Learning-based Approximate Model Predictive Control with Guarantees

Joining Neural Networks with Recent Robust MPC

Master Thesis

Intelligent Control Systems Group
Max Planck Institute for Intelligent Systems

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Johannes Köhler
Dr. Vincent Berenz
Prof. Dr. Raffaello D’Andrea
Dr. Sebastian Trimpe

31.08.2019
Abstract

In this work, we present a model predictive control (MPC) method for applications in complex constrained physical systems. We base our work on a novel robust model predictive control (RMPC) scheme guaranteeing constraint satisfaction and recursive feasibility under disturbances. The used scheme keeps the computational complexity comparable to the nominal case. We adopt this approach and extend it by practical useful additions, such as robust dynamic set point tracking and nonlinear constraints on the output function. We are able to demonstrate this method’s practical significance by controlling a complex robotic system (MPI Apollo robot). Furthermore we tackle the inherent computational complexity of the optimization problem by approximating the RMPC controller with neural network (NN) regression. By only using the NN, we are able to control the robot in a much more dynamic way due to a computational speed up of several magnitudes. Finally, we also analyze the quality of the approximation by providing NN controller closed loop statistical guarantees, also taking into account additional experimentally validated non-idealities of our model. Most of the code, including the TensorFlow NN inference, is fully written in C++ and available on the AMD Gitlab server.
Preface

This literary work was created as a result of the work on my (i.e. Julian Nubert’s) Master Thesis. Besides doing some of the work at the University of Stuttgart, the prime part of the work was performed at the Max Planck Institute for Intelligent Systems. This work is the written part of my Master Thesis for my Master’s degree studies in Robotics, Systems & Control at ETH Zürich, under the official supervision of Prof. Dr. Raffaello D’Andrea. My supervisors at the Max Planck Institute for Intelligent Systems are Dr. Sebastian Trimpe and Dr. Vincent Berenz and my supervisor at the University of Stuttgart is Johannes Köhler.

The core idea of this work, i.e. approximating a robust model predictive controller with a supervised learning technique and to be in a position to give statistical guarantees, originally results from a joint idea of my two supervisors Dr. Sebastian Trimpe and Johannes Köhler. This idea was first elaborated in a previous work done by Michael Hertneck [19], which was supervised by Dr. Sebastian Trimpe and Johannes Köhler.

This thesis can be understood as a continuation of this previous work focusing on further extensions, newly used approaches, more advanced frameworks and the application to a real world robotic use case. It was performed in the time interval lasting from January 2019 till August 2019.

I would like to thank the Max Planck Institute for Intelligent Systems, enabling me to do work on such a great project including real world robot experiments for my Master Thesis. Also big thanks go to Prof. Dr. Raffaello D’Andrea who is the tutor of my Master’s degree and the ETH supervisor of this thesis, making it possible for me to work on it externally. Furthermore, I would like to thank the Intelligent Control Systems Group, which welcomed me in a unique positive way. Conversations with the members of the group, interns and other master students were extremely helpful in tackling involved challenges. Special thanks go to my supervisor, Dr. Vincent Berenz who supported me along the way regarding implementation and framework questions. Another big thank goes to Johannes Köhler who helped me, even remotely during the first three months, in understanding, implementing and extending the robust model predictive control theory and was available whenever I needed help. Last but not least, I would like to thank Dr. Sebastian Trimpe who hired me for this project, supported me during the 7 months by any account, was always available and also offered off-topic help and advice.

Thanks to all of you! Yours sincerely, Julian.
# Contents

## Nomenclature

<table>
<thead>
<tr>
<th>Nomenclature</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>vi</td>
</tr>
<tr>
<td>1.1 Problem Formulation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Approach</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Overview</td>
<td>3</td>
</tr>
</tbody>
</table>

## Related Work

<table>
<thead>
<tr>
<th>Related Work</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Explicit MPC Representation</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Control of Redundant Robot Manipulators</td>
<td>7</td>
</tr>
</tbody>
</table>

## Theoretical Background

<table>
<thead>
<tr>
<th>Theoretical Background</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 MPC - Theory</td>
<td>10</td>
</tr>
<tr>
<td>3.1.1 General MPC Formulation</td>
<td>10</td>
</tr>
<tr>
<td>3.1.2 Set Point Tracking MPC</td>
<td>12</td>
</tr>
<tr>
<td>3.2 RMPC - Theory</td>
<td>13</td>
</tr>
<tr>
<td>3.2.1 Standard Tube-Based Robust Nonlinear MPC</td>
<td>13</td>
</tr>
<tr>
<td>3.2.2 Min-Max MPC</td>
<td>14</td>
</tr>
<tr>
<td>3.2.3 Computationally Efficient RMPC [36]</td>
<td>14</td>
</tr>
<tr>
<td>3.3 Learning - Theory</td>
<td>17</td>
</tr>
<tr>
<td>3.3.1 Supervised Learning</td>
<td>18</td>
</tr>
<tr>
<td>3.3.2 Ability to Overcome the Curse of Dimensionality</td>
<td>18</td>
</tr>
<tr>
<td>3.3.3 Statistical Learning</td>
<td>19</td>
</tr>
</tbody>
</table>

## RMPC Design

<table>
<thead>
<tr>
<th>RMPC Design</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Setup</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Procedure</td>
<td>21</td>
</tr>
<tr>
<td>4.2.1 Algorithm for Offline Computation</td>
<td>21</td>
</tr>
<tr>
<td>4.2.2 Algorithm for Online Computation</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Computations in Continuous Time</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Terminal Ingredients</td>
<td>23</td>
</tr>
<tr>
<td>4.4.1 Terminal Cost - Literature Review</td>
<td>23</td>
</tr>
<tr>
<td>4.4.2 Terminal Set - New Approach Based on Incremental Lyapunov Function</td>
<td>25</td>
</tr>
<tr>
<td>4.5 Summary - Resulting Optimization Problem</td>
<td>27</td>
</tr>
<tr>
<td>4.5.1 Constant Set Point</td>
<td>27</td>
</tr>
<tr>
<td>4.5.2 Varying Set Point</td>
<td>27</td>
</tr>
<tr>
<td>4.6 Derivations and Computations of Required Parameters - A Comprehensive Collection</td>
<td>28</td>
</tr>
<tr>
<td>4.6.1 Incremental Lyapunov Function</td>
<td>28</td>
</tr>
<tr>
<td>4.6.2 Constraint Tightening</td>
<td>30</td>
</tr>
<tr>
<td>4.6.3 Check Exponential Decay Rate</td>
<td>30</td>
</tr>
<tr>
<td>4.6.4 Terminal Ingredients</td>
<td>31</td>
</tr>
</tbody>
</table>
5 AMPC Approach

5.1 Problem Setting ........................................ 32
5.2 Procedure .............................................. 33
5.3 Previous Work [19] ...................................... 33
  5.3.1 Validation Criterion ................................ 33
  5.3.2 Statistical Guarantees .............................. 34
5.4 Validation ................................................. 34
  5.4.1 New Validation Criterion ......................... 34
  5.4.2 Statistical Guarantees .............................. 35
5.5 New Ways of Sampling .................................. 35
5.6 Improved Neural Network Approximation ............... 36

6 Frameworks .................................................. 37

6.1 Overview ................................................. 37
6.2 Model Predictive Control Framework .................... 37
  6.2.1 ACADO ............................................... 37
  6.2.2 CasADi .............................................. 38
6.3 Machine Learning Framework - TensorFlow ............. 38
  6.3.1 Training - Python .................................. 38
  6.3.2 Inference - C++ ................................... 38
6.4 Real Time Robot Control Framework - SL ............... 39
  6.4.1 Real Time .......................................... 40
  6.4.2 Modularity .......................................... 40
  6.4.3 Simulation .......................................... 40
6.5 Motion Tracking - Vicon ................................ 40
  6.5.1 Vicon Bridge ....................................... 40
6.6 Developed Frameworks ................................... 40
  6.6.1 Conclusion .......................................... 40

7 Simulation Example - CSTR .................................. 42

7.1 System Formulation ...................................... 42
  7.1.1 Continuous Time Formulation .................... 42
7.2 Setup ..................................................... 43
  7.2.1 Groundtruth ........................................ 43
  7.2.2 Assumed Disturbance .............................. 43
  7.2.3 Incremental Stabilizability ...................... 43
  7.2.4 RMPC Design ..................................... 44
  7.2.5 Learning - Combining Regression and Classification 44
  7.2.6 Alternative Sampling .............................. 44
7.3 Results .................................................... 46
  7.3.1 New Learning Approach ........................... 46
  7.3.2 Pure Regression ................................... 47
  7.3.3 Simulation .......................................... 47
  7.3.4 Validation .......................................... 47
  7.3.5 Overall Comparison ............................... 49

8 Introducing Apollo Robot .................................... 50

8.1 Geometry .................................................. 50
8.2 Robot Description ......................................... 50
  8.2.1 Identification of the Dynamics Equations ........ 50
8.3 General Kinematics of KUKA LBR4+ ..................... 51
8.4 SL Specifics and Constraints ............................ 53
  8.4.1 Joint Offset DH convention to SL ............... 53
  8.4.2 Joint Limits ....................................... 53
Nomenclature

Symbols

\( t \in \mathbb{R} \) \hspace{1cm} Time \hspace{1cm} [s]
\( T \in \mathbb{R} \) \hspace{1cm} Sampling Time of Controller \hspace{1cm} [s]
\( x \in X \subseteq \mathbb{R}^n \) \hspace{1cm} System State Vector
\( u \in U \subseteq \mathbb{R}^m \) \hspace{1cm} Input Vector
\( y = h(x, u) \) \hspace{1cm} System Output Vector
\( \Theta \) \hspace{1cm} Angle \hspace{1cm} [rad]
\( \dot{\Theta} \) \hspace{1cm} Angular Velocity \hspace{1cm} [rad/s]
\( \ddot{\Theta} \) \hspace{1cm} Angular Acceleration \hspace{1cm} [rad/s^2]

Notation

\( x(k|t) \) \hspace{1cm} Predicted discrete time system state at time step \( k \) starting at time \( t \)
\( x(\tau|t) \) \hspace{1cm} Predicted continuous time system state in sampling interval, for \( \tau \in [0, T) \)
\( |x|_Q^2 \) \hspace{1cm} Quadratic matrix norm given as \( x^TQx \)
\( A, B \) \hspace{1cm} Matrices of linear System Dynamics

Acronyms and Abbreviations

AMPC \hspace{1cm} Approximate Model Predictive Control
CSTR \hspace{1cm} Chemical Stirred-Tank Reactor
CT \hspace{1cm} Continuous-Time
DH \hspace{1cm} Denavit-Hartenberg
DT \hspace{1cm} Discrete-Time
ETH \hspace{1cm} Eidgenössische Technische Hochschule
GPS \hspace{1cm} Guided Policy Search
LQR \hspace{1cm} Linear Quadratic Regulator
MPC \hspace{1cm} Model Predictive Control
MPI \hspace{1cm} Max-Planck-Institute
NN \hspace{1cm} Neural Network
RMPC \hspace{1cm} Robust Model Predictive Control
SL \hspace{1cm} Simulation Laboratory (robot control framework)
Chapter 1

Introduction

Applications of modern electrical, mechanical or electro mechanical systems would not be possible without the use of feedback control. Classical control schemes such as P-, PI-, PD- or PID control have performed reliably for a long time and are nowadays de facto still industrial standard. However, due to increasing complexity of the systems and rising demands in terms of performance, such modest approaches are often not longer expedient. In the last decades optimal and robust control techniques such as Linear Quadratic Regulator (LQR) [27], $H_2$- and $H_\infty$-[70] control became more and more important. Especially for multivariable systems these approaches guarantee stability and achieve good results for many applications, offering properties such as optimal performance for a given problem set (LQR) or acting robustly with respect to defined disturbances and uncertainties ($H_2$, $H_\infty$).

Figure 1.1: Apollo robot at the MPI in Tübingen equipped with two KUKA LBR4 robotic arms. Max Planck Institute for Intelligent Systems

Nevertheless, none of these approaches is able to properly handle additional restrictions such as state and input constraints. Extensions and ways of handling nonlinearities are possible but still challenging in many situations. In contrast to this, Model Predictive Control (MPC) techniques allow one to consider state and input constraints in the formulation and therefore to steer the system in the optimal way, potentially even by consciously applying inputs lying at the constraint boundaries. Even handling nonlinear output constraints is possible, as we demonstrate within this work. MPC allows for finding the optimal solution along a predicted trajectory, given some constraints and a predefined cost function. Furthermore, MPC is relatively easily extendable to
general nonlinear dynamics.

Standard MPC design only works well if the given model reflects reality to a certain degree. Even basic requirements such as stability can only be guaranteed for the nominal and therefore often non-realistic case. To be in a position to deal with uncertain real world scenarios and hence, to guarantee safe operation, several robust MPC (RMPC) schemes have been developed in the past years. These are able to take disturbances and uncertainties into account and guarantee stability and recursive feasibility for certain (predefined) disturbances. However, a general drawback of MPC or RMPC approaches is its computational complexity since a mathematical optimization problem needs to be solved in every time step. Therefore, MPC controllers have first been applied to control of systems with larger time constants such as chemical reactors [66].

1.1 Problem Formulation

Examples for challenges and requirements in modern control design can, e.g., be found in two recent research areas; automated driving and human-robot-collaboration. It is obvious that both examples are complex, highly safety critical, relatively uncertain and constrained. In fact, they are representative for a full class of modern control problems. Besides the mentioned challenges, many of them are highly nonlinear and in addition high dimensional.

This is exactly the area we tackle within this project. Our superior goal of this thesis is the deployment of a novel control approach for a complex real-world robotic system, more precisely the end-effector control of an humanoid-like robot called Apollo which is depicted in Fig. 1.1. Apollo is equipped with two KUKA LBR4+ robotic arms and located at the MPI for Intelligent Systems in Tübingen. We try to obtain

1. state of the art performance,
2. while ensuring safety,
3. as well as the ability to deploy our approach in fast acting or computationally constrained systems.

Guaranteeing safety in this area requires the avoidance of real world obstacles, in particular in a robust manner. Furthermore, the latter goal - to offer a time efficient solution - would facilitate a utilization in other areas, such as flight-control for quadrotors. Apollo has hard constraints on its state- and input space. Moreover, we require to take obstacles in the task space into account. This motivates the following chosen approach.

1.2 Approach

Our approach was chosen with the requirements given in the last section in mind. For reaching state of the art performance for our task, we aim to do the robot end-effector control based on MPC. In our opinion this is the most direct solution for the given problem, which involves amongst others the need of handling hard constraints. To guarantee safety, we went for a novel robust MPC (RMPC) scheme by Koehler et al. [36]. This approach guarantees stability, robust constraint satisfaction as well as recursive feasibility under bounded disturbances. The biggest advantages of this approach are moderately introduced conservatism and a computational complexity comparable to the nominal case.

To tackle the time critical demands, we choose to get an explicit representation of the RMPC by approximating it with supervised learning, neural networks (NN) in our case. We denote the resulting control technique as approximate model predictive control (AMPC). The foundation of the approach was adopted from [19]. By exploiting the underlying robustness properties of the deployed RMPC and using statistical bounds, we are able to give guarantees for the resulting NN-AMPC.

To reach our goals, strong focus lies on the development of a suitable software framework. It is used to sample the controller, train the networks, do the validation and to actually control the
Chapter 1. Introduction

Figure 1.2: Overview of the main components of this work. We use a novel RMPC design [36] to guarantee safe behavior, including stability and recursive constraint satisfaction. Furthermore, we use this controller to train an AMPC. Vice versa, the RMPC is used afterwards to validate the resulting AMPC. The development of the framework was a central part of this work, among others to be able to use the RMPC as well as the AMPC to control Apollo robot.

