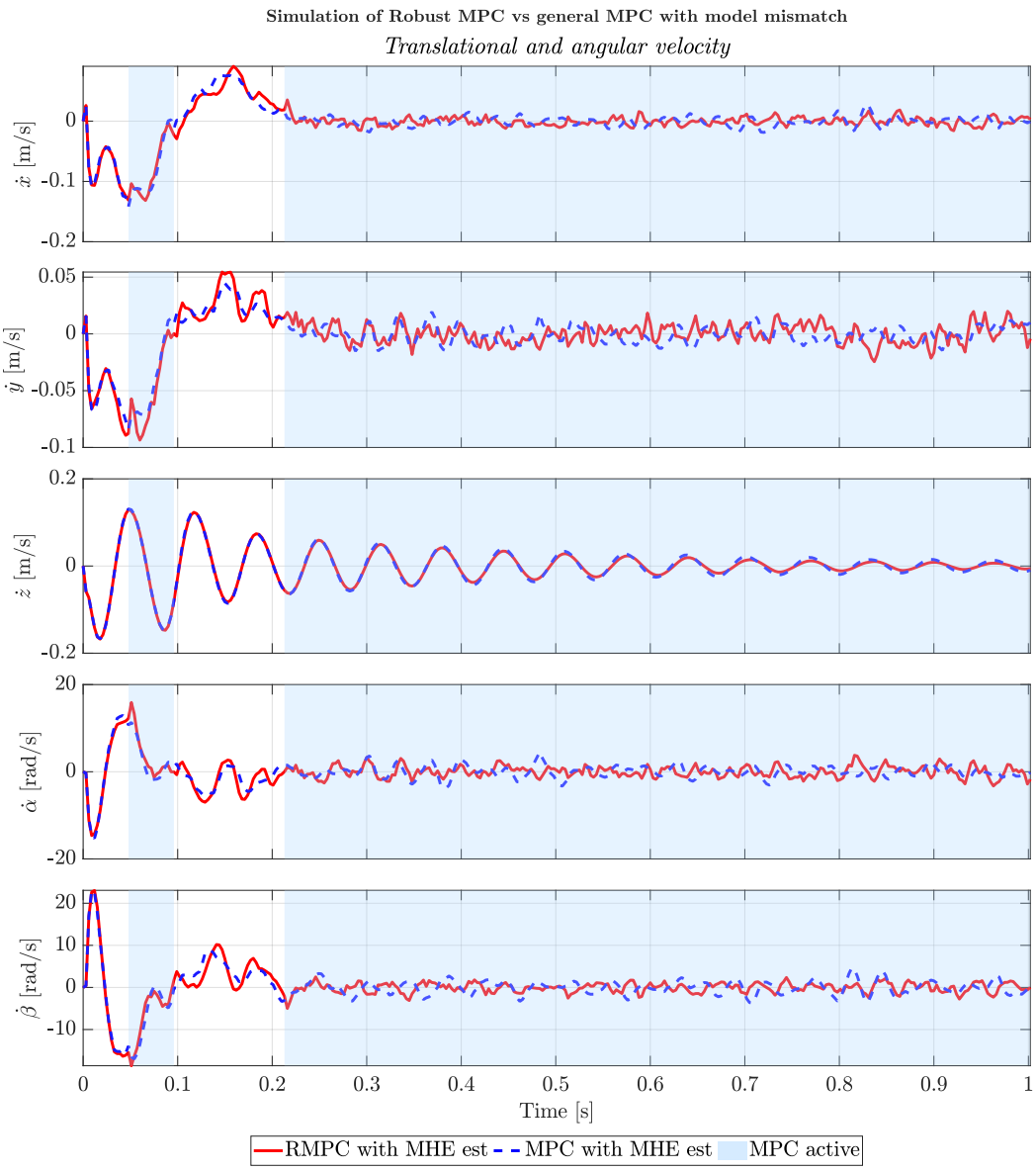


Figure 16: Simulation of RMPC and MPC with MHE estimates and model mismatch.

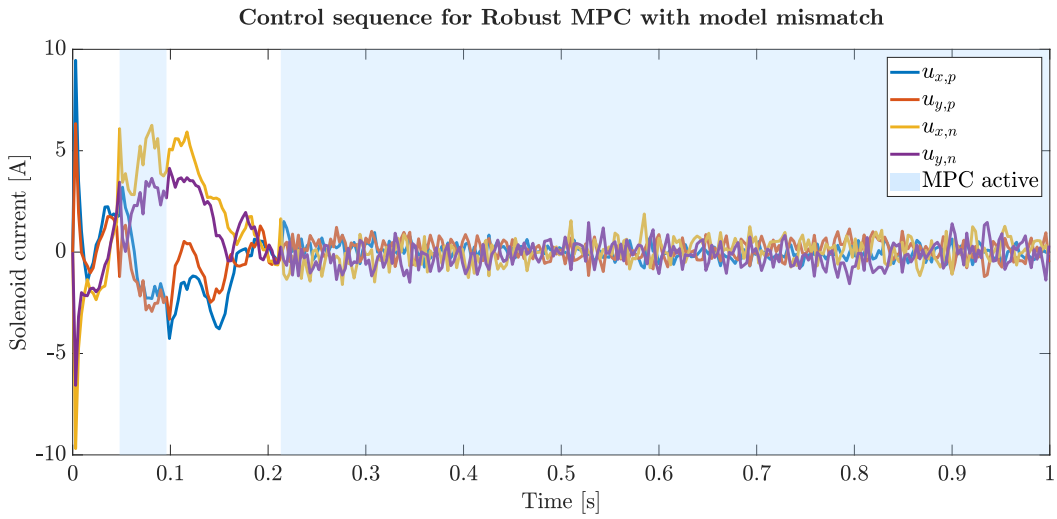


Figure 17: Control sequence of RMPC with MHE estimates and model mismatch.

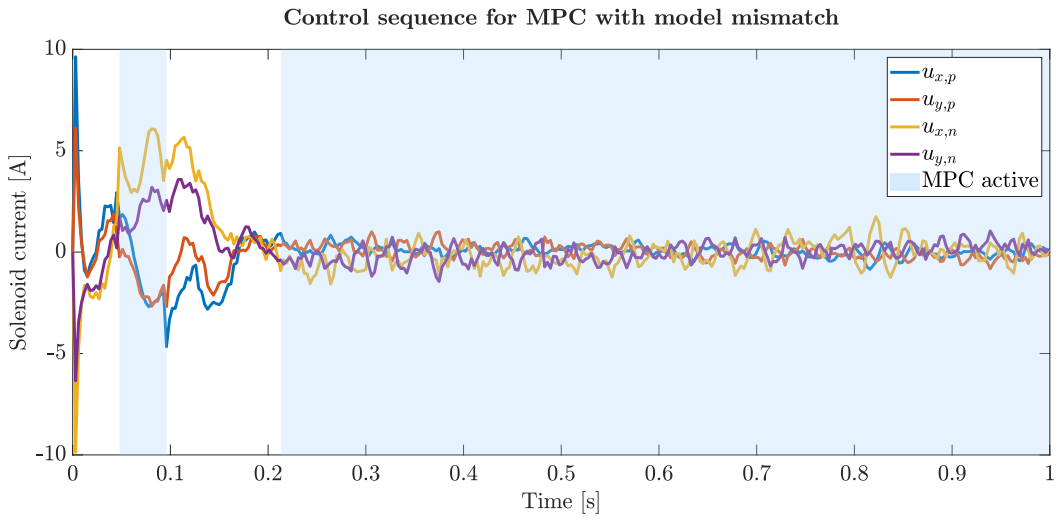


Figure 18: Control sequence of MPC with MHE estimates and model mismatch.

6.3 Experiment 3 - Constraint Handling and Feasibility

This experiment was based on testing how strict the control input bounds could be set for both MPC and RMPC and see how they both responded. The tuning parameters for this experiment is listed in Table 6, where we have two different upper and lower bounds. This experiment is separated into two parts, where the first part was used with `u_min_1` and `u_max_1`. The second part introduced slack variables to see how different the system reacts in both cases, with even stricter control bounds, by letting the controllers violate the bounds to some degree. The bounds for the second part were `u_min_2` and `u_max_2`, with weights on slack variables given by `rho` and `S`.

Table 6: Tuning Parameters for Experiment 3

Tuning Parameter	Value	Description
`N_MPC`	20	MPC Prediction horizon
`N_MHE`	15	MHE Prediction horizon
`dt`	0.003	Time Step
`t`	1	Simulation Time
`R_MPC`	$2 \cdot \mathbf{I}_4$	MPC Control weight
`Q_MPC`	$\text{diag}\begin{smallmatrix}(2000, & 2000, & 7500, & 100, & 100, \ldots \\ 1, & 1, & 1000, & 1, & 1)\end{smallmatrix}$	MPC State weights
`R_LQR`	$10^2 \cdot \mathbf{I}_4$	LQR Control weight
`Q_LQR`	$\text{diag}\begin{smallmatrix}(10, & 10, & 10, & 10, & 10, \ldots \\ 10, & 10, & 10, & 10^5, & 10)\end{smallmatrix}$	LQR State weights
`alpha`	0.9	EMA Constant
`noise_std`	10^{-4}	Magnetic sensor noise
`R_MHE`	$(\texttt{noise_std}^2 \cdot \mathbf{I}_9)^{-1}$	MHE Measurement noise weights
`Q_MHE`	$10^6 \cdot \text{diag}\begin{smallmatrix}(1, & 1, & 1, & 1, & 1, \ldots \\ 5, & 5, & 5, & 5, & 5)\end{smallmatrix}$	MHE Process noise weights
`Qscaling`	$5 \cdot 10^3$	Post-scaling for `Q_MHE`
`weightScaling`	10^{-4}	MHE Uniform scaling for weights
`X0`	$(0.003,\ 0.002,\ \text{zeq} + 0.002,\ \mathbf{0}_{1\times 7})^\top$	Initial state
`u_min_1`	-1.4 `A`	Lower bound on input without slack
`u_max_1`	1.4 `A`	Higher bound on input without slack
`u_min_2`	-0.04 `A`	Lower bound on input with slack
`u_max_2`	0.04 `A`	Higher bound on input with slack
`rho`	10^2	Weight on linear term - slack
`S`	$10^2 \cdot \mathbf{I}_{\texttt{N_MPC}\times\texttt{nControls}}$	Weight on quadratic term - slack

6.3.1 Without Slack variables

Figures 19 - 22 show the performance of the MPC controller, with and without MHE estimates, under strict control bounds defined by `u_min_1` and `u_max_1` in Table 6. The state trajectories and orientation, in Figure 19, demonstrate that the MPC were able to stabilize the system. However, a key observation is that the MPC controller remeined inactive for a significant portion of the simulation. This delay is due to the difficulty the MHE had in meeting the innovation consistency criteria needed for enabling MPC, see Section 5.4.5. Despite this, the system was successfully stabilized, primarily by the LQR during the early phase. This can be confirmed by inspecting the control sequences in Figures 21 - 22, which show large corrective actions before the MPC becomes active.

Figures 23 - 26 illustrate the results of the RMPC controller under same strict control limits. In contrast to the general MPC, the RMPC was able to activate earlier and maintain stability

throughout the simulation. This suggests that the inherent robustness of RMPC improves feasibility and stability maring under tight input constraints. Again, early stabilization is mainly driven by the LQR, which is visible in the contorl inputs shown in Figures 25 - 26. Notably, the RMPC also appears to use slightly more control effort compared to the nominal MPC, indicating that robustness may come to the cost of slightly more aggressive actuation when necessary.

Overall, this part of the experiment illustrates that both controllers can handle constrained inputs effectively, but the RMPC provides better feasibility and faster engagement under tight bounds. This is particularly beneficial in scenarios where precise control must be maintained without sacrificing stability.

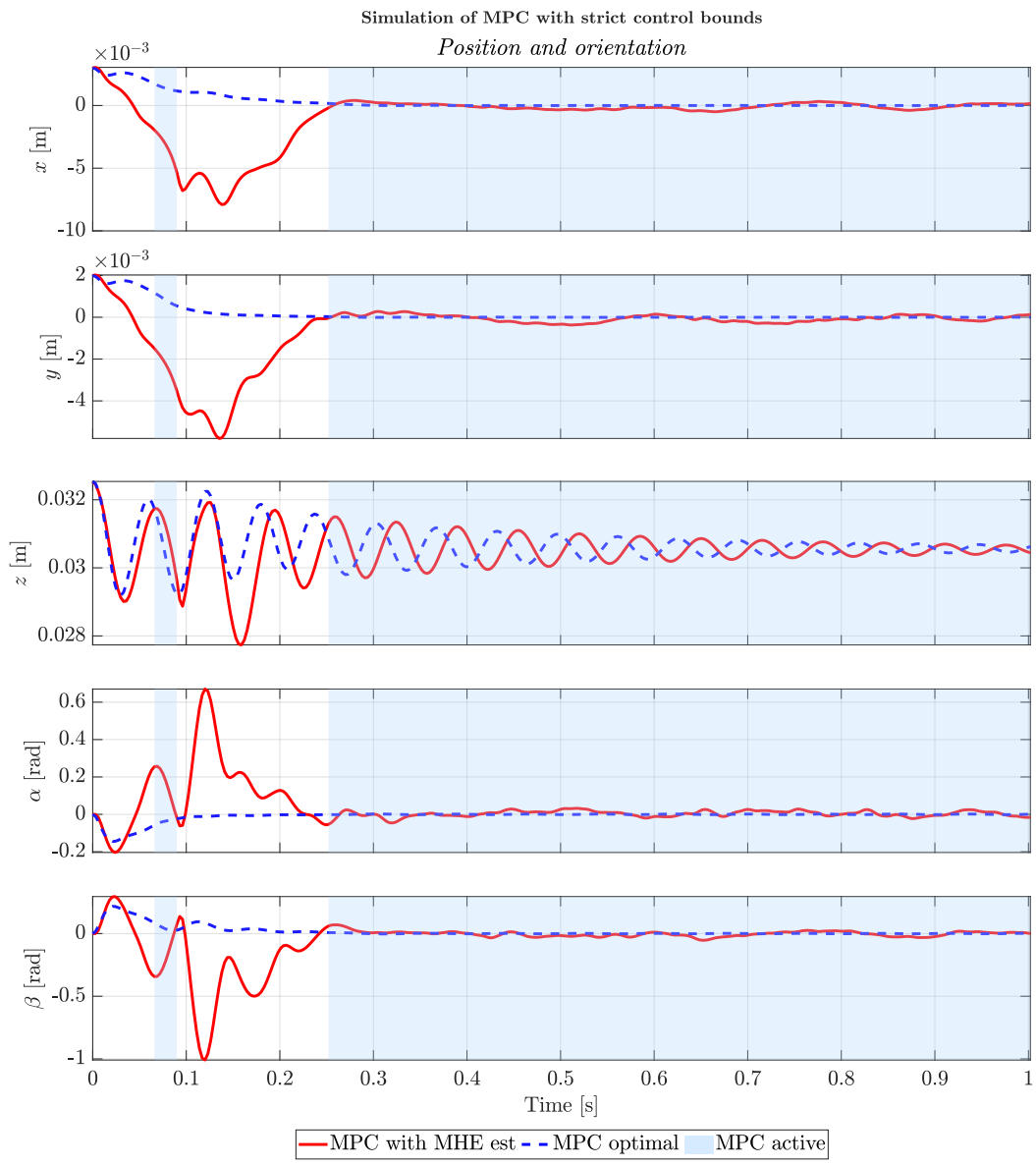


Figure 19: MPC simulation of position with and without MHE estimates with strict control bounds, see Table 6.