As just mentioned, we investigate and deploy a novel RMPC scheme based on the work in [36] which is similar to works by Limon et al. [41, 45, 42]. We extend it by additional features needed for practical application on a robotic system. These include the extension to dynamic robust (output) set-point tracking, online-computation of a suitable terminal set and the robust handling of nonlinear output constraints. The inherent computational complexity of the optimization problem is tackled with a recent learning-based approach, based on the idea in [19]. We improve the given approach by introducing a new and less conservative validation criterion, using more advanced learning techniques and by improving the sampling of the data points. In particular for the high dimensional robot control the sampling acts as the bottleneck of the whole pipeline.

The idea of using sampled solutions of the optimization problem and creating an explicit MPC controller is not new and was done before (see Chapter 2). However, in comparison to most existing solutions, we are able to give statistical guarantees regarding stability even though ML techniques are used, as introduced in [19]. This is done by taking disturbances into account during the design procedure of the RMPC controller, enabling the controller to compensate for them. We consider disturbances in the form of imprecisions of the NN and model uncertainties of the real world system. Hence, our MPC controller is designed in a way to take into account both types of disturbances.

From the very beginning we put emphasis on using fast and powerful tools regarding numerics, optimization and ML. During this project we developed a reliable and fast working software framework, fully implemented in C++. By using the inference of the NN for the needed control signals we gain a potential speedup of several magnitudes compared to solving an optimization problem in every step. Hence, by doing so we are able to control the system in a significantly more dynamic way.

1.3 Contributions

As introduced in the last section, we worked on all areas which are depicted in Figure 1.2. None of the topics remained unchanged to previous work and we can present improvements in each
1.3. Contributions

of the areas. This thesis makes contributions to the two broader areas of robust MPC (RMPC) and approximate MPC (AMPC). Furthermore, a real-time safe software framework is developed to deploy both approaches safely on the robot. Finally, we were able to demonstrate both the extended RMPC scheme as well as this AMPC controller on a complex real world system for the first time, pointing out the significance for the robotics community.

In this work we extended a RMPC scheme based on [36] in order to make it suitable for the described motion control task of a robotic system. We provide all derivations and needed computations within this work. The main extensions compared to the work in [36] are the following:

- We perform output set point tracking, i.e. the reference $x_t$ in the state space is unknown; only the reference output of the system $y_t$ is given, i.e. the reference of the robot in the task space. To overcome this, an artificial set point $x_s$ is integrated into the formulation according to [43].

- We can do dynamic set point control, i.e. the artificial set point $x_s$ is not known in advance and changes during the operation. This requires the design and computation of proper terminal ingredients. We propose a formulation of suitable terminal ingredients for which the size of the terminal set is optimized in an online fashion.

- We deploy a quasi real-time prestabilization to make the system more good-natured and allow for more dynamic control.

The design contains additional features and considerations which are needed for many real world use cases but often not regarded in theoretical analysis and simulation examples. Regarding the theory for these special use cases we got inspired by works of Koehler et al. [32, 35].

Furthermore, we present a technique for taking the RMPC controller as given, sampling it and then creating an AMPC controller. Many theoretical ideas were adopted from [19]. However, we made it practically more applicable by

- introducing a new and less conservative validation criterion for the AMPC controller.

Besides the fact that it enables us to be less conservative, in contrast to [19] it also allows us to take additional disturbances such as model uncertainties or measurement noise into account. We also present alternative techniques and approaches for the sampling, training and NN prediction of the AMPC approach. This includes mainly the following:

- We present more efficient sampling techniques such as recursive sampling and sampling by trajectories. We can also demonstrate its advantages compared to the obvious (uniform or random) approaches.

- We deploy additional supervised classification to eliminate the need of high sampling density in all areas of the state space.

For a more high dimensional problem we try to overcome the curse of dimensionality and achieve decent results despite a small amount of samples per dimension. In this case we combine random sampling - allowing the network to get a good idea about all areas - with trajectory sampling leading to higher precision in regions of interest. The introduced improvements enable us to reach a speed up by a factor of up to 50 for the simulation example investigated in [19] while reaching the same performance.

As another important part of this work, we demonstrate the direct control of the position of the end effector of an industrial robot in a complex real world robot experiment, using the presented RMPC as well as the AMPC. Hence

- we show the practicability of [36] by applying it to a complex real world system for the very first time.

No additional packages or algorithms for path planning, inverse kinematics etc. are needed. This approach is flexible and allows for easy integration of additional constraints.
Chapter 1. Introduction

For reaching those goals and really underlining its practical significance, we developed a software framework, notably including C++ code integrated in a real-time environment. It involves extensive optimization and neural network inference, self written sampling procedures, parallelization and simulations. Furthermore, we present MATLAB code for doing the RMPC offline calculations and for doing the uncertainty quantification. Finally we also offer some Python code for visualization purposes and the training of the NNs, ready to be used in a cluster environment.

1.4 Overview

As seen in the last section, this Master Thesis consists of multiple bigger parts. In Chapter 2 we present related work and delimitate what we have done. In Chapter 3 we introduce the concepts for MPC and RMPC design needed to understand our used concepts and their motivation. In Chapter 4 we then go into more detail regarding the used RMPC scheme and explain the needed modifications we propose in order to be able to get the required functionality and to still guarantee stability and recursive feasibility. Chapter 5 then introduces the formulation of our AMPC approach and explains the modified validation criterion we use for the validation.

With Chapter 6 - introducing the used frameworks and toolboxes - we get started with the more practical parts of this work. We set up the whole pipeline once for a simpler simulation example. We choose a chemical stirred tank reactor (CSTR) which is also used in [19]. We show the needed steps and results in Chapter 7. Providing the theory, doing the computations and implementing the software is the biggest and most challenging part of this thesis. We go into detail about this in Chapter 9 after having introduced the robotic system we are using in Chapter 8.

We conclude the thesis in Chapter 10. We discuss our results, summarize our findings and show potential areas for future work.

Additionally we also provide a user manual which can be found in the Appendix of this thesis, containing more practical components, e.g. how to install the frameworks and how to make use of them. Also a detailed description of how to use our developed code is provided.
Chapter 2

Related Work

In this chapter we will mainly describe related work with respect to two aspects: other approaches to do fast explicit (and potentially approximate) MPC and other techniques to do robot end-effector control with relations to ours. Work we have used to design our approach (i.e. mainly for the RMPC design) as well as further extensions for potential future work are described directly in the related chapters.

2.1 Explicit MPC Representation

In [44], Lovelett et al. tackle the computational complexity of solving the MPC formulations at every time step. They make use of data mining and manifold learning to make predictions of the MPC control policy for complicated function mapping from the system state. Particularly, they design a similarity measure to learn an intrinsic parameterization of the MPC controller using a diffusion maps algorithm (i.e. a nonlinear dimensionality reduction). In contrast to a fully learned formulation as in our case, they use this function approximation to first project the points from the state space to this intrinsic space. Then, in a next step another function approximation is used to map it from this space to the state policy space. The mapping from the intrinsic space to the control policy is usually also done by using polynomial regression or artificial neural networks. The first part, the manifold learning is amenable to alternative parameterizations (instead of diffusion maps). They also demonstrate the approach by showing a simulation example of the continuously stirred-tank reactor (CSTR) as we do in Section 7.

In [11], Chen et al. present an approach for approximating explicit MPC by using constrained neural networks. The corresponding optimal MPC control law for constrained LQR problems is piecewise affine on polytopes but becomes computationally intractable for high dimensional systems. As a solution the authors propose a modified reinforcement learning policy gradient making use of knowledge about the system model. Noticeable is that guarantees can be given with respect to the feasibility of the control inputs generated by the network, by deploying Dykstra’s algorithm for projecting the output of the NN. The latter could also be an interesting extension to our work. A strong drawback of this approach is that this projection is done to an offline computed control invariant set, which is inherently limited to low dimensional linear systems. They evaluate the performance of the described approach on numerical examples including a 2D double integrator and a 4D linear system.

Lucia et al. propose in [1] a deep learning-based approach to robust nonlinear model predictive control. This approach’s idea of approximating generated policies with deep networks is similar to the one of [19] and ours. However, in contrast to [19] or us, they use multi-stage NMPC for generating data. One of their main results is that they were able to show that deep networks perform significantly better in learning the policy compared to shallow networks. However, a big downside here is that the underlying multi-stage NMPC only provides an approximation for continuous-valued uncertainty and disturbance. Furthermore, no guarantees on the resulting NN can be given, in contrast to [11] and [19] or us.
Another recent work is given in [2], where the authors present a framework for approximating the explicit MPC law for linear parameter-varying systems. In particular, additional to the control policy they also learn a certificate policy to estimate the sub-optimality during execution time. Both policies are learned from data while providing a randomized strategy guaranteeing the quality of the learned policy. This allows the authors to bound the probability of a solution for being infeasible or sub-optimal. The algorithm doesn’t require the solution of an optimization problem online and can therefore be applied on resource-constrained systems, similar to us. They are able to achieve a performance speedup of up to two magnitudes with small performance degradation on a vehicle dynamics control problem in simulation.

In [9], the authors present an approximation of a nonlinear model predictive controller back in 2008. They use membership function approximation methodologies under the nearest point approach. Furthermore, a finite number of solutions of the nonlinear MPC controller is computed offline. The functions deployed for the approximation fulfill input constraints and are computation-wise time independent w.r.t. the control horizon. However, this approach is not easily scalable to higher dimensions and increases in computational complexity and memory demands with increasing number of samples.

Another interesting work is given in [28]. In this work, the authors analyze the ability of deep neural networks to approximate MPC control laws from a theoretical perspective. They provide theoretical bounds on the minimum number of hidden layers and neurons per layer that a network should have to exactly represent a given policy. Finally, they also present an approximate explicit MPC scheme. On the one hand, they deal with the approximation error similar to [11] by doing feasibility recovery based on control invariant sets which can be challenging in practice. On the other hand, they present an a-posteriori statistical verification technique to compute safe sets, based on trajectories. The safe set is obtained by either representing it as an ellipsoidal with a convex optimization for finding the maximum-sized hypercube enclosing this ellipsoid, or by using support vector machines (SVM) with a suitable SVM learning problem. The ellipsoidal set is a more conservative solution while the SVM allows a trade-off between the size of the safe set and the proneness towards misclassification. Finally, they validate those safe sets, since their computation is based on a finite number of data points. In contrast to [19] and similar to us, Karg et al. don’t try to find statistical guarantees on the closed-loop behavior but instead on the closed-loop performance of the approximate solution. For the sake of this statistical verification, the authors also use Hoeffding’s inequality, similar to us.

In contrast to the previously mentioned works, in [24] the authors are not proposing another approach for approximating MPC controllers but instead, they present a hybrid system approach (Verisig) to verifying safety properties of closed-loop systems using neural networks as controllers. In contrast to other work in this field, Verisig is also able to verify properties of the closed-loop system for sigmoid-based networks. By exploiting the fact that sigmoid functions are a solution of quadratic differential equations they find a way of transforming the neural network to an equivalent hybrid system. They can decide for reachability for networks with one hidden layer or for general networks with Schanuel’s conjecture. In the end, the authors evaluate Verisig on a neural network used to approximate an MPC. However, this approach is only applicable to linear hybrid systems and finite number of possible actions (e.g. bang-bang control of control-affine systems following a piece-wise linear plan.

2.2 Control of Redundant Robot Manipulators

In the following we will explain how control of redundant robot manipulators is usually done.

In a tutorial [56] from 1990, Siciliano et al. explained the state of the art motion control techniques at that time. They formulated the kinematic control problem as finding a joint space trajectory \( q(t) \) s.t. \( f(q(t)) = x(t) \), with \( x(t) \) being the trajectory in the task space. The direct kinematic mapping can be written as \( x = f(q) \) and therefore \( \dot{x} = J(q)\dot{q} \). As usually redundant robots are used, the inverse kinematics problem admits an infinite number of solutions. The authors suggest to exploit this redundancy to meet additional constraints and describe that most
2.2. Control of Redundant Robot Manipulators

approaches at that time tried to invert the mapping from joint space to task space. One way is for example to use the Moore-Penrose pseudoinverse of the Jacobian matrix to obtain \( \dot{q} = J^\dagger(q) \dot{x} \).

This approach generates the minimum norm joint velocities. However, this approach doesn’t avoid kinematic singularities in any practical sense. As a result other (non-singular) inversion variants of the Jacobian have been proposed. Siciliano et al. mention, that one of the big drawbacks of this class of approaches is that a solution of the type \( \dot{q} = K(q) \dot{x} \) as we have just described is inherently open-loop, since this equation is integrated over time to generate the desired joint path \( q(t) \). This results in unavoidable drifts in the task space due to integration on the computer in DT. To overcome this, an additional error term can be introduced to the task space vector \( \dot{x} \) which captures the deviation to the desired trajectory in the task space. This would then look as following: \( \dot{x} = \dot{x}_d + \Lambda \cdot e \). In this tutorial, further approaches such as gradient projection, task space augmentation and the finding of inverse kinematic functions are described. The solution of the latter is always given for simple non-redundant geometries and allows for online implementation. However, all of the mentioned approaches assume a trajectory \( x(t) \) given beforehand, which is not given in our case (see Chapter 8).

A more recent and more complete collection of methods and approaches is given in the book [57]. In Chapter 4 of the book, methods for trajectory planning are described. In Chapter 8, the authors devoted a separate chapter to motion control. In this chapter they explain two classes of methods: joint space control as well as operational space control. For both of the formulations calculation of inverse kinematics is needed. For joint space control this is done before the feedback loop, i.e. the control is done in an open loop fashion, for the latter formulation the inverse kinematics are embedded in the feedback control loop. In contrast, our approach only requires forward kinematics and thus, we don’t need to design multiple decoupled components as involved in other robot control approaches.

As a next step, we will mention works which address robot control more recently and with more advanced techniques.

In 2006, Bertram et al. [6] describe an integrated approach for path planning of redundant manipulators. They propose a way of unifying the calculation of the goal configuration with the search of a suitable path. They adopt workspace heuristic functions implicitly defining goal regions of the configuration space and guide an extension of rapidly-exploring Random Trees (RRT) used for searching these regions. The approach is demonstrated in simulation for a humanoid grasping task.

In [60], Vahrenkamp et al. present an approach for efficient motion planning for a dual-arm manipulator. They investigate solving of the inverse kinematics problem and motion planning by combining a gradient-descent approach in the robots pre-computed reachability space with random sampling of free parameters. Additionally they propose two RRT-based motion planning algorithms that interleave the search for an inverse kinematic solution with the search of a collision-free trajectory. They demonstrate this approach in simulation and on the humanoid robot ARMAR-III.

Bargsten et al. propose a full framework for control of the KUKA LBR4+ robot from performing system identification to model-based control [5]. For the control task, they are solving a trajectory tracking problem by using a computed torque controller (a.k.a. inverse dynamics controller). They choose a feedback for linearizing the dynamics of the system, i.e. they are performing exact feedback linearization to the system. This is also highly relevant, since it allows to transform the robot dynamics to the kinematic formulation we investigate in this work. They tested it on the real world system but only for sampling rates up to 100 Hz, since Python was used for the implementation. However, they still achieved decent tracking results.

In the next step, we will look into MPC approaches used for robot control.

In [13], the authors look at a real-time feasible implementation of a model predictive path-following control for the KUKA LBR4+ robot. They investigate constrained output path-following with and without reference speed assignments. Similar to our work, they are also trying to track a geometric reference in the output space. They also consider the system with input and state constraints and make use of a CT sampled-data nonlinear MPC scheme which they denote as model predictive path-following control. In contrast to us, they base the control on an augmented
system description not only containing the original states but also a path parameter $\Theta$. In their work they treat contact forces as disturbances and solve the optimization problem by using the ACADO real-time iteration scheme. They demonstrate their approach in practical results. For the optimization they consider a quadratic cost trying to minimize the error, the deviation to $\Theta$ as well as to $\dot{\Theta}_{\text{ref}}$. For the whole computation they assume the geometric reference in the output space to be given by $P = \{ y \in \mathbb{R}^n \mid \Theta \in \mathbb{R} \mapsto y = p(\Theta) \}$ with parameterization $p(\Theta)$. In our work, we are not constraining ourselves to such a parameterization and can principally track every (even non-reachable) 3D output reference.

In [4], the authors also describe a MPC path following and trajectory tracking approach, however not specifically for the KUKA LBR4 robotic arm. Arbo et al. propose a model predictive path following approach which allows robotic manipulators to stop at obstructions. Similar to us, they assume the robot to have known forward kinematics. In contrast to us, they deploy a constrained path-timing variable $s$ moving from 0 to $s_f$ and for their approach the path needs to be given in the task space beforehand. They define two cost functions for the two approaches (path following and trajectory tracking) and demonstrate their work. They compare it to model predictive trajectory tracking in a simulation example for a two-link manipulator.


Chapter 3

Theoretical Background

In this section, we introduce the necessary theoretical background that this thesis builds upon. We start by describing standard MPC theory in Section 3.1. Then we will describe techniques for doing RMPC, including the core work from Köhler et al. [36], in Section 3.2. Finally, we will close this chapter with background theory in Machine Learning (ML).

3.1 MPC - Theory

MPC is a control scheme trying to repeatedly obtain the open-loop finite horizon optimal control. In contrast to standard optimal control such as LQR control its formulation is easily expandable to nonlinear systems and it is also able to take several types of constraints, such as input and state constraints into account. In this chapter we will show the formulation of the corresponding optimization problem. Additionally, we will provide the most important requirements for successfully designing a nominal MPC controller. In Sec. 3.1.2 we will introduce a special formulation to be in a position to perform (output) set point reference tracking which will be used in the practical application (Chap. 8) of this thesis.

3.1.1 General MPC Formulation

For theory on MPC we refer to the standard work in [53]. We first make the following definition:

**Definition 3.1.** Compact nonlinear constraint set:
\[ Z = \{(x, u) \in \mathbb{R}^{n+m} | g_j(x, u) \leq 0, \ j = 1, ..., p \} \subset \mathbb{R}^{n+m} \]

Including the definition of this constraint set, the simplest standard MPC formulation can be seen in (3.1):

\[
\begin{align}
V_N(x(t)) &= \min_{u(t)} \sum_{k=0}^{N-1} l(x(k|t), u(k|t)) \\
\text{subject to} \quad &x(0|t) = x(t), \\
&x(k+1|t) = f(x(k|t), u(k|t)), \\
&(x(k|t), u(k|t)) \in Z, \\
&k = 0, ..., N - 1,
\end{align}
\]

(3.1)

\[ l(x(k|t), u(k|t)) \] denotes the objective and \( Z \) defines the state and input constraints. Additionally the state must evolve according to the given state dynamics. In practice the objective is very often formulated as a sum of 2 quadratic norm terms:

\[ l(x, u) = |x|_Q^2 + |u|_R^2. \]

(3.2)
Practically, systems are usually controlled using the receding horizon strategy [48]. This means that at every time step a complete optimization problem along a prediction horizon is solved and the optimal input \( u^*(0|t) \) is applied to the system. Sometimes it is possible to obtain good results by using the simple formulation in (3.1). However, having stable and converging open-loop solutions does not necessarily imply stability for the resulting closed-loop system. Especially short prediction horizons can cause the loss of stability properties, i.e. the trajectories may not converge to the origin. Additionally no recursive feasibility is guaranteed, i.e. the problem might not have a solution at all due to non avoidable constraint violations. We pick up on the following common definition.

**Definition 3.2.** The feasible set \( \mathcal{X}_N \) is defined as the set of initial states \( x \) for which the MPC problem with horizon \( N \) is feasible.

For the standard MPC formulation, the initial state might not lie within the feasible set. This problem originates from the use of a short sight strategy; ideally we would solve the MPC problem with horizon \( N \) which is practically intractable. For more information see [67].

**Stability and Recursive Feasibility**

We introduce three common definitions (see [67]).

**Definition 3.3.** The MPC problem is called recursively feasible, if for all feasible initial states feasibility is guaranteed at every state along the closed-loop trajectory.

**Definition 3.4.** Lyapunov Stability: The equilibrium point at the origin of a system \( x(k + 1) = f(x(k), \kappa(x(k))) \) is said to be stable in \( \mathcal{X} \) if for every \( \epsilon > 0 \), there exists a \( \delta(\epsilon) > 0 \) such that, for every \( x(0) \in \mathcal{X} \):

\[
||x(0)|| \leq \delta(\epsilon) \Rightarrow ||x(k)|| < \epsilon \quad \forall k \in \mathbb{N}
\]

**Definition 3.5.** Invariant Set: A set \( \mathcal{O} \) is called positively invariant for system \( x(k + 1) = f(x(k), \kappa(x(k))) \), if \( x(k) \in \mathcal{O} \Rightarrow f(x(k), \kappa(x(k))) \in \mathcal{O}, \quad \forall k \in \mathbb{N} \).

The main approach for ensuring stability and feasibility is to introduce terminal ingredients, i.e. a terminal cost and a terminal set. The MPC formulation then looks as follows:

\[
V_N(x(t)) = \min_{u(|t|)} \sum_{k=0}^{N-1} l(x(k|t), u(k|t)) + V_f(x(N|t)) \quad (3.3a)
\]

subject to

\[
\begin{align*}
&x(0|t) = x(t), \quad (3.3b) \\
&x(k+1|t) = f(x(k|t), u(k|t)), \quad (3.3c) \\
&(x(k|t), u(k|t)) \in \mathcal{Z}, \quad (3.3d) \\
&x(N|t) \in \mathcal{X}_f, \quad (3.3e) \\
&k = 0, \ldots, N - 1,
\end{align*}
\]

Introducing terminal ingredients gives us many desirable properties for the control scheme. However, it comes at the cost of a smaller region of attraction. Especially for non reachable set points this MPC formulation can become infeasible in situations when controllers without terminal ingredients still perform well. In the following we will always assume to have reachable set points or use the set point tracking formulation from Sec. 3.1.2. A first trivial solution for the terminal set is to use the following constraint:

**Equality Constraint:** \( x(N|t) \in \mathcal{X}_f = 0 \), \quad (3.4)

which guarantees recursive feasibility and stability under the assumption of \( x(0|t) \) belonging to the feasible set. The terminal cost is zero, i.e. \( V_f(x(N|t)) = 0 \). However, it strongly reduces the region of attraction (feasible set). Therefore, the general goal is to use a more general set for \( \mathcal{X}_f \)

\[
\text{Set Constraint: } x(N|t) \in \mathcal{X}_f, \quad (3.5)
\]
that contains the set point (or without loss of generality the origin after transforming the coordinates \( \tilde{x} = x - x_s \)) in its interior.

**Assumption 3.1.** The stage cost is a positive definite function. The terminal set is invariant under the local control law \( \kappa_f(x) \). All state and input constraints are satisfied in \( X_f \). The terminal cost \( V_f(x) \) is a continuous Lyapunov function in the terminal set \( X_f \).

**Theorem 3.1.** Under Assumpt. 3.1 and the knowledge of \( x(t) \) being part of the feasible set \( X_N \), recursive feasibility and stability are guaranteed.

**Proof.** This can be shown and is a standard proof, e.g. see [67].

This argument holds for general nonlinear systems. However, this doesn’t tell us how to find the terminal ingredients.

**Linear Systems** For linear systems the problem can be split up in two subproblems: until time step \( N - 1 \) we allow constraints being active whereas for \( k \geq N \) we require the constraints to be inactive. Then we can assume to apply the LQR control law from step \( N \) to \( \infty \) and therefore get for the terminal cost the infinite horizon cost of the LQR controller

\[
V_f(x(N|t)) = |x(N|t)|^2_P, \tag{3.6}
\]

with \( P \) being the solution of the DT algebraic Riccati equation. To make the previous terminal cost valid, the terminal set \( X_f \) needs to be chosen such that constraint satisfaction can be guaranteed when applying the LQR control law \( K_{LQR}x \).

**Nonlinear Systems** For nonlinear systems, Chen et al. [10] present a MPC scheme that guarantees asymptotic closed-loop stability and recursive feasibility, given the optimization problem at time \( t = 0 \) is feasible. For this purpose, a terminal region is introduced which is invariant for the nonlinear system, controlled by a local linear state feedback. The terminal cost they introduce is quadratic and bounds the infinite horizon cost of the nonlinear system controlled by the same linear state feedback, starting from their terminal region. The size of the obtained terminal region hereby strongly depends on the nonlinearity of the system to be controlled; the stronger the nonlinearity, the smaller the terminal region will be. For details on how to find the terminal region and terminal cost for this approach please have a look at Sec. 4.4.

### 3.1.2 Set Point Tracking MPC

In many situations the goal of using a controller is to do output reference tracking, i.e. to track a given set point \( y_t \) with the actual output \( y \) of the corresponding plant. MPC controllers in their standard formulation (see (3.1) and (3.3)) can not be used for this task, particularly not if no inverse transformation from the output to the states is known along the prediction horizon, i.e.

\[
x_s = g_x(y_s), \quad u_s = g_u(y_s) \tag{3.7}
\]

are unknown. For this application Limon et al. developed a MPC formulation allowing for output reference tracking. It was introduced in [40] and generalized to nonlinear systems in [43]. The cost function in (3.8) was slightly adapted to our use case but is conceptually taken from [43].

\[
J_N(x, y_t; u(\cdot|t), y_s, x_s, u_s) = \sum_{k=0}^{N-1} l(x(k|t) - x_s, u(k|t) - u_s) + V_f(x(N|t) - x_s, y_s) + V_O(y_s - y_t) \tag{3.8}
\]

In this case \( V_O \) needs to be positive definite. Next to the well known control inputs \( u(\cdot|t) \), additional decision variables \( y_s, x_s \) and \( u_s \) are introduced. With this cost function we get the set point tracking problem.
MPC control law according to [43]:

\[ V_N(x(t), y_t) = \min_{u(t)} J_N(x, y; u(t), y_s, x_s, u_s) \]  

(3.9a)

subject to

\[ x(0) = x(t), \]  

(3.9b)

\[ x(k + 1) = f(x(k), u(k)), \]  

(3.9c)

\[ (x(k), u(k)) \in Z, \]  

(3.9d)

\[ x_s = f(x_s, u_s), \quad y_s = h(x_s, u_s), \]  

(3.9e)

\[ (x(N), x_s) \in X_f, \]  

(3.9f)

\[ k = 0, ..., N - 1, \]

This approach has several advantageous properties:

- It converges to the desired setpoint \( y_t \), if admissible.
- The optimal setpoint is asymptotically stable for piece-wise constant references.
- For unreachable set points, the asymptotic stability of the optimal reachable set point is given.
- Constraint tightening of the RMPC approach can easily be applied.

We use this approach for our practical application on Apollo (Chap. 8). Further explanations on the implementation are given along the way.

3.2 RMPC - Theory

In Section 3.1 we introduced several MPC formulations also including guarantees for stability and recursive feasibility. However, its design always relies on an exact model description which is never given in practice. As a result, it can happen that stability and recursive feasibility are lost when disturbances are introduced in simulation or the MPC is deployed on a real world (physical) system. RMPC tries to account for the deviation of the model by making it robust to disturbances.

3.2.1 Standard Tube-Based Robust Nonlinear MPC

One way of dealing with these disturbances for nonlinear systems is tube-based robust nonlinear MPC. Tube-based MPC can be seen as an implementable form of feedback MPC which can ensure that the deviation of the actual state from the nominal state is smaller than that obtained with open-loop MPC [46].

Mayne et al. [46] described in their work a popular tube-based MPC extension to nonlinear systems. In their approach, first a reference trajectory is determined by solving the nominal MPC problem with tightened constraints. Then in a second step, the trajectories of the disturbed system are forced to lie within a tube with the center being the previously computed reference trajectory. In their description they only cover additive disturbances, i.e. \( w \in W \) in

\[ x(k + 1) = f(x, u) + w, \]  

(3.11)

with \( W \) being a compact, convex set containing the origin and \( f \) being twice continuously differentiable. The simple approach of using a linear feedback as in [47] is unlikely to work for nonlinear
systems. Hence, Mayne et al. employ a conventional ancillary MPC controller additional to the nominal open-loop MPC trajectory to attract the actual state of the disturbed system towards the nominal one. Hence, they are introducing a second degree of freedom. Furthermore, constraint tightening is performed to get a robustly stabilizing feedback MPC controller.

For the ancillary controller a cost function as seen in (3.12) is used to attempt to steer each state encountered towards the nominal trajectory.

$$J_N(x(t), x_c; u^*) = V_f(x(N), x_c) + \sum_{k=0}^{N-1} l(x(k|t) - z^*(t + k, x_0, x_c), u(k|t) - v^*(t + k, x_0, x_c))$$  (3.12)

In (3.12), $z^*$ denotes the nominal reference trajectory and $v^*$ the reference input sequence.

A big downside of this tube-based MPC approach is that offline tuning and validation is needed.

3.2.2 Min-Max MPC

Min-Max MPC is a well known technique to approach RMPC problems [52]. It tries to find the minimum cost with respect to the worst case disturbance while still satisfying all of the constraints. The cost function can therefore be written as in the following:

$$\min_{u(\cdot|t), \max w(\cdot|t)} J_N(x(\cdot|t), u(\cdot, t)).$$  (3.13)

It’s especially useful as reference for other approaches. However, its high computational demand prohibits a practical application for most use cases.

3.2.3 Computationally Efficient RMPC [36]

For this thesis we decided to base our RMPC design on a newer work by Koehler et al. introducing a computationally efficient tube-like RMPC control scheme [36]. It can be interpreted as an approximate extension of [12] to nonlinear systems. The related work in [19] - this thesis conceptually builds upon - bases its RMPC design and formulation onto the work in [33]. Even though the results for the special case of additive disturbances are similar, the work in [36] covers more general types of disturbances and uncertainties and offers a less conservative formulation.

The whole work is based on the assumption of local incremental stabilizability. This is given in Assumption 3.2.

**Assumption 3.2.** Local incremental stabilizability [36]: There exists a control law $\kappa(x, z, v)$, an incremental Lyapunov function $V_\delta(x, z, v)$ which is continuous in the first argument and satisfies $V_\delta(z, z, v) = 0$ for all $(z, v) \in Z$, and parameters $c_{\delta, f}$, $c_{\delta, u}$, $\delta_{locc}$, $k_{\max} > 0$, $\rho \in (0, 1)$, such that the following properties hold for all $(x, z, v) \in \mathbb{R}^N \times Z$ with $V_\delta \leq \delta$, and all $(x^+, z^+, v^+) \in \mathbb{R}^N \times Z$ with $x^+ = f(x, \kappa(x, z, v))$, $z^+ = f(z, v)$:

$$c_{\delta, f} ||x - z||^2 \leq V_\delta(x, z, v) \leq c_{\delta, u} ||x - z||^2,$$  (3.14a)

$$||\kappa(x, z, v) - v||^2 \leq \kappa_{\max} V_\delta(x, z, v),$$  (3.14b)

$$V_\delta(x^+, z^+, v^+) \leq \rho^2 V_\delta(x, z, v).$$  (3.14c)

**Problem Formulation**

The work in [36] considers nonlinear perturbed (discrete time) systems of the form

$$x(t + 1) = f_w(x(t), u(t), d(t)), \quad d \in \mathbb{D} \subset \mathbb{R}^q.$$  (3.15)

This contains general nonlinear state and input dependent disturbances or uncertainty. Only a general assumption of the effect on the system compared to the nominal system is needed, as follows:
Assumption 3.3. The perturbed system satisfies

\[ f_w(x, u, d) - f(x, u) \in W(x, u), \tag{3.16} \]

with \( W(x, u) \) being compact. Correspondingly, there exists a scalar function \( \hat{w} \) that satisfies \( |d_w| \leq \hat{w}(x, u) \) for all \( d_w \in W(x, u) \).