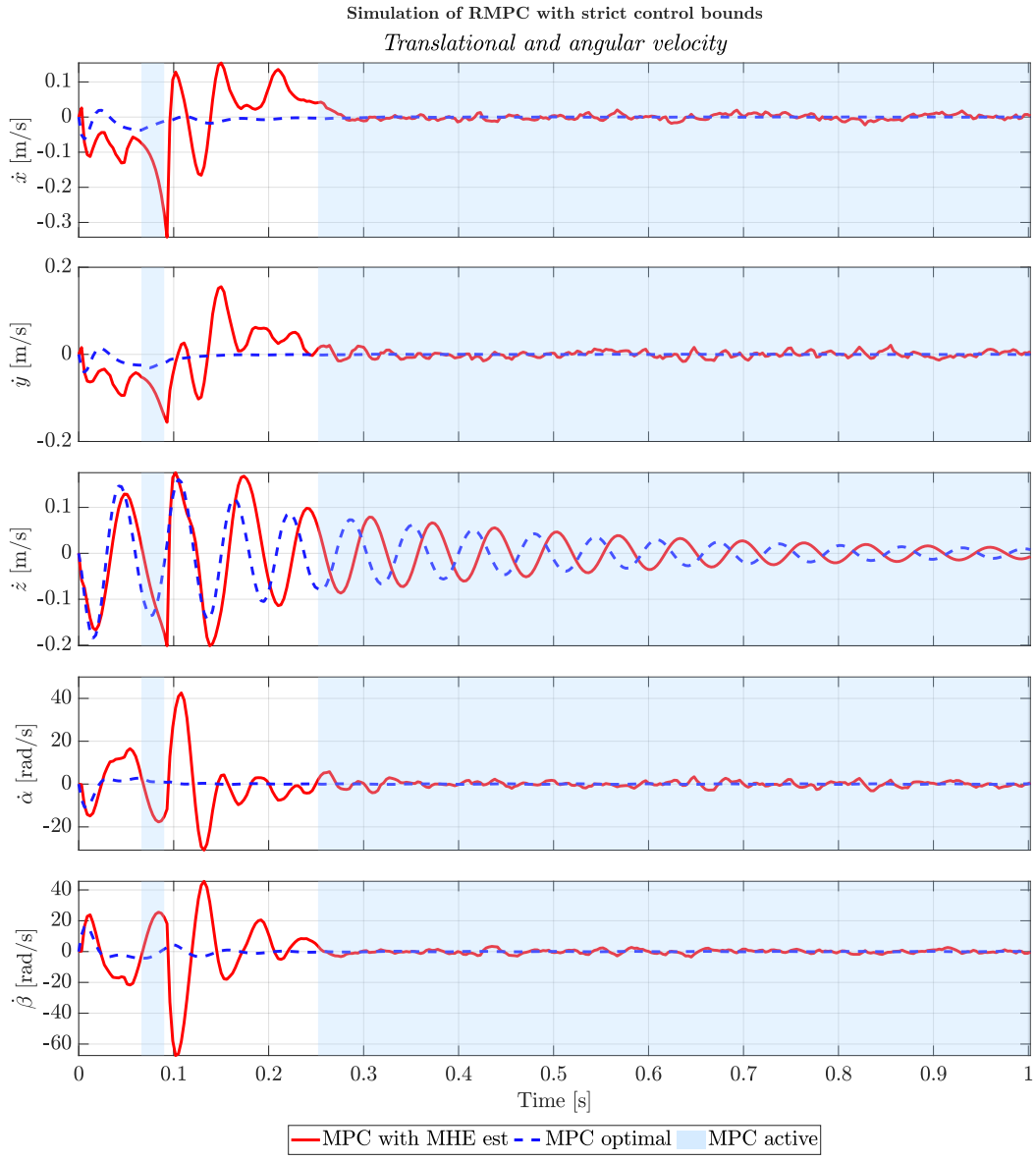


Figure 20: MPC simulation of velocity with and without MHE estimates with strict control bounds, see Table 6.

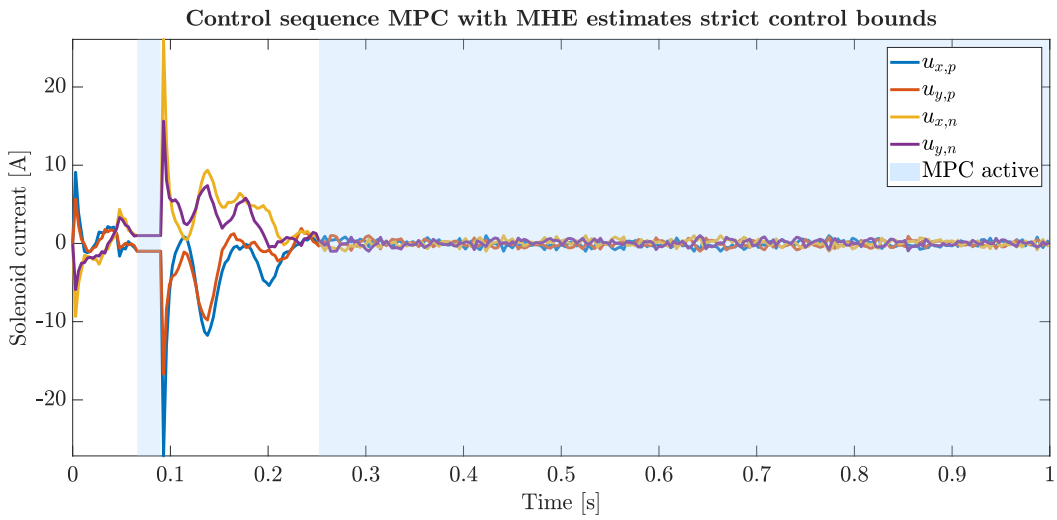


Figure 21: Control sequence for MPC with MHE estimates with strict control bounds, see Table 6.

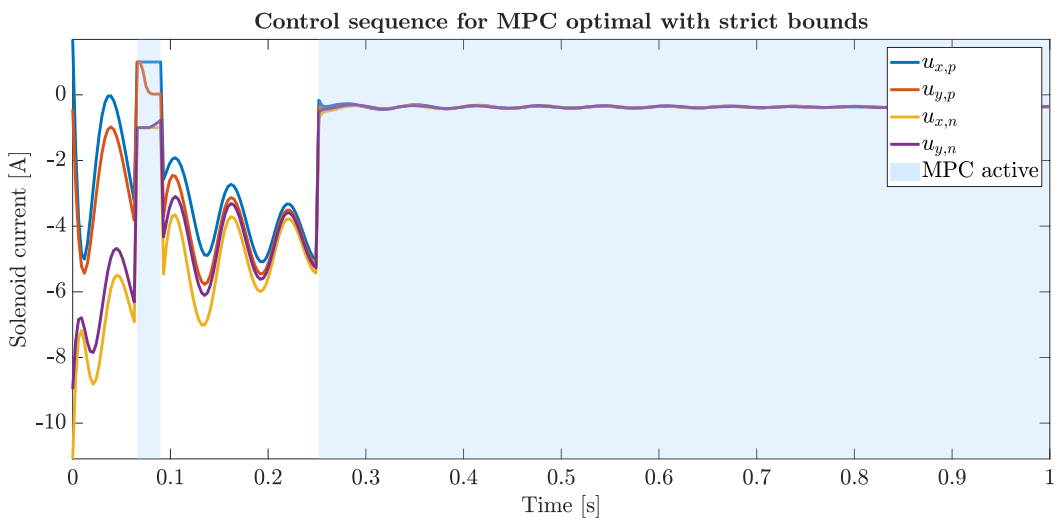


Figure 22: Control sequence for MPC optimal with strict control bounds, see Table 6.

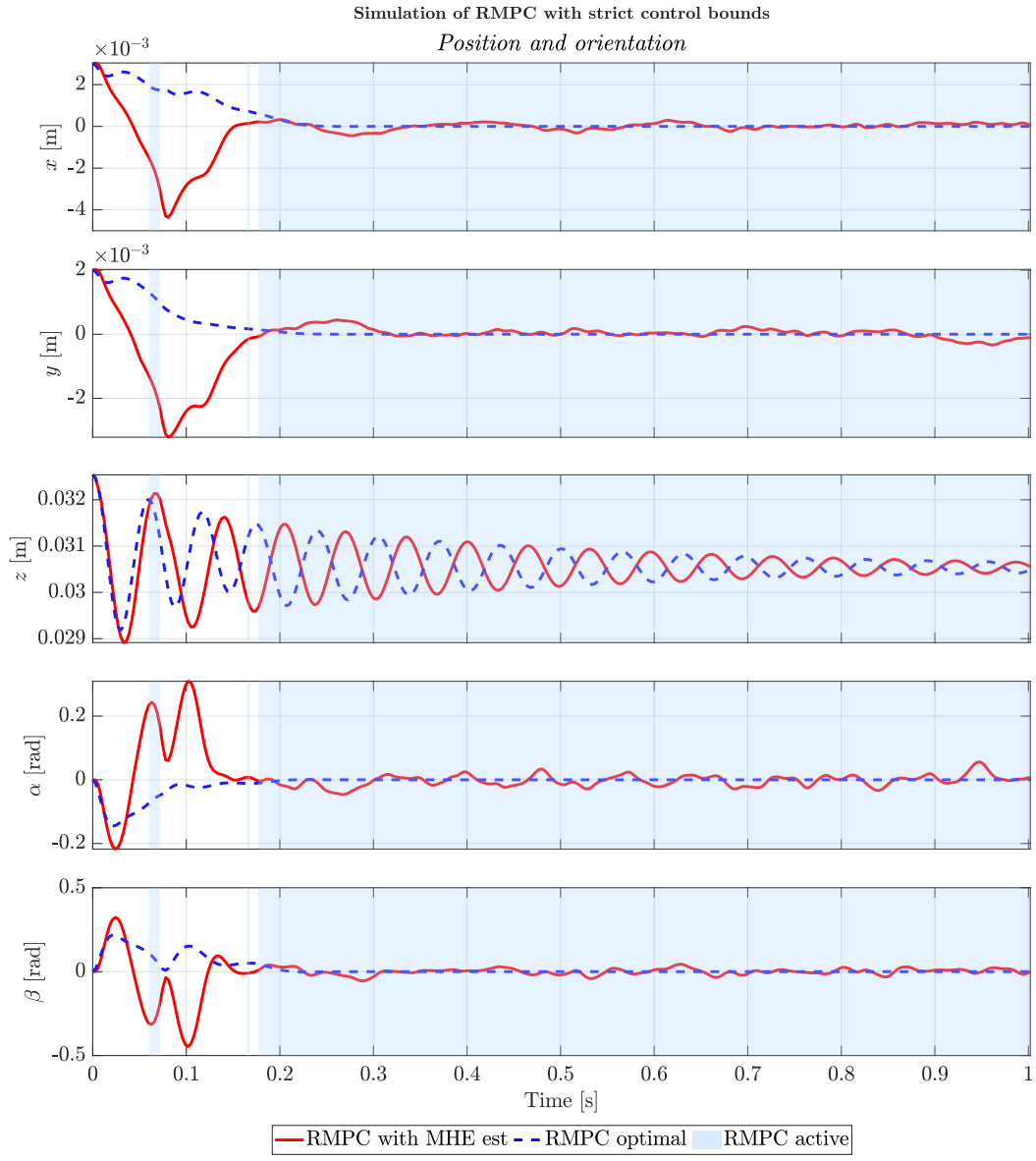


Figure 23: RMPC simulation of position with and without MHE estimates with strict control bounds, see Table 6.

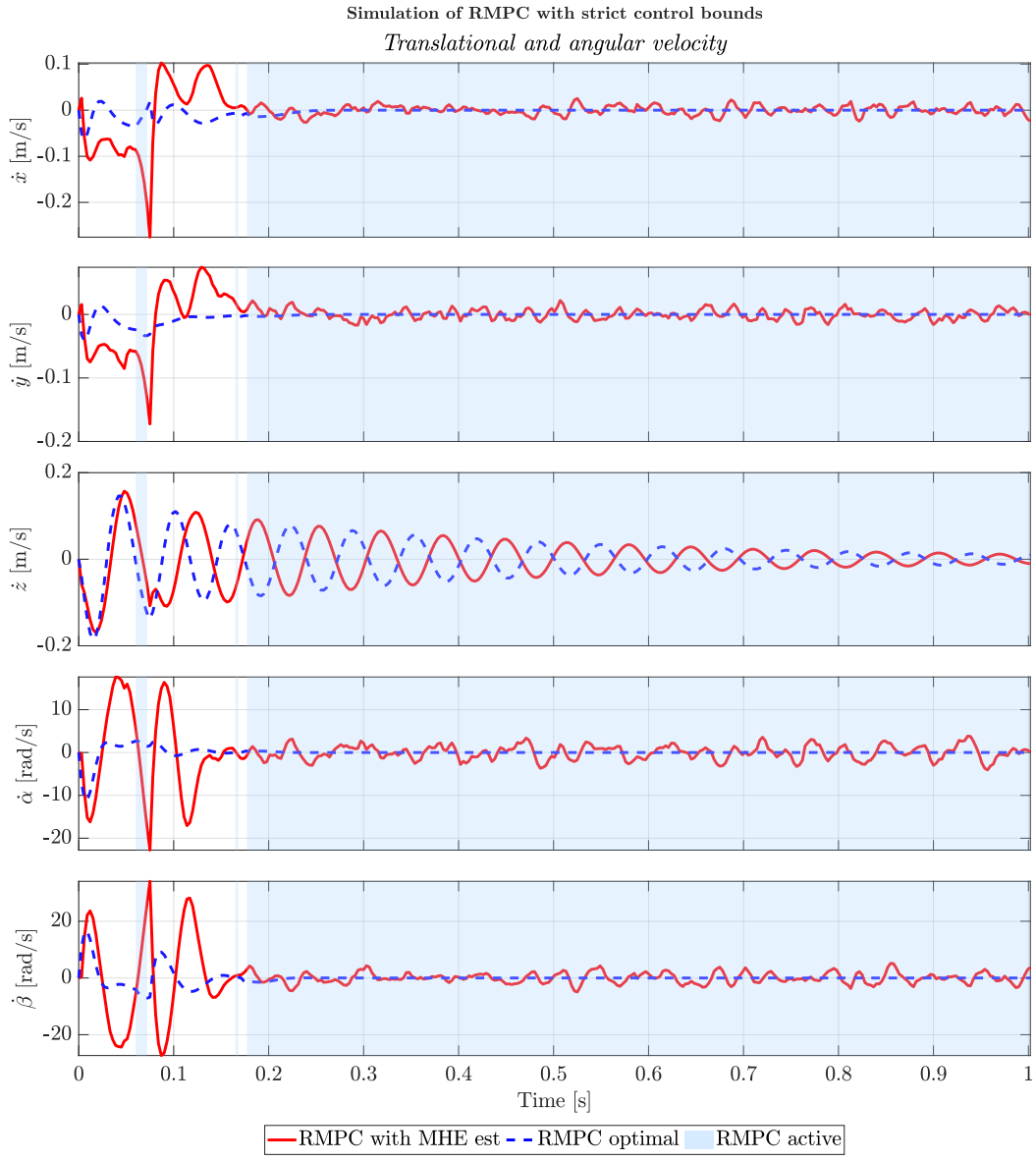


Figure 24: RMPC simulation of position with and without MHE estimates with strict control bounds, see Table 6.