Note that different norm definitions can be used for the constraint on \( d_w \), e.g. the 2- or the \( \infty \)-norm. The state and input constraints are still denoted and given as defined in Def. 3.1. This RMPC scheme [36] addresses incrementally exponentially stabilizable nonlinear systems (see Assumpt. 3.2). It describes a MPC control formulation which ensures practical stability, recursive feasibility and guarantees constraint satisfaction under the general disturbance in (3.15) and Assumption 3.3.

Disturbance Description

The RMPC design could principally directly be conducted using the disturbance bound \( \hat{w} \) from Assumption 3.3. However we would then always need to bound the disturbance using Lipschitz-like constants, introducing additional conservatism. Therefore, we directly consider the influence of the disturbance on the incremental Lyapunov function \( V_\delta \). To be in a proper position to do this, the following assumption is needed.

Assumption 3.4. Consider \( \hat{w} \), stabilizing feedback \( \kappa \) and \( V_\delta \). There exists a function \( w_\delta \), s.t. for any point \( (x, z, v) \in \mathbb{R}^N \times Z \) with \( V_\delta(x, z, v) \leq c^2 \), and any \( c \in [0, \sqrt{\delta_{loc}}] \), we have

\[ \hat{w}(x, \kappa(x, z, v)) \leq w_\delta(z, v, c). \tag{3.17} \]

Furthermore, \( w_\delta \) satisfies the monotonicity property: With constants \( 0 \leq c_2 \leq c_1 \leq \sqrt{\delta_{loc}} \) and \( V_\delta(x, z, v) \leq (c_1 - c_2)^2 \), we have

\[ w_\delta(x, \kappa(x, z, v), c_2) \leq w_\delta(z, v, c_1). \tag{3.18} \]

With this it is possible to obtain the following proposition (see [36]).

Proposition 3.1. Let Assumption 3.2, 3.3 and 3.4 hold. There exists \( \tilde{w}_\delta \), s.t. for any \( (x, z, v) \in \mathbb{R}^N \times Z \), any \( (z^+, v^+) \in \mathbb{R}^N \times Z \) with \( z^+ = f(z, v) \), and any disturbance \( d_w \in W(x, \kappa(x, z, v)) \), we have

\[ V_\delta(z^+, d_w, z^+, v^+) \leq \tilde{w}_\delta^2(z, v, c). \tag{3.19} \]

Furthermore, \( \tilde{w}_\delta \) satisfies the same monotonicity property as in Assumption 3.4.

Proposition 3.1 can be shown for setting \( \tilde{w}_\delta(z, v, c) = \sqrt{\delta_{loc}} w_\delta(z, v, c) \).

General Paper Findings

The paper solves this problem formulation by

1. presenting a constraint tightening for nonlinear robust MPC based on incremental stabilizability and
2. considering general nonlinear state and input dependent disturbances by including the predicted size of the tube as scalar variables in the MPC optimization scheme.

The tube size then determines the needed degree for the constraint tightening at a given time step.
General Nonlinear Disturbances. For the general formulation the disturbance descriptions in (3.15) and in Assumpt. 3.3 are taken to find a bound \( \tilde{w}_d \) on how much the incremental Lyapunov function \( V_d \) changes in the presence of the disturbance, as seen in Prop. 3.1:

\[
V_d(z^+ + d_w, z^+, v^+) \leq \tilde{w}_d^2(z, v, c), \quad \text{for } V_d(x, z, v) \leq c^2 \quad \forall d_w \in \mathcal{W}(x, \kappa(x, z, v))
\]  

(3.20)

Additionally a factor \( c_j \) is introduced which determines to which extent the effective constraints need to be tightened according to the used incremental Lyapunov function, in order to still satisfy the actual constraints under any possible disturbance.

Assumption 3.5. \( g_j(z, v) \) is the function defining the compact nonlinear constraint set in Eqn. 3.1. We require for it to hold:

\[
g_j(x, \kappa(x, z, v)) - g_j(z, v) \leq c_j \sqrt{V_d(x, z)} \quad \forall (x, z, v) \in \mathbb{R}^N \times \mathcal{Z} \text{ with } V_d(x, z, v) \leq \delta.
\]  

(3.21)

The paper [36] proposes the robust MPC scheme in (3.22)

\[
V_N(x(t)) = \min_{u(|t|)} J_N(x(|t|), u(|t|)) \quad \text{(3.22a)}
\]

subject to

\[
x(0|t) = x(t), \quad s(0|t) = 0, \quad (3.22b)
\]

\[
x(k+1|t) = f(x(k|t), u(k|t)), \quad (3.22c)
\]

\[
s(k+1|t) = ps(k|t) + w(k|t), \quad (3.22d)
\]

\[
w(k|t) = \tilde{w}_d(x(k|t), u(k|t), s(k|t)), \quad (3.22e)
\]

\[
g_j(x(k|t), u(k|t)) + c_js(k|t) \leq 0, \quad (3.22f)
\]

\[
s(k|t) \leq \bar{s}, \quad w(k|t) \leq \tilde{w}, \quad (3.22g)
\]

\[
(x(N|t), s(N|t)) \in \mathcal{X}_f, \quad (3.22h)
\]

\[
k = 0, \ldots, N-1, \quad j = 1, \ldots, p,
\]

with \( J_N \) defined as in the following

\[
J_N(x(|t|), u(|t|)) = \sum_{k=0}^{N-1} l(x(k|t) - x_s, u(k|t) - u_s) + V_f(x(N|t) - x_s).
\]  

(3.23)

The main idea is to predict the tube size \( s \in \mathcal{R}_{>0} \) online as part of the optimization problem. The evolution of the tube size is determined by the appearing noise and the contraction parameter \( \rho \).

Assumption 3.6. There exists a terminal controller \( \kappa_f \), a terminal cost function \( V_f \), a terminal set \( \mathcal{X}_f \), and a constant \( w \) s.t. the following properties hold for all \( (x, s) \in \mathcal{X}_f \), all \( w \in [\tilde{w}_{\text{min}}, \tilde{w}] \) with \( \tilde{w}_{\text{min}} := \inf_{(x,w) \in \mathcal{Z}} \tilde{w}_d(x,u,0) \), all \( s^+ \in [0, ps - \rho s + \tilde{w}_d(x, \kappa_f(x)), s] \), and all \( d_w \) s.t. \( V_d(x^+ + d_w, x^+, \kappa_f(x^+)) \leq \rho^2 N w^2 \):

\[
V_f(x^+) \leq V_f(x) - l(x, \kappa_f(x)), \quad (3.24)
\]

\[
(x^+ + d_w, s^+) \in \mathcal{X}_f, \quad (3.25)
\]

\[
\tilde{w}_d(x, \kappa_f(x), s) \leq \tilde{w}, \quad (3.26)
\]

\[
g_j(x, \kappa_f(x)) + c_j s \leq 0, \quad (3.27)
\]

\[
s \leq \bar{s}, \quad (3.28)
\]

\[
\text{with } x^+ = f(x, \kappa_f(x)). \quad (3.29)
\]

Furthermore, the terminal set cost \( V_f \) is continuous on the compact set \( \mathcal{X}_{f,x} := \{ x|3s \in [0, s], (x,s) \in \mathcal{X}_f \} \), i.e., there exists a function \( \alpha_f \in K_\infty \), s.t.

\[
V_f(z) \leq V_f(x) + \alpha_f(||x - z||), \forall x, z \in \mathcal{X}_{f,x}.
\]  

(3.30)

With Assumption 3.6, in [36] it is shown that this RMPC formulation in (3.22) is recursively feasible, the constraints are satisfied and the origin is practically asymptotically stable for the resulting closed-loop system under practically relevant assumptions. For more details see [36].
Additive Disturbances In this work we only do RMPC design assuming and considering additive disturbances. This drastically reduces the complexity of the problem - not only for the MPC formulation itself but also for the calculation of the required parameters, e.g. the description of $W$. For this simpler special case, the tube size $s$ can be explicitly written by using the formula for the geometric sum as:

$$s(k) = \frac{1 - \rho^k}{1 - \rho} \bar{w}, \quad (3.31)$$

with

$$\bar{w} := \sup_{z,v,v^+,d_w} \sqrt{V_\delta(f(z,v) + d_w, f(z,v), v^+)}, \quad (3.32)$$

$$s.t. (z,v) \in \mathcal{Z}, (f(z,v), v^+) \in \mathcal{Z}, d_w \in \mathcal{W}(z,v)). \quad (3.33)$$

This then leads to the following simplified RMPC formulation given in (3.34):

$$V_N(x(t)) = \min_{u(\cdot|t)} J_N(x(\cdot|t), u(\cdot|t)) \quad (3.34a)$$

subject to

$$x(0|t) = x(t), \quad (3.34b)$$

$$x(k+1|t) = f(x(k|t), u(k|t)), \quad (3.34c)$$

$$g_j(x(k|t), u(k|t)) + c_j \frac{1 - \rho^k}{1 - \rho} \bar{w} \leq 0, \quad (3.34d)$$

$$x(N|t) \in \mathcal{X}_f, \quad k = 0, \ldots, N - 1, \quad j = 1, \ldots, p. \quad (3.34e)$$

Also the assumption for the terminal ingredients (compare Assumption 3.6) for guaranteeing stability now becomes simpler:

**Assumption 3.7.** There exists a terminal controller $\kappa_f$, a terminal cost function $V_f$, a terminal set $\mathcal{X}_f$, s.t. the following properties hold for any $x \in \mathcal{X}_f$ and all $d_w$ s.t. $V_\delta(x^+ + d_w, x^+, \kappa_f(x^+)) \leq \rho^N \bar{w}^2$:

$$V_f(x^+) \leq V_f(x) - l(x, \kappa_f(x)), \quad (3.35)$$

$$x^+ + d_w \in \mathcal{X}_f, \quad (3.36)$$

$$\frac{1 - \rho^N}{1 - \rho} \bar{w} \leq \bar{s} = \sqrt{\delta_{loc}}, \quad (3.37)$$

$$g_j(x, \kappa_f(x)) + c_j \frac{1 - \rho^N}{1 - \rho} \bar{w} \leq 0, \quad j = 1, \ldots, p, \quad (3.38)$$

$$with \ x^+ = f(x, \kappa_f(x)). \quad (3.39)$$

Furthermore, there exists a function $\alpha_f \in \mathcal{K}_\infty$, s.t.

$$V_f(z) \leq V_f(x) + \alpha_f(||x - z||), \forall x, z \in \mathcal{X}_f. \quad (3.40)$$

Describing the disturbances as additive ones reduces the complexity of the optimization problem and makes it equivalent to the corresponding nominal MPC scheme. The main disadvantage of it is the potential of introducing a lot of conservatism.

### 3.3 Learning - Theory

Machine learning (ML) is a tool which was successfully deployed in many different tasks in the last decades. In the last few years it attracted even more attention due to the recent successes in deep learning. Particularly impressive successes, e.g. in image processing, natural language
processing and reinforcement learning, could be achieved thanks to access to a significant increase in computational resources.

 Usually, the field of ML is divided in the following three subcategories: supervised learning, unsupervised learning and reinforcement learning. In our work we will only make use of supervised learning techniques; regression and classification. Therefore we will only give an introduction of the latter ones in the following. For each of the topics, we will only give a brief account in the following.

### 3.3.1 Supervised Learning

Supervised learning is a technique for finding a function $f(x)$ by providing a set of training examples $\mathcal{X}$ with $p$ samples containing corresponding targets $y^{(i)}, i = 1,..., p$ of the function $f(x)$. For approximating these functions, very often neural networks (NN) are used. NNs are trained by using back-propagation. This is done by inserting samples to the network, calculating its output at the current state of the network, computing the error $e$ to the desired value the network should show at its output and then doing gradient descent on the weights of the activation functions in the network nodes.

Recent big successes were mainly possible thanks to deep learning techniques; to put it simply, this means that the depth of the network is increased in the hope to achieve good results. Nowadays with the help of modern graphic processing units (GPUs) it become possible to also train those big models in tractable amounts of time. Nowadays the most popular choice for the optimizer of the training (which is e.g. responsible for finding the learning rate) is the ADAM optimizer [29]. As activation functions for deep networks usually rectified linear units (ReLU) functions are chosen, since they behave advantageous with respect to problems of training networks, such as the vanishing gradient problem.

**Supervised Regression**

Supervised regression denotes the special case of the previous introduction to supervised learning when the function $f(x)$ denotes a continuous valued function. This means, we really try to find a continuous function which maps values of a specific domain into its range. The corresponding neural networks are usually built by putting a linear activation at the output layer.

**Supervised Classification**

Classification is the problem of matching data points to classes, i.e. the function $f(x)$ has discretized (but possibly infinitely many) possible values. Since differentiable activation functions need to be used in the network, the output will still take continuous values. To obtain the desired classes, then a threshold is taken of this output. The normal choice for the activation of the output layer for classification tasks is usually a soft-max function.

### 3.3.2 Ability to Overcome the Curse of Dimensionality

In our practical example in Chapter 8 we need to be able to deal with relatively high input dimensions. E.g. already for simplest case of tracking an output reference, we need an input size of 7 or 11 for the case of using the simplified or the full kinematic formulation, respectively. By regarding the sampling of this formulation as a sampling of a multi-dimensional grid, this means that we are only able to sample significantly less than 10 points per dimension. We will do some more practical computations for our robotic use case in Section 9.2.3.

The neural networks used in our case might still be able to perform well in approximating the desired function, since they often have an impressive ability of projecting high dimensional problems to lower dimensional manifolds. To make this even more effective, it is usually necessary to make assumptions about the data and to exploit them in the models. A very good example for this are convolutional neural networks (CNNs), see [17], which e.g. assume important spatial relations of neighboring pixels in image processing.
3.3.3 Statistical Learning

Statistical learning theory represents a statistics-based method to investigate ML algorithms and is often able to unveil additional, possibly hidden, information.

Standard literature such as [18] describe the main concepts. However, since in this work we are mainly interested in validating the performance of our neural network, we want to make statistical statements about the reliability of our learned policy. Several inequalities are known for providing an upper bound to the probability for a specific deviation of the observed mean value. Examples are the Bienayme-Chebychev inequality and Chebyshev’s inequality. In this work we will make use of the so called Hoeffding inequality [20]. In general, this inequality is useful, since it provides an upper probability bound to the scenario of having divergence of the real mean value of a distribution compared to the one observed from experiments. The corresponding inequality is given in the following:

\[
\mathbb{P} \left[ |\tilde{\mu} - \mu| \geq \epsilon_h \right] \leq 2e^{-2p\epsilon_h^2}.
\]

In this case, \( \tilde{\mu} \) denotes the empirical risk, i.e. in our case the observed mean value of the distribution of a collection of random variables, while \( \mu \) is the wanted mean value. To make this statement valid, \( p \) independent and identically distributed (i.i.d.) random variables need to be observed. \( \epsilon_h \) describes the difference in the mean value of the observed and the real mean value. We will go into more detail about this for our specific use case in Section 5.3.2.

A comprehensive overview of concepts in statistical learning theory is also given in [61], which also explains Hoeffding’s inequality.
Chapter 4

RMPC Design

For the RMPC design we use the theory from Sec. 3.2.3. In the following we will first show the used and simplified setup of our MPC design. Then in Section 4.2 we will give an high level overview of how the RMPC design can be done. Afterwards, we will give more insights into the computation of the sets, values and parameters needed for getting a proper RMPC controller according to the paper in [36]. We will also introduce the CT-equivalent parameters, to do the computations independently from the specific sampling time used in the real-time implementation.

4.1 Setup

Quadratic Incremental Lyapunov Function

For the incremental Lyapunov function $V_\delta$, a quadratic function is chosen, i.e.

$$V_\delta(x, z) = |x - z|^2_{P_\delta}. \quad (4.1)$$

For this special case, Proposition 4.1 provides a direct method to compute $\tilde{w}_\delta$ for general nonlinear disturbances.

**Proposition 4.1.** Let Assumption 3.3 hold. Now, suppose that Assumption 3.2 is satisfied with $V_\delta(x, z, v) = |x - z|_{P_\delta}$. Then Assumption 3.4 is satisfied with

$$w_\delta(z, v, c) := \sup_{\{x||x - z|_{P_\delta} \leq c\}} \tilde{w}(x, \kappa(x, z, v)). \quad (4.2)$$

Furthermore, the conditions in Proposition 3.1 are satisfied with

$$\tilde{w}_\delta(z, v, c) := \sup_{\{x||x - z|_{P_\delta} \leq c\}} \sup_{d_w \in W(x, \kappa(x, z, v))} |d_w|_{P_\delta}. \quad (4.3)$$

This Proposition uses directly the shape $V_\delta$ and $W$ to compute $\tilde{w}_\delta$, thus leads to less conservative bounds than many other approaches. In the following we will present this for additive disturbances.

Additive Disturbance

As mentioned earlier we will only consider additive disturbances. By doing so, the formulation in Equation 3.22 which contains the tube size as an additional state in the optimization problem simplifies to the form seen in Equation 3.34. The additive disturbance modifies the function $f_w$ to the following:

$$f_w(x, u, d) = f(x, u) + d_w. \quad (4.4)$$
With this formulation and a quadratic incremental Lyapunov function, the Proposition 4.1 simplifies even further to the one in Equation 4.5:

$$\bar{w} := \sup_{z,v,d_w} \sqrt{\left| (f(z,v) + d_w) - f(z,v) \right|^2_{P_{\bar{w}}}}$$

subject to:

$$s.t. (z,v) \in \mathcal{Z}, (f(z,v), v^+) \in \mathcal{Z}, d_w \in \mathcal{W}(z,v).$$

Quadratic Stage Cost

For all of our MPC designs in this thesis we use a quadratic stage cost $l(x,u)$. This means that

$$l(x,u) = |x - x_c|^2_Q + |u - u_c|^2_R.$$

Polytopic State Constraints

For the state and input we consider polytopic constraints, i.e. constraints of the form

$$L_{j,x}x + L_{j,u}u \leq 1, \quad j = 1, ..., r.$$ (4.8)

Nonlinear Output Constraints and Collision Avoidance

For being able to deal with the nonlinear output constraints and avoiding obstacles we deal with the general formulation for constraints (see Definition 3.1) on the output:

$$g_j(x,u) \leq 0, \quad j = r + 1, ..., p.$$ (4.9)

4.2 Procedure

Our presented RMPC approach mainly consists of two bigger parts; performing the offline computations as well as doing the online computations. In this section we will give an overview of the required components. Then later on in this chapter, we will go into detail for all of the required components.

4.2.1 Algorithm for Offline Computation

For the algorithm for the needed offline calculations we refer to Algorithm 1. In the very beginning two main tuning parameters are given; the CT contraction rate $\rho_c$ and the effect of the disturbance on the incremental Lyapunov function $\bar{w}_c$. Additionally it’s possible to set the maximum value of $c_j$, e.g. to 1. In a next step these conditions need to be formulated in LMIs. Using these LMIs as constraints, the optimal $P_{\bar{w}}$ and $K_{\bar{w}}$ can be found by maximizing the determinant of $X$, i.e. minimizing the determinant of $P_{\bar{w}}$. After computing or checking the values for $c_j$ and $\bar{w}_c$ respectively, we can then compute required values for the terminal ingredients.

Details on where Algorithm 1 comes from and how the steps are exactly performed can be seen in the following sections.

4.2.2 Algorithm for Online Computation

After having calculated important values offline, we will now shortly explain how the actual online implementation is done. This can be seen in more detail in Algorithm 2. The modified optimization problem, potentially covering dynamic set point tracking, needs to be solved. After the solution has been obtained, we can then apply it to the system in addition to the prestabilization-input which is applied to the system in (almost) real-time.
Algorithm 1 Offline Calculations for RMPC Design
1: Define $\rho_c$ and potentially $\bar{w}_c$ and $c_j$.
2: Write LMIsto enforce constraints on the resulting $P_\delta$ and $K_\delta$. The constraint on the contraction in Assumption 4.2 (more specifically in (4.15)) is needed. Constraints on $\bar{w}_c$, state constraints, input constraints and output constraints are optional but possibly result in a less conservative solution for the given circumstances. For formulating the LMIs we use the standard substitution $X = P_\delta^{-1}$, $Y = KX$ and the Schur complement.
3: Minimize $-\log(\det(X))$ with respect to the LMIs, to get solutions for $P_\delta$ and $K_\delta$.
4: Compute $c_j$, $\forall j = 1, \ldots, p$ (see (4.81) and (4.83)) or check whether constraint from LMI still holds.
5: Check whether $\rho_c$ satisfies the conditions (by using griding approach).
6: Compute $w_c$ or check whether constraint from LMI still holds.
7: Compute the terminal ingredients: terminal cost (see Section 4.4.1) and compute $s_f$ for the terminal set (Section 4.4.2).
8: Check whether Lyapunov descent condition is fulfilled for the terminal cost within the terminal set.
9: Finally compute the DT parameters $\rho$ and $w$ from $\rho_c$ and $w_c$ respectively, using the given sampling time.

Algorithm 2 Online Calculations, execute at every time step $k \in \mathbb{N}$ during the sampling time interval of length $T$ starting at $t$.
1: Solve the MPC problem from (4.53).
2: Apply the prestabilized input $u(t + \tau) = v^*_\text{MPC}(0|t) + K_\delta \cdot x(\tau|t), \tau \in [0, T]$ to the system (Section 4.3).

4.3 Computations in Continuous Time

In the following we will explain how we make use of the real-time behavior of our robotic control framework (Section 6.4). As you will see, we apply real-time prestabilizing feedback which then allows us to do the offline calculations in CT.

Prestabilization Different to the typical nominal MPC formulation where the input $v_{\text{mpc}}(0|t)$ is directly applied to the system, prestabilized dynamics can be used. This means and has the big advantage that the system is stabilized in real-time along the horizon by a (nearly continuous) feedback law $\kappa(x)$ additionally to the applied piece-wise constant MPC control input. The dynamics can then be reformulated as

$$
\dot{x} = f_\kappa(x, v_{\text{mpc}}) = f(x, v_{\text{mpc}} + K_\delta \cdot x), \quad g_{j,\kappa}(x, v_{\text{mpc}}) = g(x, v_{\text{mpc}} + K_\delta \cdot x),
$$

(4.10)

where $g_{j,\kappa}$ is the function defining the compact nonlinear constraint set defined in Def. 3.1. We therefore have the real system input as

$$
u(t + \tau) = v^*_\text{MPC}(0|t) + K_\delta \cdot x(\tau|t)), \tau \in [0, T],
$$

(4.11)

in the sampling interval of length $T$ starting at $t$. This system input is then also used in the stage cost $l(x, u(t), x_s, u_s)$.

Offline Calculations
Assumption 4.1. We can implement the CT feedback $\kappa(x(t)) = K_\delta \cdot x(t)$.

Under Assumpt. 4.1, we are doing the RMPC design parameter computation in CT to be independent of the actual sampling time. This is also new in comparison to the work in [19]. We will
introduce the most important CT constants and design parameters in the following. According to (3.15), we use the following definition for our perturbed system:

\[
\dot{x} = f(x,u,d), \quad (x,u) \in Z, \quad d \in D.
\]  

(4.12)

The general bound on the disturbance remains the same as denoted in Assumption 3.3. By going to CT, the criteria for incremental stability from Assumpt. 3.2 becomes the following:

**Assumption 4.2. Incremental stabilizability in CT:** In analogy to Assumption 3.2. There exists a local control law \( \kappa(x,z,v) \) for the incremental Lyapunov function from (4.1), s.t. the following holds:

\[
\frac{d}{dt} V_\delta(x,z) = \frac{\partial V_\delta}{\partial x}|_{x,z} f(x,v) + \frac{\partial V_\delta}{\partial z}|_{x,z} f(z,v) \leq -2 \rho_c V_\delta(x,z) + 2 \bar{w}_\delta(z,v,c) \sqrt{V_\delta(x,z)}.
\]  

(4.13)

\[
\frac{\partial V_\delta}{\partial x}|_{x,z} f(x,v) + \frac{\partial V_\delta}{\partial z}|_{x,z} f(z,v) = 2(f(x,\kappa(x,z,v)) - f(z,v))^T P(x-z) \leq -2 \rho_c V_\delta(x,z),
\]  

(4.15)

\[
\frac{\partial V_\delta}{\partial x}|_{x,z} d_w \leq 2 \bar{w}_\delta(z,v,c) \sqrt{V_\delta(x,z)}.
\]  

(4.16)

\( \rho_c \) in this case is the continuous equivalent of \( \rho \). They can be transformed when knowing the sampling time by using the following relation:

\[
\rho = e^{-\rho_c h}.
\]  

(4.17)

Using the description in Equation 4.1 for \( V_\delta \), for the continuous disturbance bound we get the following:

\[
\bar{w}_c := \sup_{z,v,d_w} \sqrt{V_\delta(f(z,v) + d_w, f(z,v))} = \sup_{z,v,d} \sqrt{||f_w(z,v,d) - f(z,v)||_2^2}.
\]  

(4.18)

The tube size also becomes a continuous description (compare to Equation 3.31):

\[
s(t) = \frac{1 - e^{-\rho_c t}}{\rho_c} \bar{w}_c, \text{ equivalent to DT with } \bar{w} = s(h) = \frac{1 - e^{-\rho_c h}}{\rho_c} \bar{w}_c.
\]  

(4.19)

With the CT stabilizing feedback (Section 4.3) and incremental Lyapunov function, we can now apply the DT RMPC scheme according to Equation 4.19.

### 4.4 Terminal Ingredients

For the RMPC design we also need appropriate terminal ingredients, i.e. \((V_f, \lambda_f, \kappa_f, \bar{w})\). In contrast to the nominal case (compare Sec. 3.1.1), this time also the tightened constraints need to be satisfied for the terminal ingredients. We need to find a robust positively invariant (RPI) terminal set. For more information on set theory in control we refer to [7]. For the general approach the terminal ingredients need to satisfy Assumption 3.6. However, we limit ourselves to the case of additive disturbances and therefore need to satisfy the simpler Assumption 3.7. First we will show how to compute the terminal cost \( V_f \) and its corresponding local terminal controller. Then we will show two ways of constructing the terminal set.

#### 4.4.1 Terminal Cost - Literature Review

In this section, we will shortly explain methods for computing the terminal cost known in the literature. This covers linear and nonlinear system dynamics as well as constant and varying set points.
4.4. Terminal Ingredients

**Linear System Dynamics**

Computing the terminal cost for linear system dynamics is the simplest case. The terminal cost has the task to bound the infinite horizon closed-loop cost of the MPC controller. Therefore, we can simply use the infinite horizon cost of the LQR controller (compare Sec. 3.1.1). We only need to make sure that state and input constraints are satisfied within the terminal set.

**Nonlinear System with Constant Set Point**

For the problem of steering nonlinear system dynamics with quadratic stage cost, quadratic terminal cost $V_f(x) = ||x||_P^2$ to a preknown set point, we can compute a terminal set that locally satisfies Assumption 3.7. This can be done using standard methods [10, 53].

Without loss of generality, we consider the Jacobi linearization of the system around the set point $x_s = 0, u_s = 0$:

$$
\dot{x} = Ax + Bu, \quad A := \frac{\partial f}{\partial x}|_{0,0}, B := \frac{\partial f}{\partial u}|_{0,0}.
$$

If $(A, B)$ Equation 4.20 is stabilizable, then we can find a stabilizing (linear) terminal controller s.t. $A_K = A + BK$ is asymptotically stable. In [10] it is shown that the Lyapunov Equation with $Q^*$ being positive-definite and symmetric and $\kappa \in [0, -\lambda_{\text{max}}(A_K)]$:

$$
(A_K + \kappa I)^TP + P(A_K + \kappa I) = -Q^*, \quad Q^* = Q + K^TRK,
$$

admits a unique positive definite and symmetric solution $P$. The infinite horizon cost $J^\infty(x, u)$ for any $x$ at time $t$ lying in the terminal set is then bounded from above by

$$
J^\infty(x, u) = \int_t^\infty ||x(t)||_Q^2 + ||u(t)||_R^2 dt \leq x^TPx.
$$

The value function $V(x)$ is given as in the following:

$$
V(x) = J(x, t, t + T_p),
$$

Chen and Allgoewer show the stability property of this cost function

$$
V(x(t + \delta)) - V(x(t)) \leq \int_t^{t+\delta} l(x(\tau), u(\tau))d\tau \leq -\delta \gamma,
$$

with $\gamma > 0$.

We only need to make sure that the state and input constraints are not violated within the terminal set and that the terminal cost $V_f$ is a control Lyapunov function. If this does not hold, the terminal set needs to be shrunk. The latter can be checked using the following condition (see [10]):

$$
x^TP(f(x, Kx) - A_Kx) \leq \kappa x^TPx.
$$

In our case we choose the LQR controller as feedback controller and get the terminal cost as

$$
V_f(x(N|t)) = ||x(N|t)||_P^2.
$$

**Nonlinear Systems with Varying Set Points**

For general nonlinear systems and variable set points, the computation is more difficult.

For a simple approach using a LTV description of the nonlinear system close to the steady-state manifold, we refer to [65, 64]. More general, in [43] the steady-state manifold is split into different intervals and for each interval a constant $P, K$ are computed for each interval using a description of LTV systems. Alternatively, in [32] the terminal ingredients $P, K$ are continuously parameterized and computed using quasi-LPV methods. Since we are not using this approach, we won’t show these techniques in more detail.
Chapter 4. RMPC Design 25

4.4.2 Terminal Set - New Approach Based on Incremental Lyapunov Function

Since we need to be able to do robust set point tracking with varying set points, we need to find a suitable, non conservative terminal set for our online computations. For this, we got inspired by the work in [36] which uses the incremental Lyapunov function to find a terminal set for constant set points.

Constant Set Point

In [36] a Proposition states that by using

\[ \mathcal{X}_f = \{ (x,s) \in \mathbb{R}^{n+1} | s \in [0, \bar{s}_f], V_f(x) \leq \gamma_1 \} \]  \hspace{1cm} (4.27)

as the terminal set, Assumption 3.7 is satisfied for specific \( \gamma_1 \). As we’ve seen before we can calculate the terminal cost and the terminal controller using standard methods [10]. For the computation of the terminal set, i.e. \( \gamma_1 \), the maximal tube size \( \bar{s}_f \) is needed to make sure that the tightened constraints are satisfied within the terminal set. The required conditions for \( \gamma_1 \) are described in [36]. However, since we need to handle varying set points which are unknown beforehand, we are not able to use this approach and won’t describe it in more detail here.