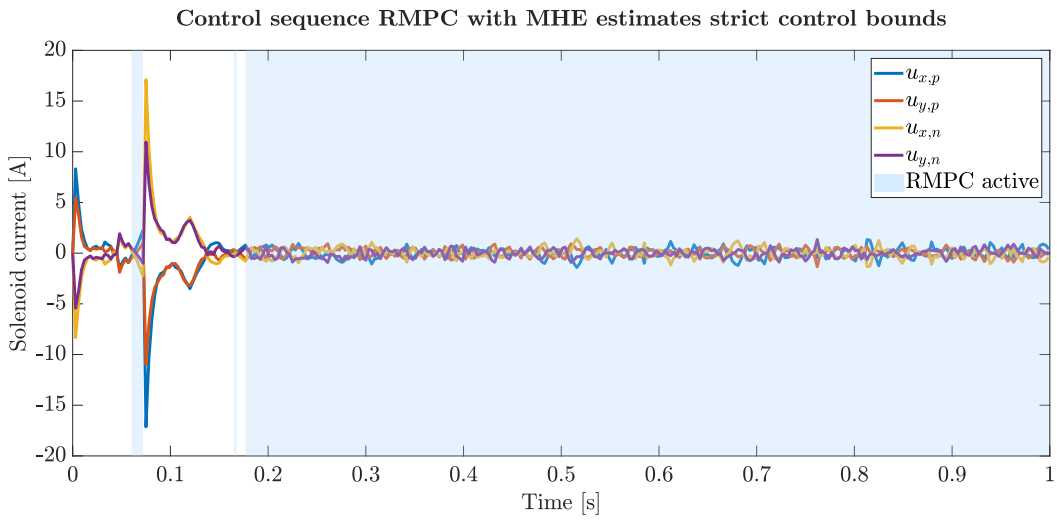


Figure 25: Control sequence for RMPC with MHE estimates with strict control bounds, see Table 6.

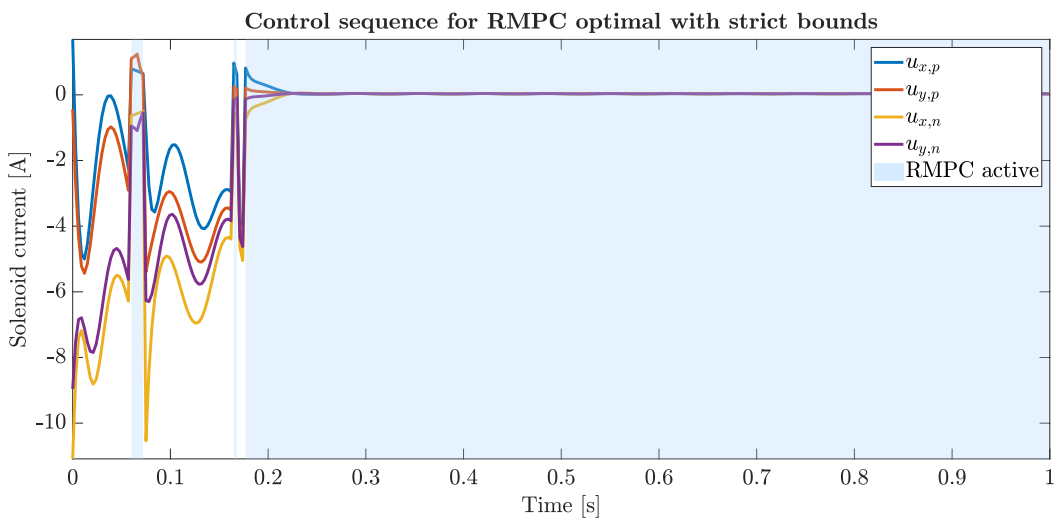


Figure 26: Control sequence for RMPC optimal with strict control bounds, see Table 6.

6.3.2 With Slack variables

In the second part of this experiment, slack variables were introduced to test whether even stricter bounds on the control inputs could be imposed without compromising system stability. The bounds `u_min_2` and `u_max_2`, in Table 6, were significantly narrower than in the previous experiment. By allowing controlled violation of these bounds through penalized slack variables, we aimed to maintain feasibility and reduce conservativeness in the control action.

Figures 27 - 28 display the position and velocity responses for MPC with and without MHE estimates, where red is with MHE and blue dashed is without. Compared to the case without slack variables, the MPC was activated much earlier, and the system reached stability faster. The control sequences are illustrated in Figures 29 - 30, where the red shaded regions indicates when the slack variables got activated in the optimization problem. As seen in the figures the slack variables got activated quite frequently in the early phase of the simulation, helping the controller overcome the tight input limits during the transient phase. For the full closed-loop case with MHE estimates, minor slack activation persisted throughout the simulation. For the optimal case, the slack variables were only used briefly early on.

The magnitude of slack variable ϵ was also monitored, and most notable values included:

- $\epsilon \approx 2$ at startup
- Occasional peaks of $\epsilon \approx 1$ and $\epsilon \approx 0.5$
- Most frequent, lower activations of $\epsilon \approx 0.2 - 0.3$

These values indicate that while the violations did occur, they were relatively small for most of the simulation. The effective bounds remained stricter than in the no-slack case, showing that constraint softening was used efficiently rather than excessively.

For RMPC, similar results were observed, illustrated in Figures 31 - 33. Importantly, the RMPC activated even earlier than the MPC, and stabilized the system comparably well. The control sequences confirm this behavior. The slack variables in RMPC were activated mostly at the start, but with fewer and smaller violations than the MPC. However, the RMPC had somewhat more violations later in the simulation indicating that the robustness achieve stability faster, but the control action became more aggressive from the correction feedback term.

The magnitude of the slack variable ϵ were similar as to the MPC, and were:

- $\epsilon \approx 2$ at beginning
- Peaks around $\epsilon \approx 0.8$, once, and $\epsilon \approx 0.4$
- Occasional activations of $\epsilon \approx 0.2 - 0.3$

This indicates that the RMPC achieved comparable performance to MPC but with less reliance on slack activation early on. However, with somewhat more frequent activation later in the simulation to maintain stability.

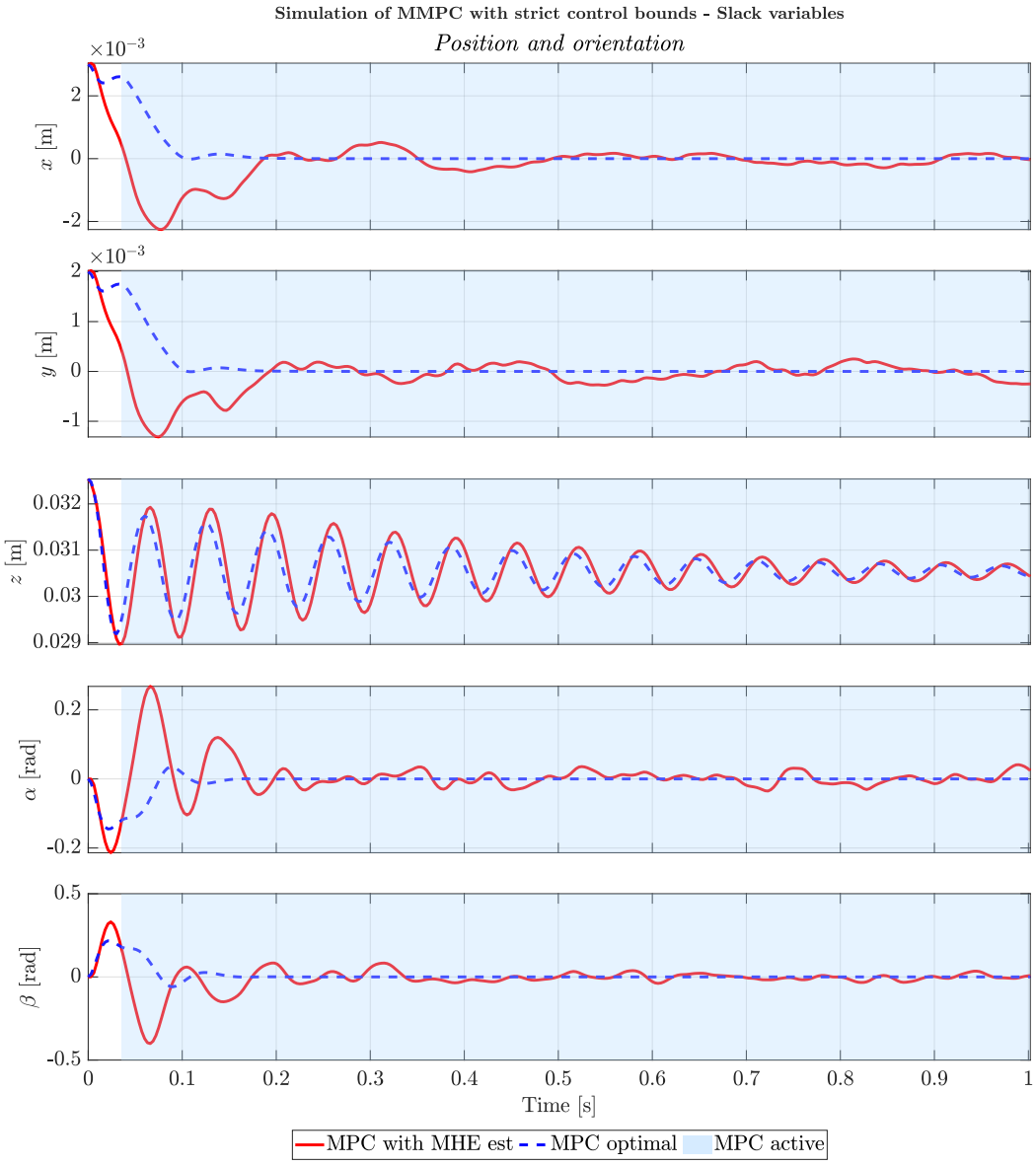


Figure 27: MPC simulation of position with and without MHE estimates with strict control bounds and slack variables active, see Table 6.

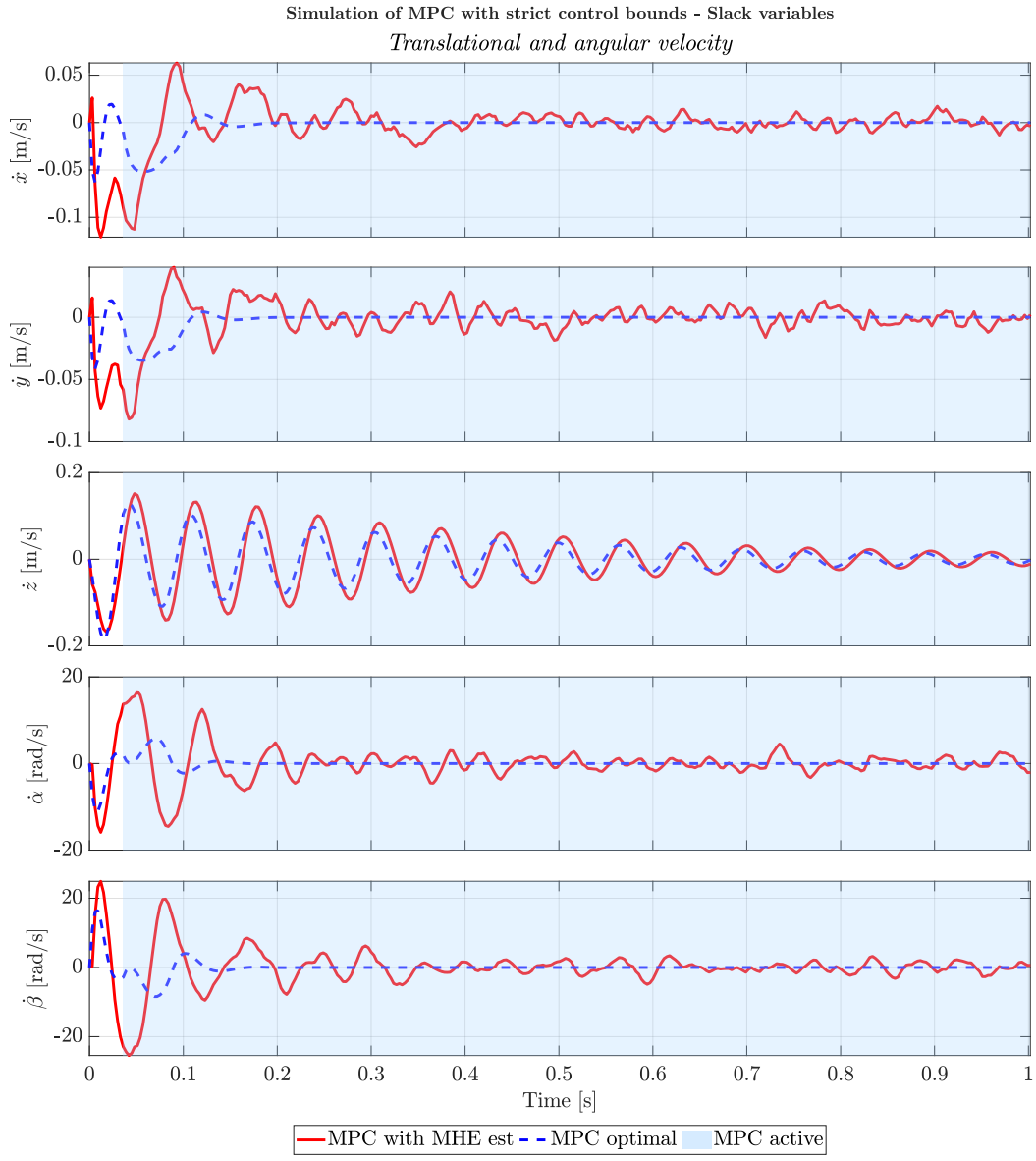


Figure 28: MPC simulation of velocity with and without MHE estimates with strict control bounds and slack variables active, see Table 6.

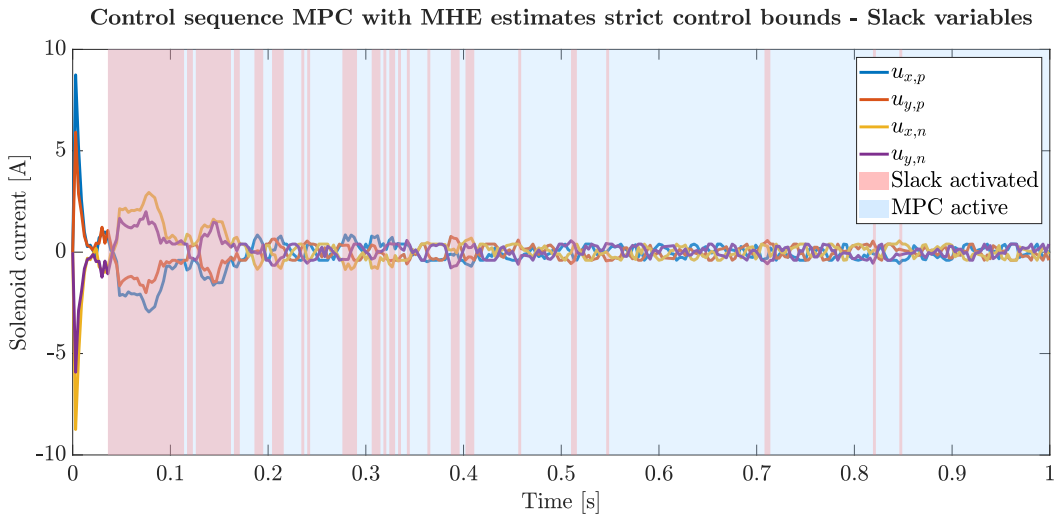


Figure 29: Control sequence for MPC with MHE estimates with strict control bounds and slack variables active, see Table 6.

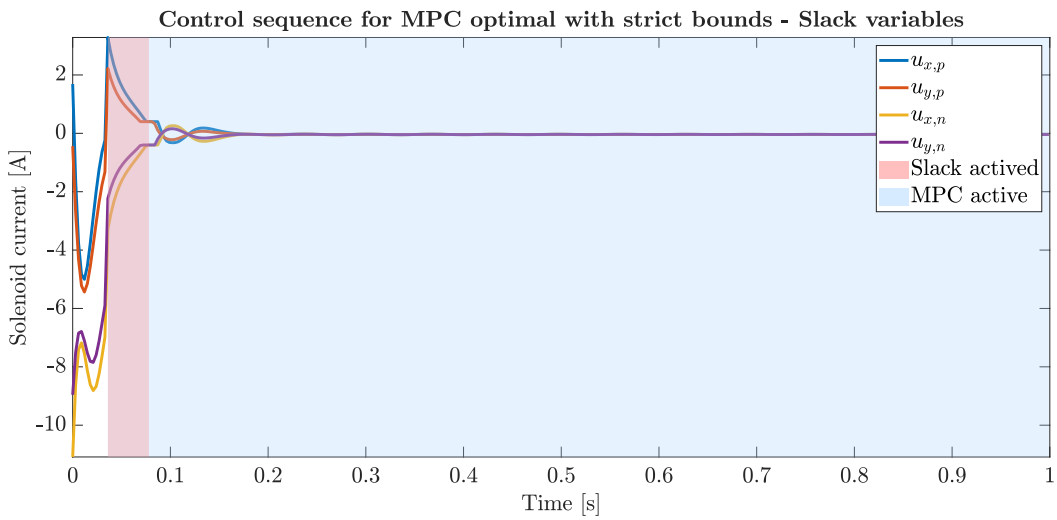


Figure 30: Control sequence for MPC optimal with strict control bounds and slack variables active, see Table 6.

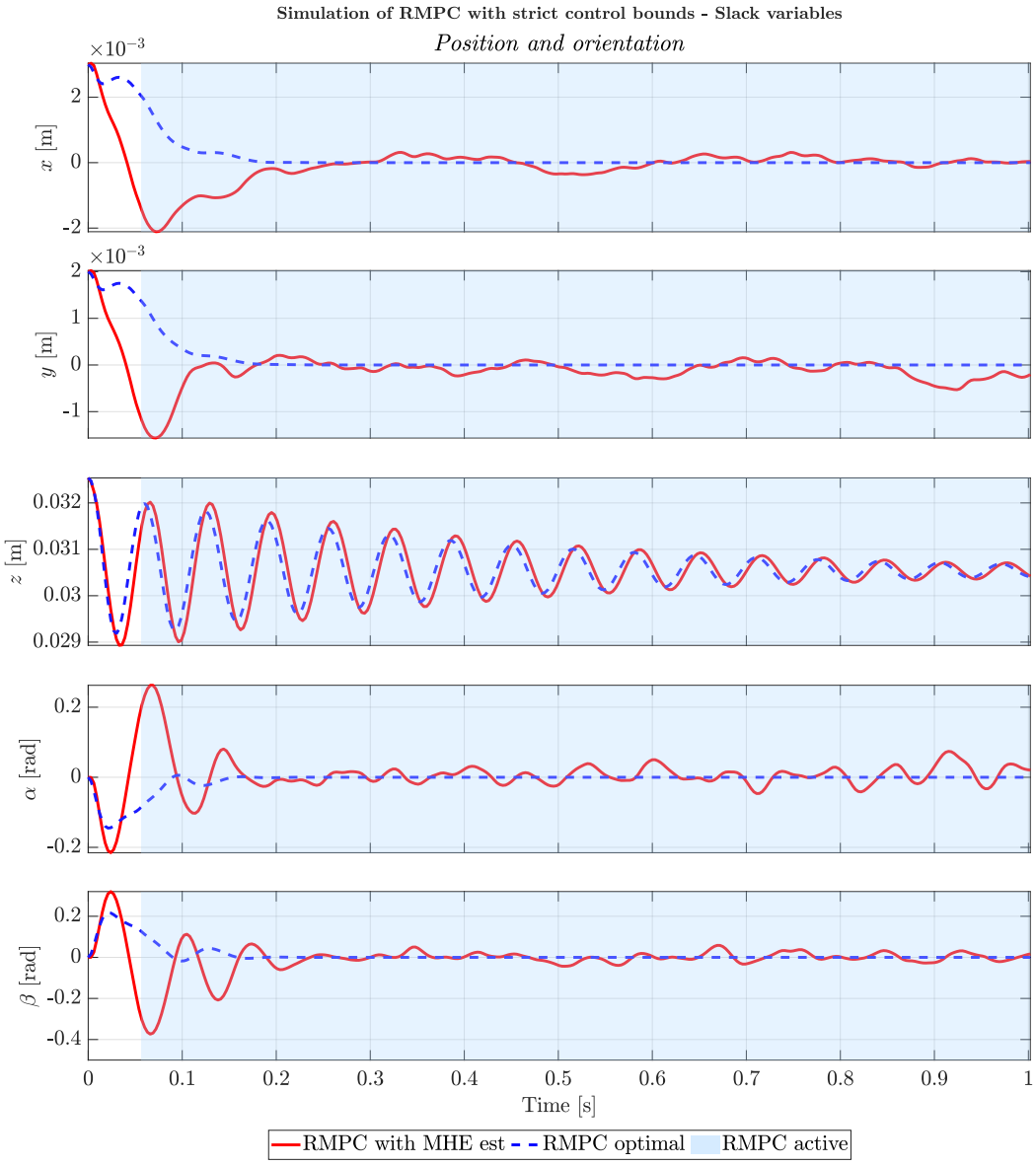


Figure 31: RMPC simulation of position with and without MHE estimates with strict control bounds and slack variables active, see Table 6.

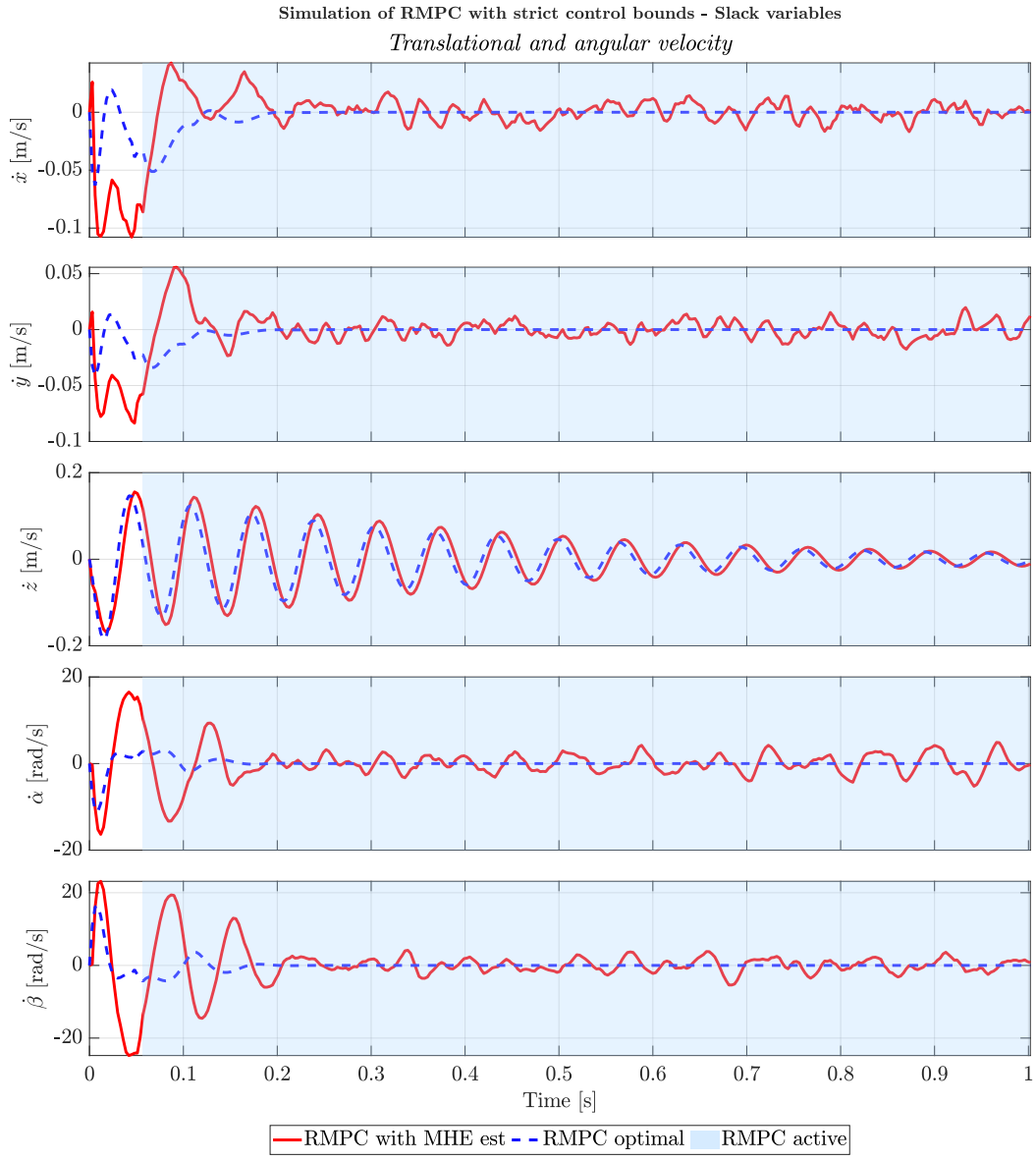


Figure 32: RMPC simulation of position with and without MHE estimates with strict control bounds and slack variables active, see Table 6.

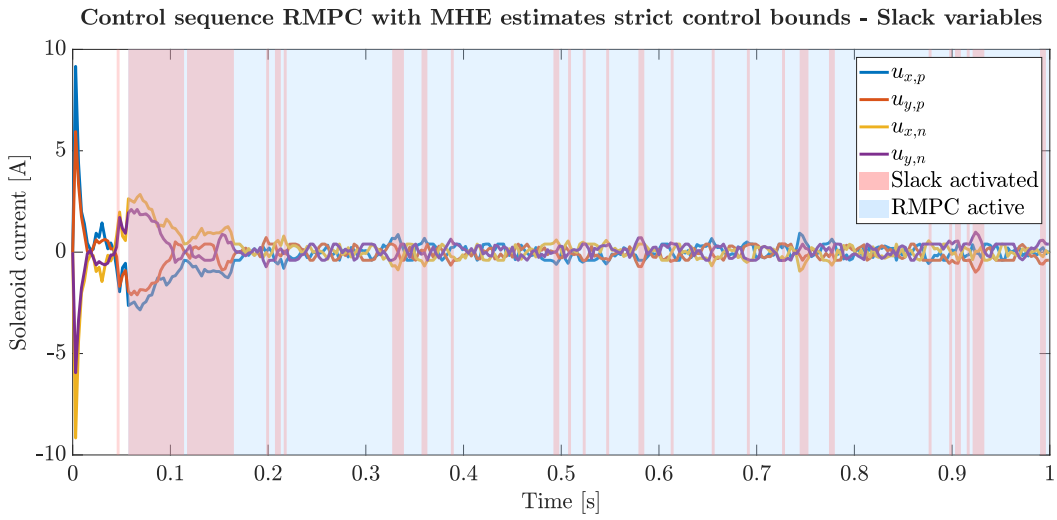


Figure 33: Control sequence for RMPC with MHE estimates with strict control bounds and slack variables active, see Table 6.

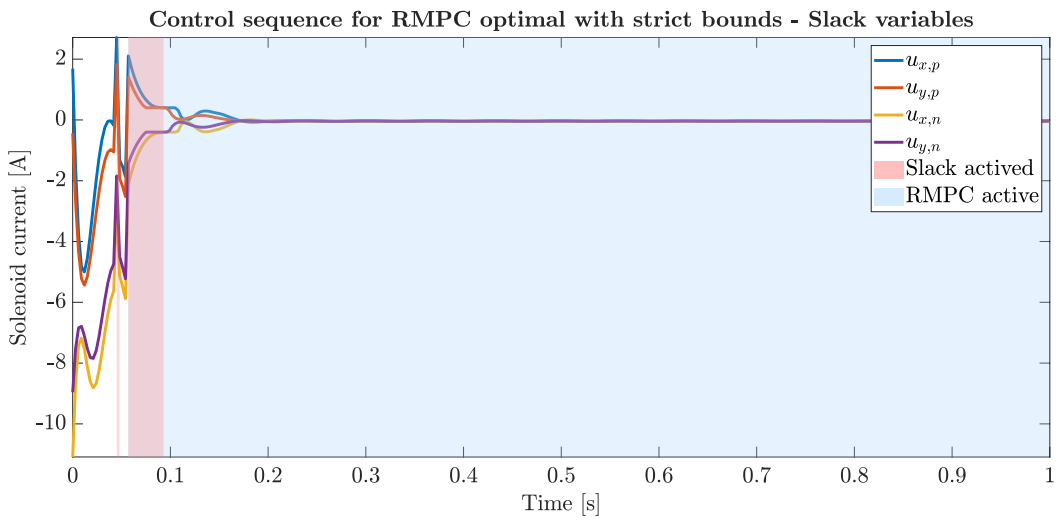


Figure 34: Control sequence for RMPC optimal with strict control bounds and slack variables active, see Table 6.