New Approach for Varying Set Points

We now want to compute a terminal set for varying set points. Similar work was done in [35]. Since we do not know the desired set point in advance and therefore can not simply compute a non-conservative terminal set size for every offline point, the idea of our approach here is to include the size of the terminal region in the optimization problem. This approach differs from existing approaches in the following ways:

1. Our terminal set are sublevel sets of \( V_\delta \) which significantly simplifies the conditions on the terminal set. The idea of using sublevel sets of \( V_\delta \) was introduced in [30].
2. We optimize the terminal set size online, i.e. it is part of the optimization problem, compare [35].

To the best of our knowledge this has not been done before for the cases we cover here.

Notation

Let us denote the non-constant set point in state and input as \( x_s \) and \( u_s \), respectively.

We consider a terminal set as a function of \( x_s \) and \( u_s \) of the following form:

\[ \mathcal{X}_f(x_s, u_s) := \{ x | V_\delta(x, x_s) \leq \gamma_2(x_s, u_s) \} \]  \hspace{1cm} (4.28)

Let us introduce the following notation:

\[ Z_k := \{ (x, u) | g_j(x, u) + c_j s(k) \leq 0 \}, \text{ with } s(k) = \frac{1 - \rho^k}{1 - \rho}, \]  \hspace{1cm} (4.29)

\[ c_{\text{max}} = \max_j c_j, \]  \hspace{1cm} (4.30)

\[ \bar{s}_f = \frac{1 - \rho^N}{1 - \rho} \bar{w}. \]  \hspace{1cm} (4.31)

Requirements for Terminal Set

For a proper terminal set the following conditions need to be satisfied:

1. The terminal set needs to be robust positively invariant (RPI) for a given set point \( (x_s, u_s) \), i.e. once a trajectory entered the RPI set, it will never leave it again. This can be formulated as:

\[ x \in \mathcal{X}_f(x_s, u_s) \Rightarrow f(x, \kappa(x, x_s, u_s)) + d_w \in \mathcal{X}_f(x_s, u_s), \forall |d_w| p_3 \leq \rho^N \bar{w}. \]  \hspace{1cm} (4.32)
2. The tightened constraints are satisfied within the terminal set, i.e.
\[(x, \alpha(x, x_s, u_s)) \in \mathcal{N}, \quad \forall x \in \mathcal{X}_f(x_s, u_s).\] (4.33)

3. The terminal cost is a continuous Lyapunov function in the terminal set \(\mathcal{X}_f\) satisfying
\[V_f(f(x, \alpha(x, x_s, u_s)), x_s) - V_f(x, x_s) \leq l(x, \alpha(x, x_s, u_s), x_s, u_s), \quad \forall x \in \mathcal{X}_f(x_s, u_s).\] (4.34)

**Approach** We choose the following ansatz for the terminal set as defined in Equation 4.28 including the tube size at the end of the horizon \(\bar{s}_f\) to write the following formulation with respect to the set point \((x_s, u_s)\):
\[\mathcal{X}_f(x_s, u_s) := \{x | \sqrt{v_b(x, x_s)} + \bar{s}_f \leq \alpha(x_s, u_s)\} = \{x | |x - x_s|_{P_b} + \bar{s}_f \leq \alpha(x_s, u_s)\}.\] (4.35)

Together with the explicit formulation of the tube size for additive disturbances from Equation 3.31 this results in the following:
\[\mathcal{X}_f(x_s, u_s) := \{x | |x - x_s|_{P_b} \leq \alpha(x_s, u_s) - \frac{1 - \rho^N}{1 - \rho} \bar{w}\}.\] (4.36)

**Derivation** In the following we will derive the conditions for \(\alpha(x_s, u_s)\) for Equation 4.35 being a proper terminal set. For this the 3 criteria defined before need to be satisfied. Due to our simplifying assumptions it holds that \(V_b(x, x_s) = |x - x_s|^2_{P_b}\).

1. The set needs to be RPI. It holds \(x \in \mathcal{X}_f\):
\[\max_{d_w} |(x^+ + d_w) - x_s|_{P_b} + \bar{s}_f^+ \leq \max_{d_w} \left| x^+ - x_s \right|_{P_b} + \max \left| d_w \right|_{P_b} + \bar{s}_f^+.\] (4.37)

\[\text{Triangle Inequality} \leq |x^+ - x_s|_{P_b} + \max_{d_w} |d_w|_{P_b} + \bar{s}_f^+.\] (4.38)

\[\text{Ass. 3.7} \leq |x^+ - x_s|_{P_b} + \rho^N \bar{w} + \bar{s}_f^+.\] (4.39)

\[\text{Def. of } s(k+1) \leq |x^+ - x_s|_{P_b} + \rho^N \bar{w} + \rho \bar{s}_f + \bar{w} - \rho^N \bar{w} \leq \rho |x - x_s|_{P_b} + \rho \bar{s}_f + \bar{w} \leq \rho \alpha + \bar{w} \leq \alpha.\] (4.40)

\[\Rightarrow \frac{\bar{w}}{1 - \rho} \leq \alpha.\] (4.41)

For this \(\alpha\) value, the terminal set is RPI.

2. All states in the terminal set need to satisfy the tightened constraints, i.e.
\[g_j(x, u) + c_j \bar{s}_f \leq 0, \quad \forall |x - x_s|_{P_b} \leq \alpha.\] (4.42)

This also strongly depends on the set point \(x_s\):
\[g_j(x, u) + c_j \bar{s}_f \leq g_j(x_s, u_s) + c_j \sqrt{v_b(x, x_s)} + c_j \bar{s}_f \leq g_j(x_s, u_s) + c_j \alpha \leq 0 \Rightarrow \alpha \leq -\frac{g_j(x_s, u_s)}{c_j}.\] (4.43)

3. The terminal cost can be calculated as seen before. For the special case of linear systems, it’s simply the infinite horizon cost of the corresponding LQR controller.
**Resulting Conditions**  Therefore we have the two conditions on \( \alpha \), again summarized here:

\[
\frac{\bar{w}}{1 - \rho} \leq \alpha \leq -\frac{g_j(x_s, u_s)}{c_j}, \quad \forall j = 1, ..., p.
\] (4.48)

The RPI condition provides a lower bound on alpha and therefore requires a sufficiently large terminal set, whereas the constraint satisfaction bounds the size of the terminal set from above.

**Remark.** These constraints implicitly constrain the set of set points \( x_s \) to the ones that are feasible.

**Formulation for the Optimization Problem**  As we want our terminal set to be as large as possible we always aim for the maximum admissible size. Therefore, the optimal value for \( \alpha \) would be:

\[
\alpha(x_s, u_s) = \max_j -\frac{g_j(x_s, u_s)}{c_j}.
\] (4.49)

We can include the terminal set as defined in Equation 4.28 with \( \gamma_2 = (\alpha - \bar{s}_j)^2 \). However, since we know \( (x_s, u_s) \) only after the optimization, we need to include the bounds on alpha in the optimization problem. The terminal constraint set therefore becomes:

\[
X_f(x_s, u_s) := \{ x | \sqrt{V_0(x, x_s, v)} + \bar{s}_j = |x - x_s|_{p_j} + \bar{s}_j \leq \alpha \}
\] (4.50)

\[
\Rightarrow X_f(x_s, u_s) := \{ x | V_0(x, x_s, v) = |x - x_s|_{p_j} \leq \gamma_2 = (\alpha - \bar{s}_j)^2, \}
\] (4.51)

with \( \frac{\bar{w}}{1 - \rho} \leq \alpha \leq -\frac{g_j(x_s, u_s)}{c_j}, \quad \forall j = 1, ..., p. \) (4.52)

### 4.5 Summary - Resulting Optimization Problem

#### 4.5.1 Constant Set Point

The RMPC formulation with suitable terminal ingredients for the simpler case of static ingredients is exactly the one from (3.34). We use this formulation for our simulation example in Chapter 7.

#### 4.5.2 Varying Set Point

The resulting optimization problem including the set point reference tracking, the output constraints as well as our special terminal ingredients can be found in the following:

\[
V_N(x(t), y_t) = \min_{u(\cdot|t), x_s, u_s, \alpha} J_N(x, y_t; u(\cdot|t), y_s, x_s, u_s, \alpha)
\] (4.53a)

subject to

\[
x(0|t) = x(t),
\] (4.53b)

\[
x(k + 1|t) = f(x(k|t), u(k|t)),
\] (4.53c)

\[
g_j(x(k|t), u(k|t)) + c_j \frac{1 - \rho^k}{1 - \rho} \bar{w} \leq 0,
\] (4.53d)

\[
x_s = f(x_s, u_s), \quad y_s = h(x_s, u_s),
\] (4.53e)

\[
\frac{\bar{w}}{1 - \rho} \leq \alpha \leq -\frac{g_j(x_s, u_s)}{c_j}, \quad \forall j = 1, ..., p
\] (4.53f)

\[
|x(N|t) - x_s|_{p_j} \leq (\alpha - \bar{s}_f)^2,
\] (4.53g)

\[
k = 0, ..., N - 1, \quad j = 1, ..., p,
\]

with \( J_N \) as introduced in Section 3.1.2 and repeated here:

\[
J_N(x, y_t; u(\cdot|t), y_s, x_s, u_s) = \sum_{k=0}^{N-1} l(x(k|t) - x_s, u(k|t) - u_s) + V_f(x(N|t) - x_s, y_s) + V_O(y_s - y_t).
\] (4.54)
4.6 Derivations and Computations of Required Parameters - A Comprehensive Collection

In this section we will now present the required computations needed to be in a position to do the calculations for the algorithms introduced in 4.2.

4.6.1 Incremental Lyapunov Function

Our goal is to compute \( P_\delta \) and \( K_\delta \) for the incremental Lyapunov function \( V_\delta(x, z) = \| x - z \|^2 P_\delta \) and \( K_\delta \) applicable as feedback for getting prestabilized dynamics. These matrices can be found by solving a minimization problem for \(-\log(\det(X))\), where \( X = P_\delta^{-1} \) under LMI conditions.

Formulating Suitable Conditions

For theory on LMIs in system theory refer to [8] and for a similar use case as ours refer to [32, 31]. It’s beneficial to include pre-known constraints into the LMIs needed for computing \( P_\delta \) and \( K_\delta \), since this allows to choose less conservative parameters. For the incremental Lyapunov function we know conditions which should hold:

1. From Ass. 4.2 and specifically Equation 4.15 we get the requirement for the incremental Lyapunov function:

   \[
   2(f(x, \kappa(x, z, v)) - f(z, v))^T P_\delta (x - z) \leq -2\rho_c V_\delta(x, z). \tag{4.55}
   \]

   For a linear system with linear state feedback this becomes

   \[\iff 2(A(x - z) + BK_\delta(x - z))^T P_\delta(x - z) \leq -2\rho_c V_\delta(x, z)\]

   \[\iff \Delta x^T (A + BK_\delta) P_\delta + P_\delta (A + BK_\delta))^T \Delta x \leq -2\rho_c \Delta x^T P_\delta \Delta x\]

   \[\iff (A + BK_\delta)^T P_\delta + P_\delta (A + BK_\delta) + 2\rho_c P_\delta \leq 0. \tag{4.59}\]

2. From Ass. 4.2 and specifically Equation 4.16 we get the requirement for the incremental Lyapunov function for the infinity norm bound on the disturbance \( d_w \):

   \[\frac{\partial V_\delta}{\partial x}|_{x, z} d_w \leq 2\bar{w}_c (z, v, c) \sqrt{V_\delta(x, z)} \tag{4.60}\]

   \[\iff \frac{\partial}{\partial x} (x - z)^T P_\delta (x - z) d_w = 2(x - z)^T P_\delta d_w \leq 2\bar{w}_c |x - z| P_\delta \tag{4.61}\]

   \( P_\delta \) pos. def. and symm.

   \[\iff (x - z)^T P_\delta^{1/2} P_\delta^{1/2} d_w = (P_\delta^{1/2} (x - z))^T P_\delta^{1/2} d_w \tag{4.62}\]

   Cauchy-Schwarz Ineq.

   \[\leq |P_\delta^{1/2} (x - z)|_2 \cdot |P_\delta^{1/2} d_w|_2 = |x - z| P_\delta \cdot |d_w| P_\delta \leq \bar{w}_c |x - z| P_\delta \tag{4.63}\]

   \[\iff |d_w| P_\delta \leq \bar{w}_c. \tag{4.64}\]

3. Since we have constraints on the states and inputs and potentially on the output function, it makes sense to also include those in the finding process of \( P_\delta \) and \( K_\delta \) to potentially get a less conservative solution for our problem. By doing this we can limit the values of the
corresponding $c_j$ parameters, $\forall (z, v) \in Z$ and $\forall r \in Z$:

$$g_j(x, \kappa(x, z, v)) - g_j(z, v) \leq c_j \sqrt{V_\delta(x, z)} = c_j |x - z|_{P_\delta} \text{ from (3.21)}$$  \hspace{1cm} (4.65)

$$\Leftrightarrow g_j(x, \kappa(x, z, v)) - g_j(z, v) \Leftrightarrow \Delta x \leq c_j |\Delta x|_{P_\delta}$$  \hspace{1cm} (4.66)

and with $|\Delta x|_{P_\delta} = \sqrt{\Delta x^TP_\delta^{1/2}\Delta x} = \sqrt{(P_\delta^{1/2}\Delta x)^T(P_\delta^{1/2}\Delta x)} = |P_\delta^{1/2}\Delta x|_2$$  \hspace{1cm} (4.67)

$$\Leftrightarrow \left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) \Delta x \leq c_j |P_\delta^{1/2}\Delta x|_2$$  \hspace{1cm} (4.68)

$$\Delta x = \frac{P_\delta^{1/2}\Delta x}{\Delta \tilde{x}} \leq \frac{|\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r P_\delta^{-1/2} \Delta \tilde{x}$$  \hspace{1cm} (4.69)

$$\leq \left|\left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) P_\delta^{-1/2} \Delta \tilde{x}\right|_2 \leq c_j |\Delta \tilde{x}|_2$$  \hspace{1cm} (4.70)

$$\Leftrightarrow \left|\left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) P_\delta^{-1/2} \Delta \tilde{x}\right|_2 \leq c_j$$  \hspace{1cm} (4.71)

$$\Leftrightarrow \left|\left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) P_\delta^{-1/2} \Delta \tilde{x}\right|_2 \leq c_j^2$$  \hspace{1cm} (4.72)

$$\Leftrightarrow c_j^2 - \left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) X \left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right)^T \geq 0.$$  \hspace{1cm} (4.73)

**Formulating LMI Conditions**  Analogous to the work in [32] we pick the following choice for the substitution of $P_\delta$ and $K_\delta$:

$$X = P_\delta^{-1}, \quad Y = K_\delta P_\delta^{-1} = K_\delta X.$$  \hspace{1cm} (4.74)

With this choice we can use the LMIs of the following equations written in variables $X$ and $Y$:

1. $$AX + BY + (AX + BY)^T + 2\rho_c X \leq 0, \text{ (from (4.59)).}$$  \hspace{1cm} (4.75)

2. $$\begin{pmatrix} X & d_w^T \\ d_w & w^2 \end{pmatrix} \geq 0, \text{ (from (4.64)),}$$  \hspace{1cm} (4.76)

which must be checked for all vertices $d_w$ in $W$.

3. $$c_j^2 - \left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) X \left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right)^T \geq 0, \text{ (from (4.73)).}$$  \hspace{1cm} (4.77)

This is not an LMI yet, but can be transformed to one:

$$\text{Schur complement } \begin{pmatrix} c_j^2 & \frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right) X \left(\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r \right)^T \geq 0.$$  \hspace{1cm} (4.78)

E.g. for a polytopic constraint on the state we get

$$\frac{\partial g_j}{\partial x}|_r + \frac{\partial g_j}{\partial u}|_r \frac{\partial \kappa}{\partial x}|_r = L_x$$  \hspace{1cm} (4.79)

$$\rightarrow \left(\begin{array}{c} c_j^2 \\ (L_x X)^T \end{array} \right) X \geq 0.$$  \hspace{1cm} (4.80)

**Remark.** Usually it makes sense to describe $d_w$ limited by its $\infty$-norm. This implies that none of the vector entries can exceed a certain boundary. Therefore, the possible set for the disturbance is given by a polyhedron. Note that for a polyhedron it’s only necessary to consider its vertices for the LMI seen before.
4.6.2 Constraint Tightening

As seen in the formulation of our optimization problem in (4.53), we need to find parameters $c_j$, $j = 1, \ldots, p$. These parameters define how much each of the constraints needs to be tightened to still be in a position to guarantee the desired robustness properties.

Polytopic State Constraints

Using polytopic constraints significantly simplifies the calculation of $c_j$ for the $j$-values corresponding to the states. Putting in the polytopic constraints to Equation 3.21 of Assumption 3.5 gives us the following relation:

$$ c_j \geq \frac{L_{j,x}(x - z) + L_{j,u}\kappa(x, z)}{\sqrt{V_{\delta}(x, z)}}, \quad \forall (x, z, v) \text{ with } V_{\delta}(x, z) \leq \delta_{loc}. \quad (4.81) $$

For the practical computation of $c_j$, we therefore get for our setup (see (4.71)):

$$ c_j := |(L_{j,x} + L_{j,u}K)P^{-1/2}|_2. \quad (4.82) $$

Nonlinear Output constraints

As we want to enforce constraints on our output which is usually highly nonlinear, we also need to find a $c_j$ for its tightening. The general condition can be found in the following:

$$ c_j \geq \frac{g_j(x, \kappa(x, z)) - g_j(z, v)}{\sqrt{V_{\delta}(x, z)}}, \quad \forall (x, z, v) \text{ with } V_{\delta}(x, z) \leq \delta_{loc}. \quad (4.83) $$

By looking at (4.71) we can again write this as:

$$ c_j = \max_r \left| (\frac{\partial g_j}{\partial x} |_r + \frac{\partial g_j}{\partial u} \frac{\partial \kappa}{\partial x} |_r)P^{-1/2} \right|_2. \quad (4.84) $$

Since in principle we need to include all $r \in \mathcal{Z}$, we again sample this to get the correct value.

Collision Avoidance

In principle our approach would also be expandable to exact collision avoidance. In contrast to what we do, collision avoidance does not require to find a differentiable function which bounds the constraint but more general and practical relevant formulations are possible, e.g. also box constraints. This approach was introduced in [69] and works by reformulating non-differentiable constraint collision avoidance constraints into smooth nonlinear constraints by using strong duality of convex optimization while not introducing additional approximations. A robust formulation of this approach can be found in [58] by Soloperto et al..

4.6.3 Check Exponential Decay Rate

Before actually applying the controller in practice, it turned out to be very useful to check the calculations done before once again. This reduces the risk of possible mistakes significantly.

Linear systems

For linear systems this is straightforward. We need to check that the following really holds:

$$ (A + BK)^T P + P(A + BK) + 2p_c P \leq 0. \quad (4.85) $$
Since we are considering linear systems only, $A$ and $B$ holds everywhere. We can modify this condition to the following:

\[(A + BK_{\delta})^T P_{\delta} + P_{\delta}(A + BK_{\delta}) + 2\rho_{c} P_{\delta} \leq 0\]  
\[\Leftrightarrow P_{\delta}^{-1/2} (A + BK_{\delta})^T P_{\delta}^{1/2} + P_{\delta}^{1/2} (A + BK_{\delta}) P_{\delta}^{-1/2} + 2\rho_{c} I_{n \times n} \leq 0\]  
\[\Leftrightarrow P_{\delta}^{-1/2} (A + BK_{\delta})^T P_{\delta}^{1/2} + P_{\delta}^{1/2} (A + BK_{\delta}) P_{\delta}^{-1/2} \leq -2\rho_{c} I_{n \times n}\]  
\[\Leftrightarrow \lambda_{\text{max}} \left( P_{\delta}^{-1/2} (A + BK_{\delta})^T P_{\delta}^{1/2} + P_{\delta}^{1/2} (A + BK_{\delta}) P_{\delta}^{-1/2} \right) \leq -2\rho_{c}.\]

**Nonlinear Systems**

For the nonlinear case we can recompute the real value for $\rho_{c}$ to check whether our imposed value during the optimization holds in practice. For this, a surrounding $\Delta z$ of $x$ defined by the incremental Lyapunov function is investigated and then $\rho_{c}$ is calculated according to the following rule:

\[\rho_{c} = \min \left( \rho_{c}, \frac{\Delta z^T P_{\delta}(\dot{z} - \dot{x})}{\Delta z P_{\delta} \Delta z} \right).\]

This needs to be done in a sampling fashion since it only allows to make statements locally.

**4.6.4 Terminal Ingredients**

The terminal ingredients consist of terminal cost and terminal set. The terminal cost can be computed according to [10] as described earlier. The terminal set can then be calculated as described in Section 4.4.2 for dynamic set points or as given in [36] for constant set points.
Chapter 5

AMPC Approach

In this chapter we introduce the idea of an AMPC by combining the contents of the preceding Chapters 3.1 and 3.3. As mentioned earlier, the idea of this thesis to approximate the RMPC controller by ML regression is based on the work in [19]. Our focus is more practical and one of our goals is to demonstrate this new approach in practice on a complex real world system, the Apollo robot. Additionally, we base our RMPC controller on the work in [36] instead of [33] as it was done in [19]. Also our validation procedure for the AMPC was modified compared to the earlier work of Hertneck et al. by not considering the error in the approximated input \( \pi_{\text{approx}} \) but on the next DT state at the next time step \( x_{\text{approx}}(t + 1) \). Therefore we use the true induced error along the trajectory which allows us to be less conservative. Additionally our RMPC problem formulation including the constraint tightening is computed in CT allowing for sampling time independent statements. Furthermore, for the MPC formulation itself we use prestabilized dynamics in (pseudo) CT with a feedback controller \( K \cdot x \). For the Apollo implementation we draw on a special MPC reference tracking formulation [43]. Finally, the whole AMPC implementation is done in an efficient and fast C++ implementation using ACADO and CASADI for solving MPC problems and TensorFlow C++ API for approximating it.

In the following we highlight the major improvements compared to [19].

5.1 Problem Setting

The main idea for the AMPC formulation is borrowed from [19]. For the sake of simplicity and the used validation procedure, we write this chapter in discrete time even though we are deriving the RMPC quantities in continuous time.

We consider the nonlinear discrete-time system

\[
x(t + 1) = f(x(t), u(t)), \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m, \quad t \in \mathbb{N}
\]

and the perturbed discrete-time system

\[
x(t + 1) = f_w(x(t), u(t), d(t)), \quad d \in \mathcal{D}.
\]

In contrast to the work in [19] where compact polytopic input and state constraints

\[
\mathcal{X} = \{ x \in \mathbb{R}^n | Hx \leq 1_p \}, \quad \mathcal{U} = \{ u \in \mathbb{R}^m | Lu \leq 1_q \}
\]

are addressed, we additionally cover nonlinear constraints of the form

\[
g_j(x, u) \leq 0, \quad j = r + 1, \ldots, p.
\]

Our control objective is to ensure stability of \( x_s \), constraint satisfaction and to optimize some cost function \( V(x; u^*) \) for the regular MPC problem and \( V(x, y_t; u^*, y_s) \) for the tracking formulation we use for Apollo. For this problem we design an MPC which is robust to additive noise on \( f(\cdot, \cdot) \)
- this also includes inaccurate inputs - within chosen bounds. This RMPC is then sampled offline over the set of feasible states \( \mathcal{X}_{\text{feas}} \), delivering the optimal solution \( \pi_{\text{MPC}}(x) \) and approximated using supervised learning, techniques, NNs in our case. This yields an AMPC function \( \pi_{\text{approx}} : \mathcal{X}_{\text{feas}} \to \mathcal{U}(u = \pi_{\text{approx}}(x)) \) with a closed loop system given by

\[
x(t + 1) = f(x(t), \pi_{\text{approx}}(x(t))). \quad (5.5)
\]

Thanks to the RMPC design, robust constraint satisfaction and stability are guaranteed if the additive noise has less influence than considered during the design procedure (compare Sec. 3.2.3).

5.2 Procedure

The algorithm for designing the AMPC is the same as Algorithm 1 in [19]. However, for the validation we use the condition in (5.17) instead of (5.9).

For the RMPC design we use the scheme described in the last chapters, based on [36]. We choose this approach to compensate for the influence of appearing disturbances. As a short recap of Assumption 3.3, we account for disturbances of the following form to preserve the guarantees of our controller:

\[
|d_w|^2 \leq \hat{w}(x, u) \quad \forall d_w \in \mathcal{W}(x, u) \subset \mathbb{R}^n. \quad (5.6)
\]

There exists a scalar function \( \hat{w} \) that satisfies the following:

\[
|d_w| \leq \hat{w}(x, u) \quad \forall d_w \in \mathcal{W}(x, u), \quad (5.7)
\]

for a given norm \( \cdot \). For the sake of simplicity we only consider the case of additive disturbances, i.e. the perturbed system from (5.2) becomes the following:

\[
x(t + 1) = f_w(x(t), u(t), d(t)) = f(x(t), u(t)) + d_w, \quad \text{with} \quad d_w \in \mathcal{W}. \quad (5.8)
\]

Remark. The main difference between using additive and general disturbances is denoted by using \( d_w \in \mathcal{W} \) instead of \( d_w \in \mathcal{W}(x(t), u(t)) \) for the disturbed system \( f_w(x(t), u(t), d) \).

5.3 Previous Work [19]

5.3.1 Validation Criterion

In [19] they derived a criterion to compensate for the non-idealities in the form of the inaccurate input approximation \( \pi_{\text{MPC}}(x) \). This robustness criterion is very intuitive for the setting of developing an approximate MPC. The input is inaccurate and therefore the goal is to directly check how large this deviation might be compared to how large it is allowed to be. The criterion is given in the following:

\[
|\pi_{\text{approx}}(x) - \pi_{\text{MPC}}(x)|_\infty \leq \eta, \quad (5.9)
\]

i.e. this implies that

\[
\pi_{\text{approx}}(x) = \pi_{\text{MPC}}(x) + d, \quad \text{with} \quad |d|_\infty \leq \eta. \quad (5.10)
\]

\( \eta \) is a user determined parameter which defines how big the deviation can become without losing the given guarantees for all \( x \in \mathcal{X} \). This is possible due to an additional assumption in [19] which requires that there exists a Lipschitz constant \( \lambda \in \mathbb{R} \), s.t. \( \forall x \in \mathcal{X}, \forall u \in \mathcal{U}, \forall u + d \in \mathcal{U} \) the following holds:

\[
|f(x, u + d) - f(x, u)| \leq \lambda |d|_\infty. \quad (5.11)
\]

Hence, for this setup the influence of the input inaccuracies to the deviation of \( x(k + 1) \) is described by introducing an additional Lipschitz constant. Therefore, the input inaccuracy is transformed to a system disturbance in our RMPC setup. Since this constant needs to satisfy the disturbance bound everywhere, this can introduce additional conservatism compared to the case of just bounding the influence on \( x(k + 1) \) as seen in (5.8).
5.3.2 Statistical Guarantees
For the statistical guarantees, in [19] the statistical verification was done for whole trajectories, i.e. trajectories
\[ X_i := \{ x(t), t \in \{0, \ldots, T_i \} : x(0) = x_i, x(T_i) \in \mathcal{X}_f, x(t+1) = f(x(t), \pi_{\text{approx}}(x(t))) \}. \] (5.12)
As it can be seen, only the nominal model without additional disturbances was covered. To be in a position to observe statistically meaningful information, an indicator function is introduced:
\[ I(X_i) = \begin{cases} 1, & \text{if } |\pi_{\text{MPC}}(x) - \pi_{\text{approx}}(x)|_\infty \leq \eta, \forall x \in X_i \\ 0, & \text{otherwise}. \end{cases} \] (5.13)

Applying Hoeffding’s Inequality
The empirical risk is given as
\[ \bar{\mu} := \sum_{j=1}^{p} I(X_j). \] (5.14)
Together with Hoeffding’s inequality, given as
\[ \mathbb{P} [ |\bar{\mu} - \mu| \geq \epsilon_h ] \leq 2e^{-2p\epsilon_h^2}, \] (5.15)
the following statement can be derived:
\[ \mathbb{P} [ I(X_i) = 1 ] \geq \mu_{\text{crit}}, \quad \text{with } \mu_{\text{crit}} \leq \bar{\mu} - \epsilon_h = \bar{\mu} - \sqrt{\frac{\ln \left( \frac{d_h}{2} \right)}{2p}} \] (5.16)
holds at least with confidence level \( 1 - \delta_h \).
This result was taken from [19]. For more information on Hoeffding’s inequality, see Section 3.3.

5.4 Validation
5.4.1 New Validation Criterion
We now come up with the new validation criterion, which is not as intuitive as the one explained before but more close to the formulation of our RMPC design and therefore less conservative. We can check the overall system disturbance in the space of the incremental Lyapunov function, i.e. directly with respect to the norm we are interested in, which the least conservative measurement for this approach.
This criterion is given in the following:
\[ \sqrt{V_\delta(f_w(x, \pi_{\text{approx}}, d), f(x, \pi_{\text{MPC}}))} = \sqrt{V_\delta(f(x, \pi_{\text{approx}}) + d_{w, \text{model}}, f(x, \pi_{\text{MPC}}))} \] (5.17)
= \left| d_{w, \text{approx}} + d_{w, \text{model}} \right| P_\delta = |d_w| P_\delta \leq \bar{w}. \] (5.18)
Here, \( \bar{w} \) is the maximum admissible disturbance in this norm, which results from our assumption of the overall occuring disturbance \( d_w \). In the ideal case \( d_w \) would be a vector of zeros, whereas in practice it takes values unequal to zeros due to the approximation error \( (d_{w, \text{approx}}) \) and the model mismatch and measurement noise \( (d_{w, \text{model}}) \).

Remark. By comparing our new validation criterion from (5.17) to the old one in (5.9) it is obvious that we look at the full disturbance including the error of the approximation as well as additional disturbances such as model mismatch and measurement noise. In the old validation criterion, solely the approximation error was covered.
5.4.2 Statistical Guarantees

If we would do it the same way as in [19], the indicator function (compare (5.13)) for our trajectory would look as following:

\[
I(X_i) = \begin{cases} 1, & \text{if } |f_w(x, \pi_{approx}, d) - f(x, \pi_{MPC})|_{P_k} = |d_w|_{P_\delta} \leq \bar{w}, \forall x \in X_i \\ 0, & \text{otherwise.} \end{cases}
\] (5.19)

Discussion

As mentioned before, it is important to say that our solution also covers additional disturbances such as the model mismatch. To be in a position to give the statistical guarantees from the last section, we need the following:

1. The system needs to be sampled along closed loop trajectories.
2. The model mismatch needs to be deterministic, i.e. \(d_{w,\text{model}}\) along the trajectory needs to be deterministically determined for a given starting point.

Remark. Due to the fact that we use Hoeffding’s inequality for the validation, the trajectories needs to be i.i.d.. Since we sample the start points of the trajectories randomly, the trajectories are also i.i.d. if the overall mapping to the indicator function is deterministic. Additional uncertainty in the real world system can lead to the case of non deterministic behavior and therefore to trajectories not being i.i.d. anymore.