6.4 Experiment 4 - Reference Tracking

Table 7: Tuning Parameters for Experiment 4

Tuning Parameter	Value	Description
`N_MPC`	20	MPC Prediction horizon
`N_MHE`	15	MHE Prediction horizon
`dt`	0.003	Time Step
`t`	1	Simulation Time
`R_MPC`	$20 \cdot \mathbf{I}_4$	MPC Control weight
`Q_MPC`	$\text{diag}\begin{smallmatrix}(10^5, & 10^5, & 50, & 10, & 10, \ldots \\ 100, & 100, & 10, & 1, & 1)\end{smallmatrix}$	MPC State weights
`R_LQR`	$10^2 \cdot \mathbf{I}_4$	LQR Control weight
`Q_LQR`	$\text{diag}\begin{smallmatrix}(10, & 10, & 10, & 10, & 10, \ldots \\ 10, & 10, & 10, & 10^5, & 10)\end{smallmatrix}$	LQR State weights
`alpha`	0.9	EMA Constant
`noise_std`	10^{-4}	Magnetic sensor noise
`R_MHE`	$(\texttt{noise_std}^2 \cdot \mathbf{I}_9)^{-1}$	MHE Measurement noise weights
`Q_MHE`	$10^6 \cdot \text{diag}\begin{smallmatrix}(1, & 1, & 1, & 1, & 1, \ldots \\ 5, & 5, & 5, & 5, & 5)\end{smallmatrix}$	MHE Process noise weights
`Qscaling`	$5 \cdot 10^3$	Post-scaling for `Q_MHE`
`weightScaling`	10^{-4}	MHE Uniform scaling for weights
`X0`	$(-0.003,\ -0.002,\ \text{zeq} - 0.002,\ \mathbf{0}_{1\times 7})^\top$	Initial state
`x_ref`	$0.002 \cdot cos(6.3\,k)$	Circular reference for x
`y_ref`	$0.002 \cdot sin(6.3\,k)$	Circular reference for y
`xdot_ref`	$-0.002 \cdot 6.3 \cdot sin(6.3\,k)$	Circular reference for $\dot{x}$
`ydot_ref`	$0.002 \cdot 6.3 \cdot cos(6.3\,k)$	Circular reference for $\dot{y}$

The final experiment focused on assessing the ability of both MPC and RMPC controllers to follow a time-varying reference trajectory. The desired motion was a circular trajectory in the x-y plane, defined by sinusoidal references for x and y, and their respective velocities, as listed in Table 7. The goal was for the levitating magnet to trace a circular path around the z-axis while maintaining stability.

Figures 35 - 36 illustrate the position and velocity response of the MPC controller, both with and without MHE estimates. While the controller managed to follow the general shape of the reference, it exhibited noticeable tracking error, especially during the first half of the trajectory. With the MHE estimates the MPC had slightly degrading tracking performance, with the MHE introducing additional noise in the state estimates. The control sequences are illustrated in Figures 37 - 38 appeared stable, with no excessive or erratic actions observed.

In contrast, Figures 39 - 40 demonstrate that RMPC delivered significantly improved tracking. Both the estimated and optimal RMPC variants were able to follow the circular trajectory with high accuracy. The red and blue trajectories align closely with the reference in black trajectory, indicating a robust and consistent performance throughout the simulation. The RMPC also showed superior performance in transient behavior and faster convergence to the reference. Figures 41 - 42 show the corresponding control sequences for the RMPC, which again show the more aggressive control actions with RMPC. Overall, the RMPC showed better tracking ability than the MPC

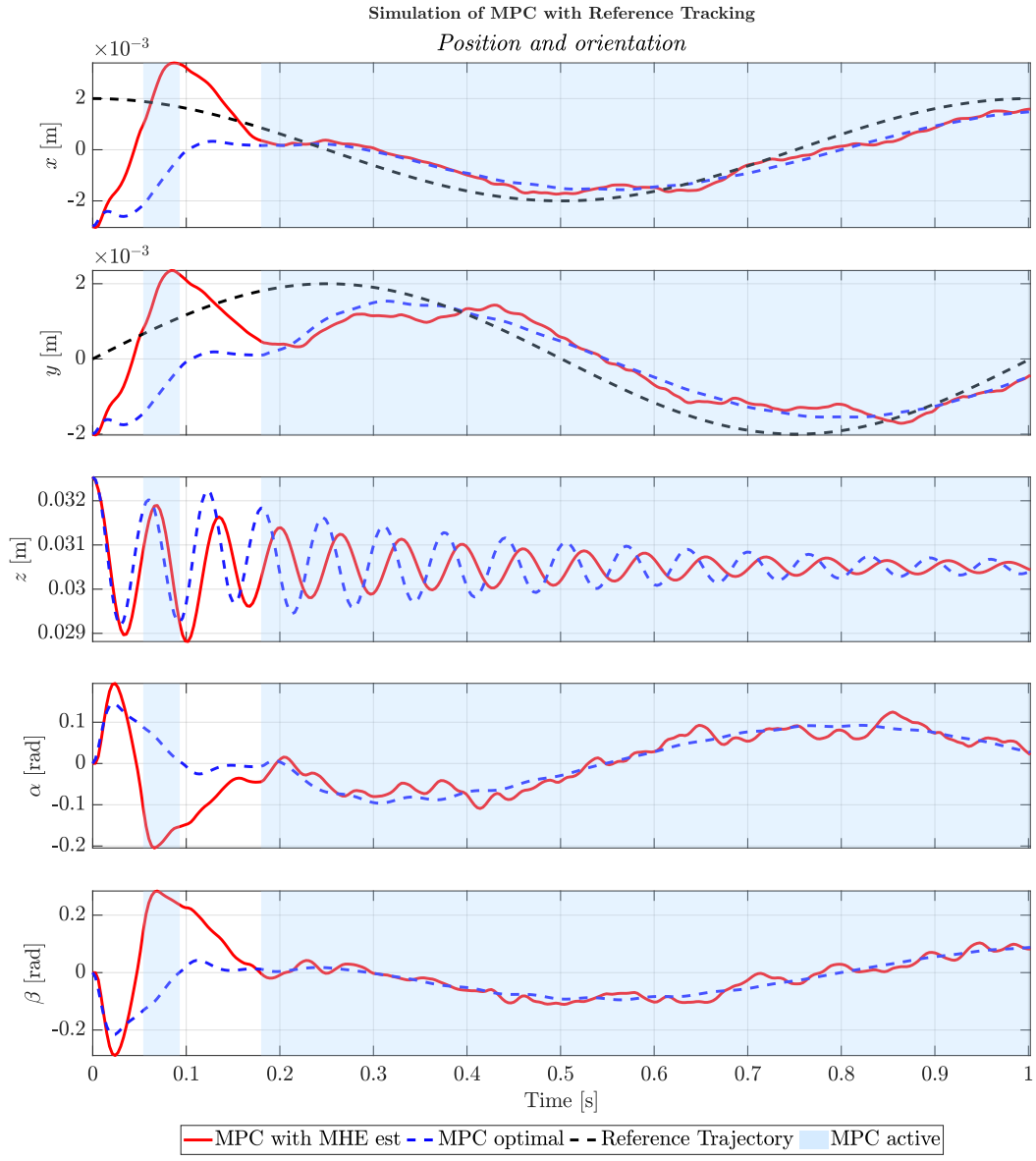


Figure 35: Simulation of position of MPC with and without MHE estimates, with reference trajectory.

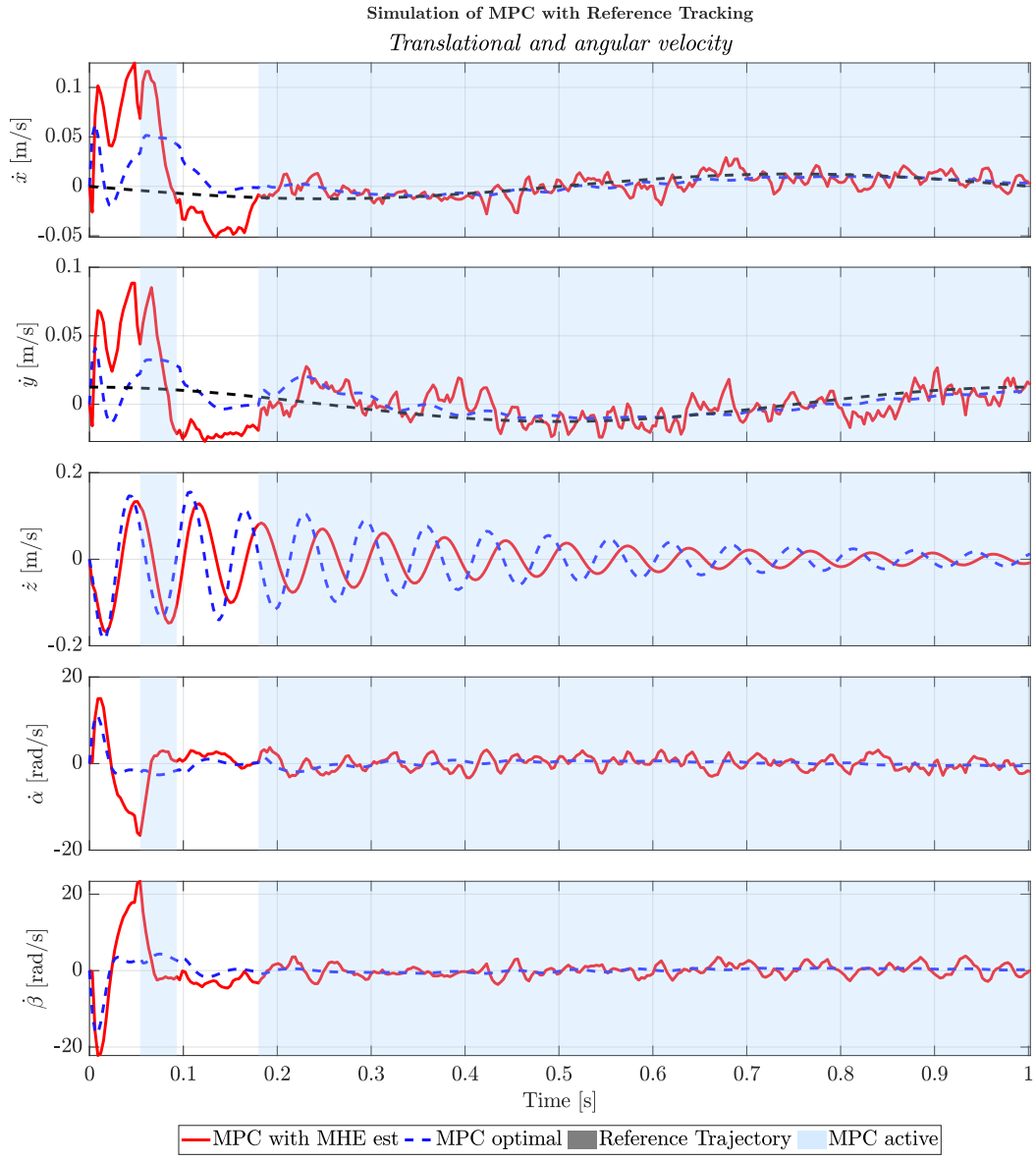


Figure 36: Simulation of velocity of MPC with and without MHE estimates, with reference trajectory.

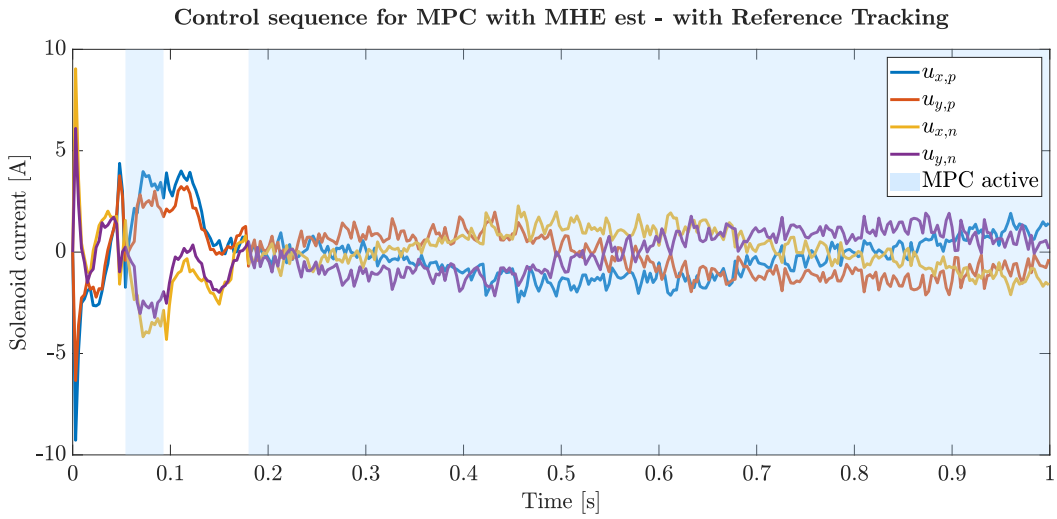


Figure 37: Control sequence for MPC with MHE estimates, with reference trajectory.

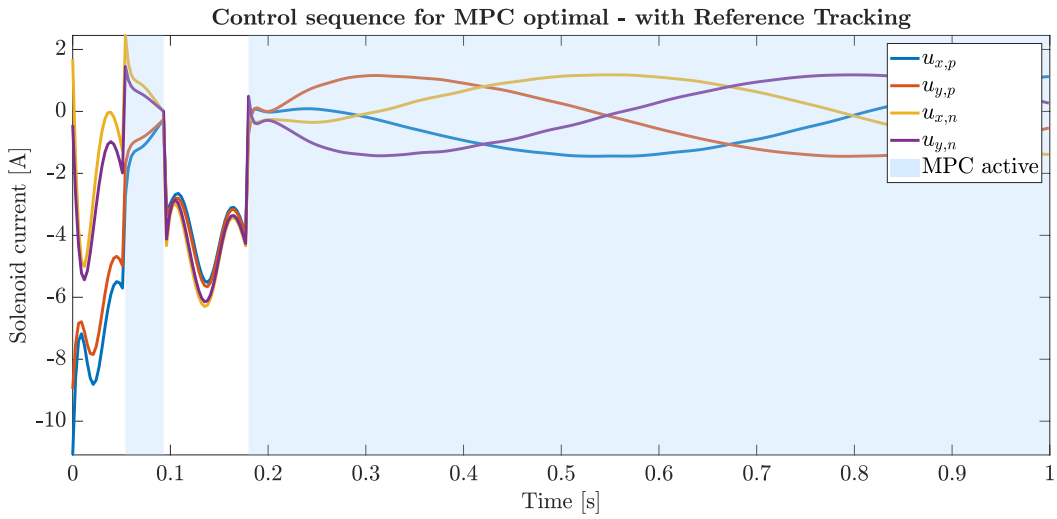


Figure 38: Control sequence for MPC optimal, with reference trajectory

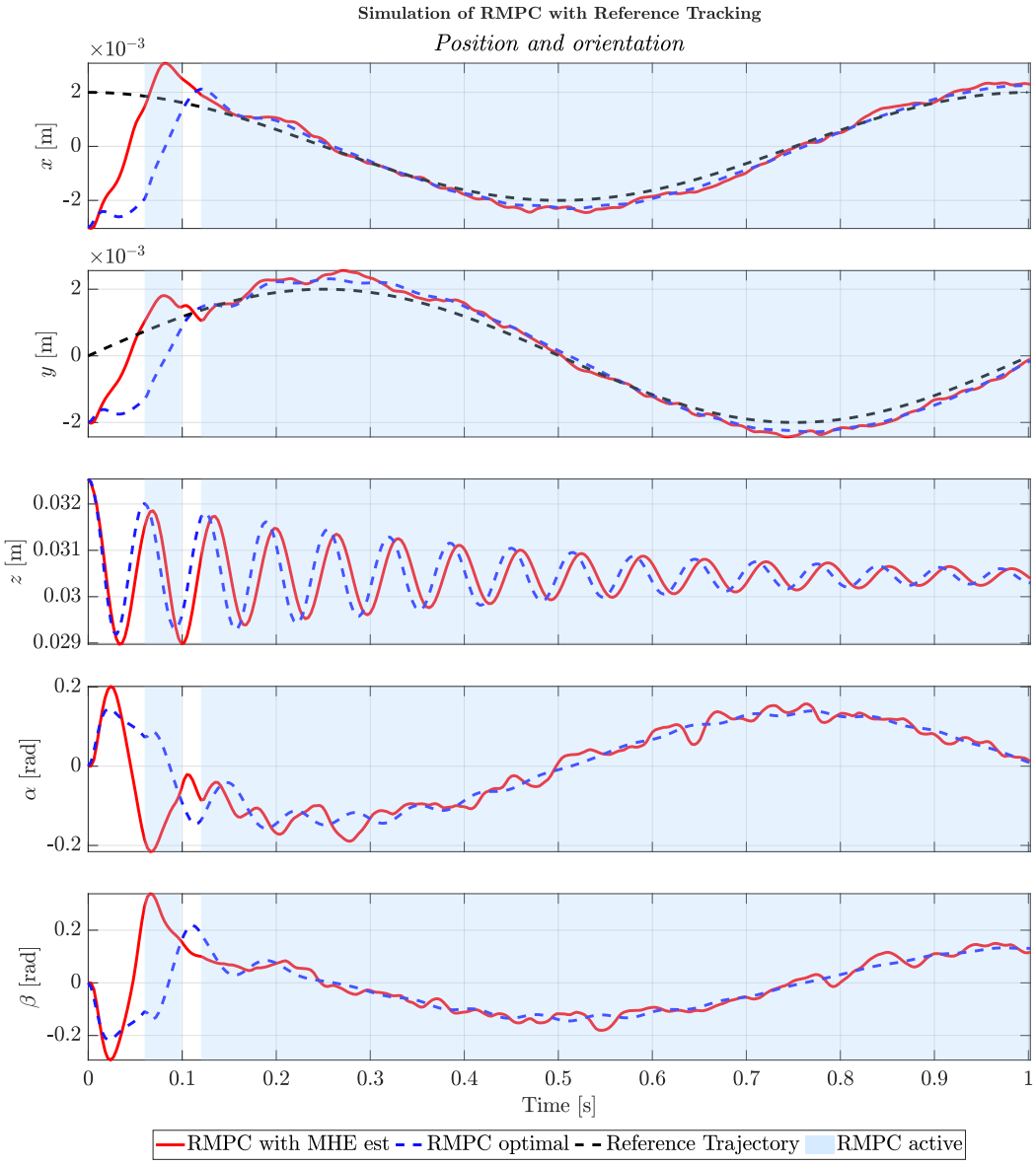


Figure 39: Simulation of position of RMPC with and without MHE estimates, with reference trajectory.

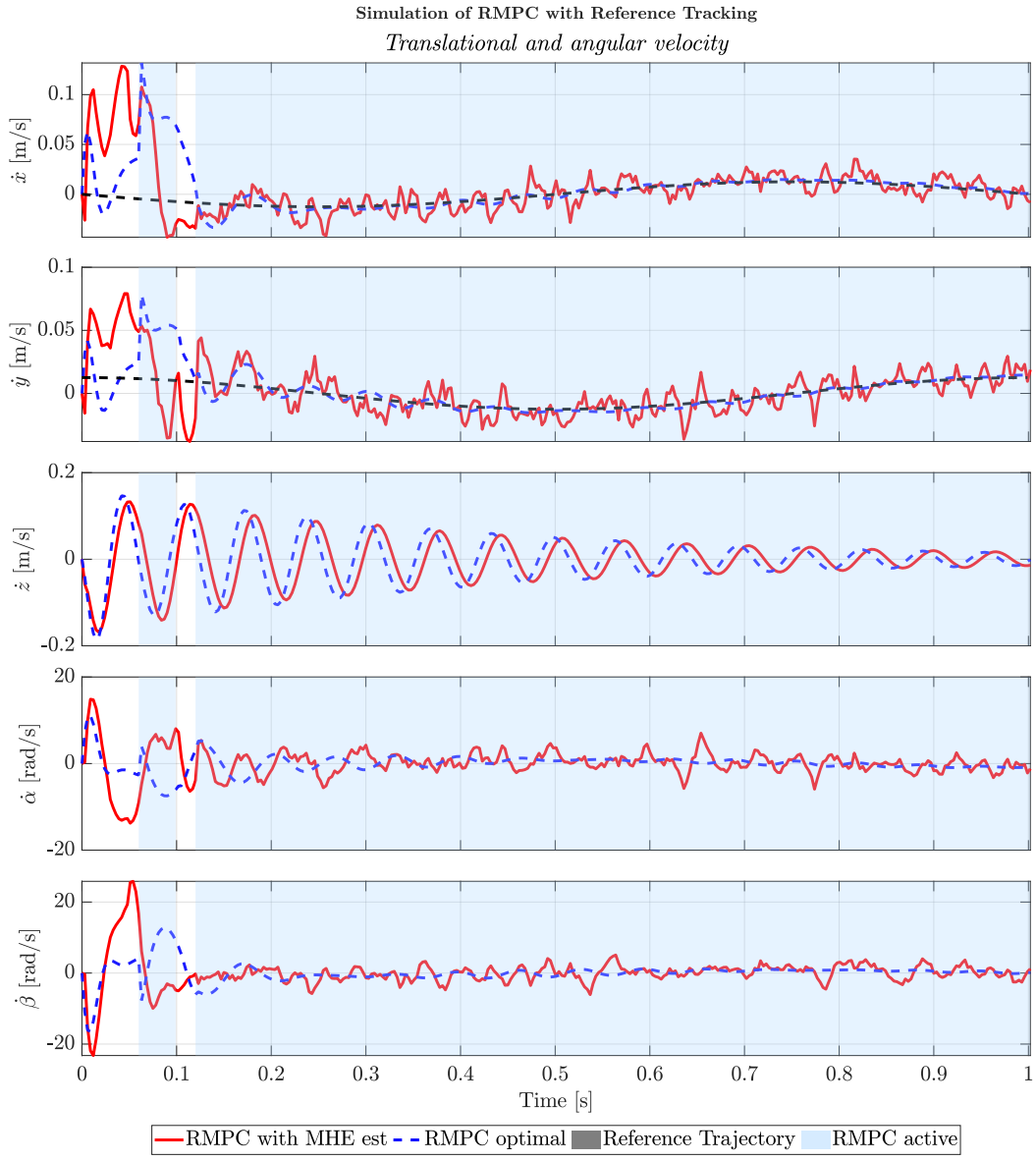


Figure 40: Simulation of velocity of RMPC with and without MHE estimates, with reference trajectory.

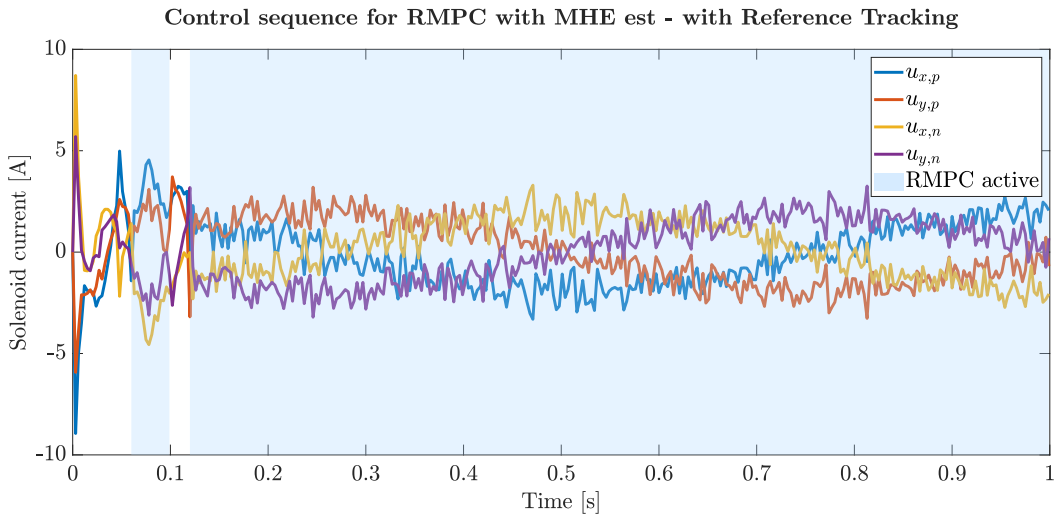


Figure 41: Control sequence for RMPC with MHE estimates, with reference trajectory.

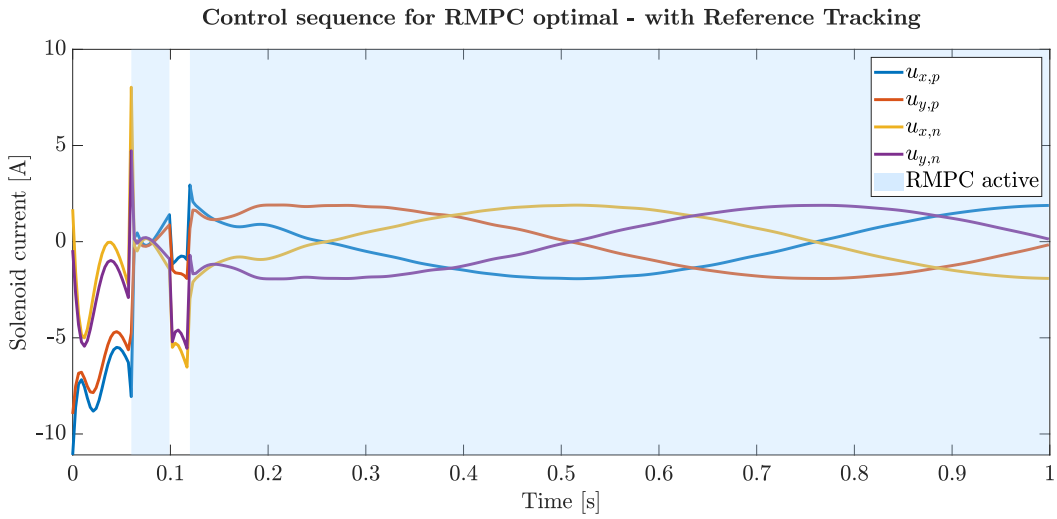


Figure 42: Control sequence for RMPC optimal, with reference trajectory

7 Discussion

This section will cover a discussion about the thesis, like the methodologies used throughout the project, the limitations and assumptions made, implementations, and a deep discussion of the results of experiments done. Lastly, a discussion about future work will be presented.

7.1 Methodology discussion

The methodology adopted in this thesis provided a robust foundation for exploring advanced control strategies for a magnetic levitation system. The use of the reduced-order model enabled manageable computational complexity while retaining key physical dynamics. Formulating the MPC as a QP problem allowed for efficient online optimization using MATLAB's built-in solvers. Moreover, the modular class-based code structure improved reusability and facilitated systematic testing of different controller configurations, including the RMPC and slack-variable-enhanced MPC.

Despite the success of the implemented control strategies, several limitations arise from necessary simplifications. The system was linearized only once around a single equilibrium point, which constrains the validity of the controller to small deviations. Although the RMPC implementation introduced robustness through the correction feedback term, the lack of tube-width computation means the RMPC lacks rigorous robustness guarantees. Similarly, slack variable formulations relied on carefully tuned penalty weights, which if poorly selected could lead to either infeasibility or overly relaxed control.

One notable design decision was to move away from the original nonlinear MPC formulation with CasADi in favor of a linear QP-based approach. This trade-off sacrificed some generality and model fidelity but significantly improved solver speed and allowed for easier debugging. The decision to use exact discretization further helped preserve the model's dynamics but introduces potential sensitivity to numerical instability as smaller time steps.

This thesis built on ground work done by the author during the project thesis [4]. The project thesis was based on getting familiar with the concept of MPC, and further investigate the different implementation methods, and different MPC configurations. During the project thesis, the author made a decision to further investigate the nonlinear MPC implementation with the use of CasADi toolbox. This was, as stated in Section 5, the original plan for the implementation of the full control-loop combination of MPC and MHE. However, because of limitations due to time and implementation complexity the decision fell on implementing the linear quadratic based MPC and MHE.

7.2 Result discussion

The results presented in Section 5.5 will be discussed during this section. The general results, comparison, challenges and improvements based on each test will be discussed.

7.2.1 Experiment 1 - Disturbance Rejection

The goal of this experiment was to evaluate how efficiently a RMPC can handle external disturbances in comparison to a standard MPC. To this end, a sudden pulse disturbance was applied during simulation, which emulates realistic scenarios such as actuator anomalies, external mechanical shocks, or rapid environmental changes. This is a well-established method for stress-testing control performance. A similar approach was used in the study *Robust Multiplexed Model Predictive Control* by researchers at Nanyang Technological University, where a pulse disturbance was applied to a mass-spring system to test the robustness of various RMPC schemes [40]. Their findings confirmed that RMPC strategies significantly outperformed standard MPC in mitigating the effects of such disturbances.

In our simulation results with `disturbance_1`, illustrated in Figures 7 - 8, both RMPC and MPC responded visibly to the disturbance. However, the RMPC maintained superior performance across most of the state variables. This was especially evident in the angular position, α and β, where RMPC limited the disturbance impact and returned to equilibrium more quickly than MPC. The MPC, by contrast, showed larger overshoots and longer settling times, indicating a slower and less stable recovery process. The improved behavior of RMPC can be attributed to its inherent design, which accounts for uncertainty and perturbations in the system model. Unlike standard MPC, which typically assumes an accurate model and limited disturbance, RMPC incorporates mechanisms, like the feedback correction term, to actively compensate for deviations. This aligns with the findings from the NTU study, where the RMPC was better equipped to handle the sudden change in input without destabilizing the system response.

Looking at Figures 9 - 10, the control input sequences further illustrate this difference. While both controllers react to the disturbance with sharp corrective inputs, RMPC shows slightly more noisy or aggressive actuation, a common trait in robust schemes aiming to quickly reject unexpected changes. Despite this, the RMPC remains within actuation bounds and stabilizes faster post-disturbance. This mirrors the NTU study, where RMPC responded more dynamically to pulse disturbances, trading off smoothness for faster correction and improved state regulation.

The second part of this experiment was aimed to evaluate the performance of RMPC and MPC under a more severe disturbance input, referred to as `disturbance_2` in Table 4, by increasing the strength of the solenoid pulse. The setup maintains consistency with the earlier low-disturbance test, allowing for direct comparison of the controllers' resilience to increasing disturbance magnitudes. In our results, illustrated in Figures 11 - 12, the increased disturbance led to a substantial difference in how each controller coped. Notably, the RMPC remained active and stable throughout the simulation, maintaining position control and recovering quickly after the pulse. In contrast, the MPC controller failed to stay active following the disturbance, as indicated by the grey deactivation period in the plot, where the control system fell back to the LQR fallback strategy. This controller dropout is critical, showing that the standard MPC combined with MHE was not robust enough to maintain feasibility and control under stronger disruptions, which was a limitation that the NTU study similarly highlighted when synchronous MPC was exposed to step-like disturbances [40].

The RMPC, by contrast, successfully handled the abrupt energy input, absorbing the effect without violating feasibility constraints or deactivating. This robustness can be attributed to RMPC's design, which explicitly considers uncertainty and disturbances by the feedback correction term. This improved disturbance rejection if further evident in Figure 12, where RMPC shows significantly quicker stabilization in the height velocity $\dot{z}$, and avoids the excessive overshoot and oscillations seen in the MPC trajectory just before failure. Once again, these results align with the NTU study, where RMPC demonstrated more reliable dynamic performance under pulse disturbances, especially in maintaining control authority without needing to switch fallback modes.

Figures 13 - 14 reinforce this distinction by showing the control sequences. In the RMPC case, Figure 13, all solenoid control inputs remain active and responsive, with visible reaction to the disturbance pulse. In contrast, the control effort in the MPC case, Figure 14, drops sharply following the pulse, indicating that the controller was deactivated due to infeasibility or constraint violation, a practical risk when dealing with unmodeled disturbances using standard MPC.

7.2.2 Experiment 2 - Model Mismatch

In this experiment, we explored the sensitivity of both the general MPC and RMPC to discrepancies between the controller model and the true system dynamics. Such model-plant mismatches are inevitable in practical control applications due to paramter uncertainty, unmodeled dynamics, or physical modifications to the system. To simulate this, we introduced changes to several model paramters, including increasing the mass and radius of the levitating magnet, enlarging the base magnet dimensions, and perturbing the spatial layout of the base magnet array, as shown in Table 5. These modifications aimed to test how each controller compensates for structural mismatches, particularly in early transient phase. The validity and relevance of such mismatch test is well-supported in the literature. In particular, Tufa and Kai systematically studied the impact

of model-plant mismatches on MPC performance and proposed a framework for determining mismatch thresholds that can compromise control performance [41]. Our approach aligns with this methodology, as we intentionally introduced controlled parametric deviations to evaluate how both MPC and RMPC handle real-world model inaccuracies.

Figures 15 - 16 show that both controllers, when combined with MHE, perform robustly despite the introduced mismatch. The transient behavior of the angular positions, α and β, reveals minor oscillations in the initial $0.1 - 0.2$ seconds, but both controllers stabilize rapidly. RMPC shows slightly better dampening in the early phase, which is consistent with its design. However, this performance gap closes quickly, and both controllers achieve similar steady-state behavior. The control inputs, illustrated in Figures 17 - 18, further support this observation. RMPC demonstrates slightly more aggressive corrective inputs immediately after activation, which reflects its robustness to uncertainty. Nevertheless, the control effort from both strategies converges after the transient period, and neither exhibits instability or sustained constraint violations.

These results corroborate the findings in [41], who noted that MPC can maintain satisfactory performance under moderate mismatch conditions, especially when paired with effective state estimation techniques. The fact that both RMPC and MPC with MHE performed comparably well in our experiment emphasizes the stabilizing role of MHE and the inherent robustness of MPC architectures, even without explicit robustness guarantees. In summary, the results confirm that both RMPC and MPC can tolarate moderate model mismatch without significant degradation in performance. RMPC offers marginally better transient control, but the overall similarity in outcomes suggests that MHE plays a crucial role in mitigating the effects of model uncertainty.

7.2.3 Experiment 3 - Constraint Handling

Firstly, a discussion of the constraints handling experiment without slack variables will be done. We discuss the results in the order for which the results where represented in Section 6.3.

The results from the first part in experiment 3, illustrated in Figures 19 - 26, illustrate the performance of MPC and RMPC with and without the MHE under strict control input bounds, ass shown in Table 6. These bounds represent the tightest limits for which the system could remain stable without the need for constraint relaxation. In Figures 19 - 20, the positions and their respective velocities are shown for the standard MPC, both with and without the MHE estimates. From the figures it is evident that both the MPC was able to stabilize the magnet in both cases. However, there was a noticeable delay before MPC took over from LQR, which was active initially. The LQR applied high control efforts, shown in Figures 21 - 22, to drive the system quickly toward the equilibrium, helping bring the states closer to their setpoints before the MPC became active. This handover from LQR to MPC is marked by the blue shaded region in the figures. Despite the strict limits, the system managed to remain stable, which highlights the effectiveness of the parameters and the robustness of the LQR-MPC transition. It also shows that this bound set, from Table 6, represents a critical threshold tight enough to limit control effort significantly, but still feasible for successful regulation.

Including the MHE in the MPC loop led to a more dynamic response in the early phase. The MPC with MHE shows larger overshoots and higher oscillatory behavior than the MPC on optimal states. This can be attributed to the transient estimation errors at the beginning of the simulation, where MHE is still converging to an accurate estimate of the states. Once estimation stabilizes, the two MPC variants behave similarly, with near-identical convergence and tracking accuracy. Figures 21 - 22 reveal differences in the control effort between MPC with optimal states and with MHE estimates. In the MHE case the control inputs show more aggressive early behavior, reflecting both the estimation uncertainty and the LQR's stronger effort to compensate for initial deviations. In contrast, the optimal MPC shows smoother and more contained inputs before MPC activation, indicating a more stable transition from LQR to MPC. Interestingly, both control sequences settle quickly after MPC takes control, with all solenoid currents converging near-zero-state values. This implies efficient regulation and minimal control activity after initial transients.

The second part of the experiment evaluates the performance of RMPC with and without MHE

under the same strict control input bounds as the MPC case. Figures 23 - 26 provide insight into the system's state evolution and control effort during the transition from LQR to RMPC. A notable distinction from the MPC case is that RMPC is activated earlier. The blue shaded region shows the RMPC activation, which begins significantly earlier than in the MPC case. Despite operating under the same tight input bounds, RMPC is able to take over the control task more quickly. This suggest that the robust formulation of RMPC, with the feedback correction term, improves feasibility and ensures earlier compliance with control objects compared to standard MPC. Both cases for the RMPC successfully stabilize the system across all states. However, RMPC with MHE again display larger initial transients compared to the RMPC on optimal states. This effect is noticeable in the angular and vertical dynamics, and their respective velocities, where the MHE-based RMPC introduces higher oscillations during the estimation convergence phase. Despite this, both cases for RMPC eventually settle into similar behavior, showing that the robustness enable consistent performance once initial uncertainties diminish. Interestingly, the RMPC with MHE appear more aggressive early on, especially in velocities. This could be a direct consequence of the estimator reacting strongly to initial measurement noise and attempting to counteract the perceived uncertainty.

Figures 25 - 26 show the solenoid currents applied by the RMPC. Similar to the MPC case, the LQR applies high control actions before RMPC is activated. Once the RMPC takes control, the currents settle more quickly in comparison to the MPC control sequences. The faster convergence of the control signals and the earlier controller takeover indicate that the RMPC is more capable of respecting strict input limits while ensuring feasibility and regulation performance. In both RMPC variants, control inputs stay within the strict bounds, confirming the feasibility of the formulation. The MHE-based RMPC produces slightly higher early peaks in the control signals, especially in the positive solenoids, likely reflecting the system's attempt to reject initial state uncertainty.

The next part of this experiment introduced the use of slack variables. The result of this experiment demonstrated the practical utility of slack variables in maintaining stability and feasibility in MPC under strict control input bounds. A study conducted by Zeilinger et al. [42], propose a soft-constrained MPC scheme that guarantees recursive feasibility and robust stability, even when hard constraints would lead to infeasibility. Their approach penalizes constrain violations using slack variables with quadratic penalties, a method directly analogous to the one used in this experiment, where slack variables were added to allow for controlled constraint violations when inputs fell outside tight bounds. The bounds for this part of the experiment were tightened compared to the first part, and are listed as the number 2 bounds in Table 6.

Figures 27 - 30 show the result of the MPC case both with and without MHE combined. Figures 27 - 28 show that the MPC in both cases successfully stabilize the system within 1 second, despite strict bounds. This aligns with Zeilinger et. al.'s findings that constraint violations can be strategically allowed to preserve stability, provided the violation is appropriately penalized. In our case, this penalty was implemented through a combination of linear ρ and quadratic $\mathbf{S}$ terms. One quick notice was the difference in activation time for the MPC, in comparison to the case without the slack variables. By letting the controllers violating the bounds to some degree, it let the controller take over from the LQR earlier than without the slack variables. Additionally, our results confirm that slack variables primarily activate during the initial transient phase, illustrated in Figures 29 - 30, when the system is farthest from the equilibrium and the control effort is most constrained. This mirrors Zeilinger et. al.'s observations that constraint violations tend to be most relevant during aggressive transients, after which the system naturally returns to operation within the bounds.

An important validation point is the magnitude and frequency of slack usage. While Zeilinger et al. emphasize that violations should be minimal and decay over time for stability guarantees, our experiment also observed that the largest violations, $\epsilon \approx 2$, occurred at the start and quickly diminished to values around $0.2 - 0.3$. This suggest that the controller consistently prioritized remaining within bounds when possible, using slack variables only when necessary, a behavior consistent with MPC principles. Furthermore, the use of MHE improved responsiveness and estimation accuracy, leading to slightly faster convergence and more proactive control.

The RMPC simulations with slack variables under strict input constraints, as shown in Figures

31 - 33, reinforce and extend the observations made in the standard MPC case. Similar to MPC, the RMPC formulation benefit significantly from the integration of slack variables, particularly during the early transient phase when the system is far from equilibrium. Across all position and orientation states, illustrated in Figure 31 both with and without MHE estimation, demonstrates fast and stable convergence. Another important notice, as for the MPC case, is that with slack variables the RMPC actives significantly earlier than for the RMPC case without slack variables. This earlier activation supports improved transient performance and quicker stabilization. Oscillations in angular states are efficiently damped, a little earlier than for the MPC case, which confirms the RMPC's robustness combined with the real-time estimation through MHE. The corresponding velocities, illustrated in Figure 32, again reveal that RMPC with MHE is more responsive during the early phase. Peaks in velocities are slightly more pronounced when using estimates, suggesting a more aggressive control action, yet still within acceptable bounds. Over time, the velocities settle consistently across both configurations, indicating that slacks enabled in RMPC ensures stability despite the tight constraints.

Perhaps the most telling insights come from the control sequences illustrated in Figures 33 - 34. The control sequence of RMPC with MHE show slacks activated more frequently during the first phase. These activations are spread across different inputs and tend to coincide with large control efforts required to suppress transients. Importantly, the magnitude of the slack variables reduces steadily over time, with values ranging similar to the MPC case. For the RMPC case without the MHE estimates the slack variables are only activated early on in the simulation, similar to the MPC case. Comparing the results for the MPC case and RMPC case, it shows improved stability and robustness under strict control bounds for the RMPC case. This can be attributed to the RMPC feedback correction formulation. The slack variable activation pattern show fewer violations throughout the simulation for RMPC as well as shorter violation periods. With similar values for ϵ this indicates that the RMPC has less strict bounds violation in general.

7.2.4 Experiment 4 - Reference Tracking

In the last experiment we experimented on the MPC and RMPC's ability to track a given reference trajectory, more precisely a circular pattern. The two states x and y has a circle reference trajectory simulating the magnet moving in a circle while stable. The results from this experiment demonstrated the effectiveness of both controller versions, both with and without MHE estimates, in guiding a predefined circular trajectory. Across the different simulations, the performance differences between MPC and RMPC became evident, especially when subjected to real-time state estimation noise from the MHE.

For the standard MPC controller, illustrated in Figures 35 - 36, both the positions and velocities was able to follow the given reference trajectory to a certain degree. However, with notable deviations, particularly in the early transient phases and in the steady-state performances. The x and y states occasionally drifted from the reference, both with and without MHE. The MHE also included measurement noise which visibly affected the velocities. The control sequences for the MPC case, illustrated in Figures 37 - 38, had no notable results.

When comparing these findings with the RMPC implementation, illustrated in Figures 39 - 40, a clearer advantage emerged. The RMPC achieved significantly more precise tracking of the circular reference trajectory, both with and without MHE estimation. Both the states and velocities followed the reference trajectory with minimal deviation, and the system exhibited better transient performance and steady-state accuracy. The formulation of the feedback correction in the RMPC with track reference clearly helped the MPC track the circular reference to a significantly better accuracy than without the RMPC term, both with and without the MHE estimates. This outcome supports conclusions found in the study on path tracking of tracked vehicle towing a steerable trailer, where a robust MPC design was successfully used to manage external disturbances and uncertainty, demonstrating more reliable tracking behavior under such conditions [43]. Moreover, the RMPC's ability to maintain performance despite the added noise from the MHE validates its practical relevance. Just as in the previously mentioned experimental study, incorporating robustness into the MPC formulation mitigates the impact of state estimation imperfections, which are inevitable in real-worlds systems.

In summary, the RMPC clearly outperformed the MPC in the case of reference tracking. However, one notable observation, which is relevant for all experiments, was the more aggressive control sequence compared to the MPC. This indicates that the RMPC achieved better performance and stability in some uncertain conditions, but at the cost of smoother control actions.

7.2.5 Improvements for the experiments

The experiments done gave promising results for the two different MPC configurations, and we obtained a clearer view of the advantages with a more robust implementation. The four experiments done were all constructed to test different parts of the implementations, though there could have been some improvements or changes in the experiments to obtain an even better result. The disturbance rejection experiment was conducted by adding a sudden pulse input as a disturbance. The disturbance input were given a constant value, and with a idealized timing. Real-world disturbances are often more stochastic or persistent. What could have been done differently would be to introduce a wider range of disturbance profiles to assess controller robustness under more realistic conditions.

The second experiment, model mismatch, was conducted by changing some parameters in the plant model for the MPC. However, as the test results showed, there were no clear reaction from either controller by this change. There could have been done a deeper experiment with the model mismatch, like changing the structure, or use unmodeled dynamics etc. Another improvement would be to test with model mismatch in the MHE as well to see if the full closed-loop with model mismatch in the plant would react differently then in the experiment during this thesis.

The experiment for the constraint handling, with an introduction of slack variables, was used for a specific test. Where we only wanted to see how strict the control bounds could be before we experienced infeasibility. However, a more comprehensive exploration could reveal when and how slack variables truly help. We could rather have performed a sensitivity analysis on the slack penalty weights and included infeasibility metrics or trade-off curves.

The reference tracking experiment was conducted by making the MPC track a circular pattern, which means making the levitating magnet move in a circle, while still maintaining stability. With such a reference trajectory, the x and y position and velocity had a reference. To improve this experiment, we could have made a harder multi-axis reference with high curvature or rate of change.

7.3 Future work

Several promising directions can be pursued to extend the work presented in this thesis and further improve the performance, robustness, and practical applicability of the control system.

The most immediate next step would be to deploy the current MPC and MHE implementation on the real physical maglev system. All development and experiments conducted in this thesis have been purely simulation-based. Although the simulation results have demonstrated promising behavior, real-world systems often involved unmodeled dynamics, delays, and noise characteristics that cannot be fully captured in simulation. Testing the full control loop on hardware would provide valuable insight into practical implementation challenges and validate the controller under physical constraints and sensor feedback.

A natural progression beyond this is to reintroduce a nonlinear control implementation, which was originally planned for this thesis. Currently, both MPC and MHE were implemented based on linearized dynamics around a equilibrium. While coputationally efficient, this approach limits accuracy and constraint satisfaction in regions farther from the linearization. Future work should formulate both MPC and MHE as NLP's using tools like CasADi. This would allow the full nonlinear dynamics to be retained in the optimization, thereby improving tracking fidelity and performance under larger deviations.

For the nonlinear MPC, real-time feasible schemes such as SQP, especially the Real-Time Iteration (RTI) scheme, should be considered. These methods have shown great promise in achieving fast

convergence for nonlinear control tasks, particularly when warm-started effectively. The foundational work by Diehl et al. [10] provides the theoretical basis for such implementations and justifies their use in fast embedded optimization. Another suggestion is to use interior-point solvers for the NLP formulation, like Ipopt[accessed June 2025] or the recently developed and faster Fatrop[accessed June 2025] [44].

Another important extension is to enhance the RMPC formulation. The current RMPC approach does not include tube-width estimation, which is essential for formal robustness guarantees. By incorporating online or offline estimation of the uncertainty tube future implementations could tighten constraints more effectively and reduce the conservatism in the robust controller. Methods for adaptive tube sizing and constraint tightening could improve feasibility and performance in highly dynamic environments.

8 Conclusion

This thesis has explored the implementation of MPC for a magnetic levitation system, with particular focus on the impact of robustness in practical control applications. By developing and evaluating both a standard linear MPC and a RMPC, the study aimed to assess how these advanced control strategies perform under real-world conditions such as disturbances, model mismatches, constraints handling, and reference tracking.

The results demonstrated that the standard MPC offers acceptable control performance under ideal conditions in full closed-loop combined with MHE. However, it could degrade significantly in the presence of disturbances. In contrast, the RMPC showed improved resilience in scenarios involving disturbance and reference tracking. These findings highlight the value of robustness in control system design, particular for safety-critical systems like maglev where uncertainties are unavoidable. However, due to the correction feedback term the control action for the RMPC appeared more aggressive.

While the added complexity of RMPC may not always be justified for simple applications, this study indicates that it brings an advantage in robustness and reliability when operating under less predictable conditions. The combination with the MHE in the closed-loop architecture proved to be a valuable combination, enabling tighter control.

Due to time constraints, the scope was limited to linear, time-invariant models near the equilibrium. Although this allowed for efficient implementation using QP in MATLAB, future work could explore nonlinear MPC formulations to increase accuracy and generality, especially in regions further from operating point.

Overall, this thesis provided practical insight into the benefits of robustness in MPC in a real-time maglev application, and contributed to the ongoing development of a take-home lab platform for education and research in control theory. The project also offered valuable experience in advance control theory, algorithm development, and academic research methodology.

Bibliography

[1] Z. Qadir, A. Munir, T. Ashfaq, H. S. Munawar, M. A. Khan, and K. Le, "A prototype of an energy-efficient maglev train: A step towards cleaner train transport", *Cleaner Engineering and Technology*, vol. 4, p. 100 217, 2021, ISSN: 2666-7908. DOI: 10.1016/j.clet.2021.100217.

[2] P. I. D. K. Fosmo, S. Graffer, and S. A. Senkaya, *MAGGY: Hands-On Control Learning with a Maglev System*. NTNU, 2024.

[3] H. A. Engmark and K. T. Hoang, "Modeling and control of a magnetic levitation platform", *IFAC-PapersOnLine*, vol. 56, pp. 7276–7281, 2023. DOI: 10.1016/j.ifacol.2023.10.338.

[4] M. Mjelde, "Ttk4551 specialization project - model predictive control for magnetic levitation system", Norwegian University of Science and Technology, Tech. Rep., 2024.

[5] H.-W. Lee, K.-C. Kim, and J. Lee, "Review of maglev train technologies", *IEEE Transactions on Magnetics*, vol. 42, pp. 1917–1925, Aug. 2006. DOI: 10.1109/TMAG.2006.875842.

[6] J. Richalet, A. Rault, J. L. Testud, and J. Papon, "Model predictive heuristic control: Applications to industrial processes", *Automatica*, vol. 14, no. 5, pp. 413–428, 1978, ISSN: 0005-1098. DOI: 10.1016/0005-1098(78)90001-8.

[7] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey", *Automatica*, vol. 25, no. 3, pp. 335–348, 1989, ISSN: 0005-1098. DOI: 10.1016/0005-1098(89)90002-2.

[8] M. Morari and J. H. Lee, "Model predictive control: Past, present and future", *Computers & Chemical Engineering*, vol. 23, no. 4, pp. 667–682, 1999, ISSN: 0098-1354. DOI: 10.1016/S0098-1354(98)00301-9.

[9] C. R. Cutler and B. L. Ramaker, "Dynamic matrix control—a computer control algorithm", in *Proc. Joint Automatic Control Conf. (JACC)*, San Francisco, CA, 1980, WP5–B.

[10] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. K. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations", *Journal of Process Control*, 2002, ISSN: 0959-1524. DOI: 10.1016/S0959-1524(01)00023-3.

[11] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation", in *Lecture Notes in Control and Information Sciences*, L. Magni, D. M. Raimondo, and F. Allgöwer, Eds., vol. 384, Berlin, Germany: Springer, 2009, pp. 391–417. DOI: 10.1007/978-3-642-01094-1_32.

[12] A. Zhakatayev, B. Rakhim, O. Adiyatov, A. Baimyshev, and H. A. Varol, "Successive linearization based model predictive control of variable stiffness actuated robots", in *Proceedings of the IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, Munich, Germany, Jul. 2017, pp. 1774–1779. DOI: 10.1109/AIM.2017.8014275.

[13] M. Gulan, M. Salaj, M. Abdollahpouri, and B. Rohaľ-Ilkiv, "Real-time mhe-based nonlinear mpc of a pendubot system", *IFAC-PapersOnLine*, vol. 48, no. 1, pp. 1059–1064, 2015. DOI: 10.1016/j.ifacol.2015.05.199.

[14] M. W. Spong and D. J. Block, "The pendubot: A mechatronic system for control research and education", in *Proc. 34th IEEE Conf. Decision and Control (CDC)*, vol. 1, 1995, pp. 555–556. DOI: 10.1109/CDC.1995.478951.

[15] B. Houska, J. Ferreau, and M. Diehl, "Acado toolkit: An open source framework for automatic control and dynamic optimization", *Optimal Control Applications and Methods*, vol. 32, pp. 298–312, May 2011. DOI: 10.1002/oca.939.

[16] M. Fink, *Implementation of linear model predictive control – tutorial*, arXiv preprint arXiv:2109.11986, arXiv:2109.11986 [eess.SY], 2021.

[17] F. Lorenzetti and M. Pavone, "A simple and efficient tube-based robust output feedback model predictive control scheme", in *Proc. European Control Conference (ECC)*, Sections III.A & IV. Available at https://stanfordasl.github.io/wp-content/papercite-data/pdf/Lorenzetti.Pavone.ECC20.pdf, IEEE, 2020, pp. 1724–1731.

[18] B. Foss and T. A. N. Heirung, *Merging Optimization and Control*. NTNU, 2016, Technical Report, ISBN: 978-82-7842-201-4.

[19] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd. New York, NY, USA: Springer, 2006.

[20] R. Bellman, "The theory of dynamic programming", *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954. DOI: 10.1090/S0002-9904-1954-09848-8.

[21] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd. Santa Barbara, CA: Nob Hill Publishing, 2024, ISBN: 978-0-9722877-2-6.

[22] L. Dai, M. Fu, and M. S. Mahmoud, "Discrete-time model predictive control", *Advances in Discrete Time Systems*, Dec. 2012. DOI: 10.5772/51122.

[23] J. Löfberg, "Linear model predictive control", Department of Electrical Engineering, Linköping University, Tech. Rep., 2001, Available online: https://www.researchgate.net/publication/228574166_Linear_Model_Predictive_Control [Accessed: 2025-06-02].

[24] M. Mehrez, *Mpc and mhe implementation in matlab using casadi*, https://github.com/MMehrez/MPC-and-MHE-implementation-in-MATLAB-using-Casadi/tree/master/workshop_github, Accessed: 2025-06-02, 2022.

[25] M. Johansson, *El2700: Model predictive control*, Lecture Notes, KTH Royal Institute of Technology, Available at: https://people.kth.se/~mikaelj/documents/EL2700_Lecture_Notes_20240311.pdf, 2024.

[26] D. Q. Mayne, M. M. Seron, and S. V. Rakovic, "Robust model predictive control of constrained linear systems with bounded disturbances", *Automatica*, vol. 41, no. 2, pp. 219–224, 2005. DOI: 10.1016/j.automatica.2004.08.019.

[27] A. Cotorruelo, I. Kolmanovsky, and E. Garone, *A sum-of-squares-based procedure to approximate the pontryagin difference of semialgebraic sets*, arXiv preprint arXiv:2008.06126, 2020. DOI: 10.48550/arXiv.2008.06126.

[28] M. Rehman, J. Alzabut, M. Tayyab, and F. Amir, "Interconnection between schur stability and structured singular values", *Contemporary Mathematics*, vol. 6, no. 1, 2025. DOI: 10.37256/cm.6120255707.

[29] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control", in *Lecture Notes in Control and Information Sciences*, vol. 340, Springer, 2006, pp. 65–93. DOI: 10.1007/978-3-540-36119-0_4.

[30] G. Sanchez, M. Murillo, L. Genzelis, N. N. Deniz, and L. Giovanini, "Mpc for nonlinear systems: A comparative review of discretization methods", in *Proc. XVII Workshop on Information Processing and Control (RPIC)*, 2017.

[31] Ørnulf Damsgaard, "Moving horizon state estimation for predictive control of magnetic levitation systems", M.S. thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2025.

[32] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[33] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties", Institute for Systems and Robotics, Instituto Superior Técnico, Tech. Rep., 2004, Accessed: 2025-05-28.

[34] H. Masnadi-Shirazi, A. Masnadi-Shirazi, and M.-A. Dastgheib, *A step-by-step mathematical derivation and tutorial on kalman filters*, arXiv preprint arXiv:1910.03558, 2019.

[35] C. T. Chen, *Linear System Theory and Design* (The Oxford Series in Electrical and Computer Engineering). Oxford University Press, 2014, ISBN: 9780199964543.

[36] I. Landau and G. Zito, "Computer control systems", in *Communications and Control Engineering*, Springer, 2006, ISBN: 978-1-84628-055-9. DOI: 10.1007/978-1-84628-056-6_2.

[37] A. Kværnø, *Partial differential equations and finite difference methods*, Lecture Notes for TMA4125/4130/4135 Mathematics 4N/D, NTNU, Department of Mathematical Sciences, 2020.

[38] T. O. Hodson, "Root-mean-square error (RMSE) or mean absolute error (MAE): When to use them or not", *Geoscientific Model Development*, vol. 15, no. 14, pp. 5481–5487, 2022. DOI: 10.5194/gmd-15-5481-2022.

[39] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – a software framework for nonlinear optimization and optimal control", *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. DOI: 10.1007/s12532-018-0139-4.

[40] W.-L. Tan, T. H. Lee, and Q.-G. Wang, "Robust multiplexed model predictive control", in *Proc. IFAC World Congress*, vol. 48, Elsevier, 2015, pp. 313–318. DOI: 10.1016/j.ifacol.2015.09.264.

[41] L. D. Tufa and C. Z. Kai, "Effect of model-plant mismatch on mpc performance and mismatch threshold determination", *Journal of Applied Sciences*, vol. 11, no. 20, pp. 3579–3585, 2011.

[42] M. N. Zeilinger, M. Morari, and C. N. Jones, "Soft constrained model predictive control with robust stability guarantees", *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3080–3086, 2014. DOI: 10.1109/TAC.2014.2335611.

[43] X. Wang, Z. Wang, W. Wang, J. Wang, and W. Wang, "Robust model predictive control for path tracking of a tracked vehicle towing a steerable trailer", *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 230–235, 2016. DOI: 10.1016/j.ifacol.2016.07.631.

[44] L. Vanroye, A. Sathya, J. D. Schutter, and W. Decré, "Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control", in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, IEEE, 2023, pp. 10 036–10 043.