Due to the need of having a deterministic function, we decided to do our validation on our robot simulator, trying to mimic the real world scenario. It contains effects neglected by our model such as inertia and friction but is still deterministic, why our arguments still hold. To be in a position to do the validation on the real world system, modifications would be needed.

Advantages

Nevertheless, this approach has several advantages over the one in the previous work. It is

1. less conservative since the influence of the input does not need to be described by an upper bounding constant,
2. also accounts for the integration error in the simulation,
3. it covers the model mismatch between our simplified kinematic model and the real identified system given in the robotic simulation (SL, compare Chapter 6)
4. and it could even be used to detect in real-time whether our model assumptions are still satisfied.

5.5 New Ways of Sampling

Later on in this thesis, we will both show, a simulation example (the same as in [19]) used to validate our frameworks and add further extensions as well as a practical example; controlling a robotic arm on Apollo robot. Due to the different nature of the two examples with respect to the state dimensionality we apply different techniques for doing the sampling. E.g. the chemical stirred-tank reactor (CSTR) has only 2 dimensions and its policy is easily visualizable and interpretable while the robot has much higher dimensions.

For both approaches we developed new approaches for doing the sampling. For the continuous stirred-tank reactor (CSTR) we introduced a recursive approach for the sampling enabling vanishingly small approximation errors while saving computational costs. For the robot experiments we first do random sampling (makes it easier to add data afterwards or to use data while the sampling
has not finished) and then randomly sample closed-loop trajectories to get more related data to our real world scenario and our validation procedure.

We will explain each of them in the corresponding chapters (Section 7.2.6 and Section 9.2.3) in more detail and outline the main differences compared to the previously conducted sampling in [19].

5.6 Improved Neural Network Approximation

Also in this case we introduced new techniques (especially for the simulation example) and advantageous practices for the neural network regression, trying to learn the RMPC policy. The structure of the underlying policy as well as the dimensionality of the simulation example and the robot experiment is quite different. Therefore, we describe the improvements and considerations in the corresponding chapters (Section 7.2.5 and Section 9.2.3).
Chapter 6

Frameworks

6.1 Overview

As described earlier in Chapter 1, one of the main tasks of this Master Thesis is to develop and implement an AMPC approach with guarantees on Apollo robot (see Fig. 1.1) and to deploy it in practice. Due to the high time critical demands of the robot - all hardware related command signals needs to be updated at a frequency of $1kHz$ - the whole robotic control framework (Sec. 6.4) is implemented in real-time C++. To be compatible with this framework and also to fulfill the demanding requirement, we decided in the very beginning to also implement all of the code which will actually be needed to run on the robot in C++. Some other offline calculations, e.g. for training the network or validating theoretical ideas, are also implemented in Python and MATLAB.

Detailed instructions on how to use the created code can be found in our corresponding user manual which is given in the Appendix of this thesis.

6.2 Model Predictive Control Framework

We knew from the very beginning of this work that we should be able to solve optimal control problems (OCPs) in C++ to avoid error-proneness due to required interfaces and to not introduce additional delay. Otherwise, effort in handling the communication of different frameworks or potentially even programming languages would have been needed. This would have become especially difficult since we need to be able to deploy our code in a real-time environment (Section 6.4).

6.2.1 ACADO

After doing some research and comparison between CasADi and ACADO [21], we decided to do our programming using ACADO in C++. The important reason for this decision was the expected faster evaluation of ACADO, the fact that it’s tailored for optimal control problems as well as the potential real-time iteration scheme promising evaluations in the magnitude of $1ms$ [23, 62, 14, 63]. However, it still unveiled some difficulties in the practical usage. It seems as if the online support by the developers has stopped several years ago. For ordinary use cases useful information can be found in the official ACADO manual [22]. However, for more specific problems and questions it turned out to require quite some trial and error, especially due to the fact that the user manual is not fully complete towards the end.

When we started implementing code for the robot control, we encountered bigger problems using ACADO. On the one hand, the underlying quadratic program (QP) solver qpOases had some difficulties solving our optimization problem, potentially due to the highly nonlinear and non quadratic output function. On the other hand it was also really difficult to formulate the control problem in a way such that it would satisfy our mathematical formulation. E.g. even the introduction of additional decision variables is not trivial in ACADO.
6.2.2 CasADi

CasADi [3] is a general purpose optimization problem solver using symbolic variables. It is not tailored for MPC problems such as ACADO. However, in the meanwhile it also offers the opti package which simplifies writing code for optimal control problems and makes the syntax more intuitive. It is possible to introduce additional decision variables, define the optimization objective in a desired way and much more. The standard underlying solver is IPOPT. It offers APIs for Python, MATLAB, C++ and octave. We used CasADi in the very beginning for solving optimization problems in MATLAB and planned to use ACADO for the C++ implementation. However, for the more complicated case of our robotic control we had difficulties in writing our control problem due to the previously mentioned challenges (Section 6.2.1). Therefore, we also switched to CasADi for the C++ robot implementation. It is also worth mentioning that the C++ documentation of CasADi is really poor and the authors of the software advice the users against deploying the C++ API. Also the examples given in the source code for the C++ cases are very simple so it took a while to figure out how we could implement our relatively complex optimization problem. Since also the Python and MATLAB versions are using the same C++ backend, it was clear that the C++ API must also offer the same functionality.

6.3 Machine Learning Framework - TensorFlow

Since this project contains a big learning part, a suitable ML framework is needed. Our goal was to develop the whole software framework in C++, hence, desirably also the ML framework should have a C++ API to avoid programming language interfaces. We are using NNs for the approximation needed for our AMPC controller. One of the most powerful and most common frameworks for the usage of NNs is TensorFlow which also provides a C++ API. Therefore, we decided in the very beginning to use Tensorflow. Alternatives would have been PyTorch or Caffe.

In our case we are using TensorFlow 1.13. Multiple APIs are provided by TensorFlow. The recommended and most comprehensive one is the Python API. Especially for training and creating custom models this is the way to go. For the training we mainly use the KERAS NN library which nowadays is part of TensorFlow’s core library and offers a higher-level set of abstractions which makes it easier to rapidly develop NN-ML models. Despite the skeptical opinion in the web regarding the C++ API we decided to deploy it for the inference in our own C++ programming framework.

6.3.1 Training - Python

The most complete API of TensorFlow is certainly the Python API. It offers a wide range of functionalities and - maybe most importantly - can deployed by using the KERAS front end. Therefore, we used TensorFlow Python for doing the training.

6.3.2 Inference - C++

Again, similar argumentation as in Section 6.2 we aimed for having a full C++ implementation in the end to achieve the desired performance and avoid problems with combining separate programming languages. Luckily, one of the most powerful learning frameworks to date offers a C++ API; the TensorFlow C++ API. To be in a position to use this API the graph trained with KERAS front-end needs to be frozen. This has the additional advantage that the evaluation becomes incredibly fast. An evaluation step of big NNs with millions of decision variables on a CPU only takes less than 1ms. It was also possible to successfully integrate this in our software framework. However, for using the C++ API, TensorFlow must be built from source. This turned out to be challenging. The whole building process is done by the open-source release of the Google internal building system which is called Bazel. More information on how to build TensorFlow from source and how to use it in your C++ framework can be found in the Appendix; Section A.3.4.
6.4 Real Time Robot Control Framework - SL

For the control of our robot we are partly dependent on the framework which is currently used to control the robot. In our case, this is the Simulation Laboratory (SL) [54]. In this section we only introduce aspects that are important for the choice of the system, the influence of the control signal and the dynamic behavior. In [54], many different aspects of the control framework are explained. However, in our opinion most important from an engineering perspective is the block diagram illustration of the different software modules, depicted in Figure 6.1. The motor servo takes on the task of calculating the low level motor commands from the desired quantities $\Theta^d, \dot{\Theta}^d, u_{ff}$ and the current state $\Theta, \dot{\Theta}$ of the robot [54]. The task servo is the environment where the user is intended to influence the system. This is where the control law is implemented, even though very often the low level (e.g. PD) part is done by the motor servo. In here it’s also possible to cancel out the low level control commands and control the whole system by applying the feed-forward command $u_{ff}$. The inverse dynamics servo can be used to create feed-forward commands according to the desired $\Theta^d, \dot{\Theta}^d, \ddot{\Theta}$. Note that in this formulation also the desired acceleration is used. Furthermore, we are not using the inverse kinematics servo and the vision servo of SL. The perception we need is also done in the task servo by spawning a separate thread to not threaten the real time demands.

In the beginning we were considering to only command accelerations and to use the inverse dynamics servo only. However, due to bad feed-forward estimates on the real world system - arising from non ideal system identification - we decided to also calculate reference angles and angular velocities, commanding them to the low level controllers in the motor servo.
6.4.1 Real Time
Since this framework is real-time critical - the robot stops operation when it does not get any signal for more than one interval - we needed to take this into account for the design of our software. We decided not to use a potential real-time ROS implementation but instead spawn real-time threads updating the values of the needed variables. The whole framework functionalities are packaged within a class. An instance of this class is then created in our SL task and its member variables are read out containing the important information.

6.4.2 Modularity
For us it was really important to write a code which is as modular as possible. It is possible to run different configurations, e.g. with or without the NN control, without rebuilding the project. For this purpose we decided to store the configurations in a YAML file which is read into the memory at the beginning of the program execution.

6.4.3 Simulation
In the very beginning we also investigated the use of the control toolbox [16]. It is a toolbox provided by the IDRL lab at ETH Zurich and provides efficient and useful numerical methods and control design functionalities. Especially in terms of simulating our system we considered the usage of this toolbox. However, in the end we decided to write own simulations using integrators in the orders of up to 8, and the simulation environment of SL which also considers inertia of the robotic system as well has the ability to directly test the functioning of our framework.

6.5 Motion Tracking - Vicon
Since we also wanted to be able to perform an interactive demonstration we needed to address the perception part. We considered using the camera or kinect system built in to Apollo. Since this should not be the core of this thesis we decided to first use a more straightforward approach and leave potential extensions for the end, when we would still have time. We decided to use the Vicon system for tracking the obstacles and reference with respect to the robot frame.

6.5.1 Vicon Bridge
The Vicon system uniformly streams its information into the network. Then, on any computer in the same network it is possible to access this provided data. For the purpose of decoding these network signals a really useful package is given; the Vicon bridge. It takes the network signals as inputs, decodes them and then publishes them locally in ROS. In our perception thread, we are then able to listen to this information.

6.6 Developed Frameworks
We developed several different code sources reaching from C++ code for the CSTR simulation, the robot experiments and simulation over MATLAB code used for doing the offline calculations for the RMPC design to Python code for plotting the logged data and training the NNs. More information about the different parts of our code can be found in the Appendix. All code is available in the AMD gitlab server.

6.6.1 Conclusion
From an implementation perspective it was more difficult than expected to use the C++ API of TensorFlow. First of all the API itself is quite poorly documented. Furthermore at the time of this Master Thesis TensorFlow needed to be built from source by using Bazel to get the needed
libraries for the C++ API. Additionally also CasADi has a poor documentation for its C++ usage. Therefore we needed to spent some days on finding correct and efficient ways of implementing our optimization problem.
Chapter 7

Simulation Example - CSTR

To be in a position to reach the goals of this thesis, i.e. to demonstrate the RMPC as well as the AMPC approach on a real world robotic system, we decided to first implement the whole pipeline in C++ for a smaller and less complex simulation example. We were aiming for a fast acting pipeline from the offline to the online calculations. For the sake of comparability we decided to investigate the same system as already done in [19], however with some minor implementation modifications.

The efficiency and speed of the approach in [19] was limited and it was bound to MATLAB which is unsuitable if fast behavior is needed or the pipeline is meant to be applied on a real world system in a C++ environment. Additionally the main contributions in [19] lay more in theory such as successfully proposing statistical guarantees for the AMPC controller than in optimizing the algorithms and frameworks, e.g. parallelization was not really considered. As we are interested in the functioning of the pipeline we will not show too many intermediate results but focus on the resulting learned AMPC controller and its validation.

In contrast to the system formulation for the real world robotic example in Section 9.1 we also need to handle nonlinear system dynamics in this case. Once again, this shows the variety of the underlying RMPC scheme in [36].

7.1 System Formulation

We use the same example as in [19] which is originally taken from [46]. It is a continuous stirred-tank reactor (CSTR) which is a common benchmark for MPC control design. The system is described by:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
1 - x_1 \\
M x_2 - u
\end{bmatrix}
\begin{bmatrix}
x_1 e^{-\frac{x_2}{\Theta}} - k x_1 e^{-\frac{M}{\Theta}} - \alpha u (x_2 - x_c)
\end{bmatrix},
\]

(7.1)

with given parameters\(^1\). Our goal is the same as in [19]; to stabilize the unstable steady state \(x_c = (0.2632, 0.6519)^T\) with steady state input \(u_c = 0.7853\). Constraints are given on the input \(u\) and on the transformed system state \(x_t = x - x_c\):

\[X_t = [-0.2, 0.2] \times [-0.2, 0.2], \quad U = [0, 2].\]

(7.2)

7.1.1 Continuous Time Formulation

As it was already observable in (7.1), we decided to use the CT formulation of the system. We made this choice with the real world robotic example in mind, since there we will have a fast sampling rate of the software framework (i.e. 1kHz) with a significantly smaller sampling rate of the RMPC controller. Therefore we are able to model the prestabilizing feedback as continuous and do the RMPC offline calculations in CT.

\(^1\Theta = 20, k = 300, M = 5, x_f = 0.3947, x_c = 0.3816, \alpha = 0.117\)
Chapter 7. Simulation Example - CSTR

Figure 7.1: Groundtruth taken from [19]. This Fig. depicts the continuous function learned by the neural network. The approximation has no problems to properly mimic the function due to the high number of sampling points.

7.2 Setup

For this example, our main goal was to mimic the result of the simulation example in [19] to reveal bugs in the implementation. Therefore, we did not include dynamic set point tracking in the design.

7.2.1 Groundtruth

Due to the low dimensionality - the state and input dimensions are 2 and 1 respectively - in [19] the set was simply sampled uniformly with a grid size of $2.5 \cdot 10^{-4}$ obtaining $1.6 \cdot 10^6$ feasible out of $2.56 \cdot 10^6$ possible data points $(x, \pi_{MPC}(x))$. On average one evaluation for them took 0.71s which solely leads to a runtime of more than 500 hours. The results of the sampling is depicted in Fig. 7.1.

7.2.2 Assumed Disturbance

In [19] the used RMPC controller was an input robust MPC design. As a result, a Lipschitz constant is needed to bound its influence on the resulting deviation of the next state (compare Section 5.3.1). In the given work, Hertneck et al. determined the Lipschitz constant as $\lambda = 5.5 \cdot 10^{-3}$. With the underlying RMPC design this ends up in an admissible bound on the input disturbance of $\eta = 5.1 \cdot 10^{-3}$ to still guarantee stability, recursive feasibility and constraint satisfaction.

Since we are using the scheme in [36] for the RMPC design instead of [33], we are able to tolerate an $\eta = 6.4 \cdot 10^{-3}$, i.e., 25% more compared to the previous work when bounding the induced disturbance (of the input approximation) by the same parameter $\lambda$. Determining $\bar{w}_d$ from this according to $\bar{w}_d = \lambda \eta$ returns $\bar{w}_d = 0.0226$ for our calculations. Directly checking whether the input disturbance introduced by the neural network is smaller than $\eta$ is more conservative than checking the deviation of the next step with respect to $\bar{w}_d$. Later in this Chapter we will also investigate whether there exists a measurable difference during the validation procedure.

7.2.3 Incremental Stabilizability

The assumption of the incremental stabilizability can be verified as explained in [34] and was already done in [19].
7.2.4 RMPC Design
For the RMPC design we need to handle nonlinear system dynamics. Contrary to [19], we base our design on the work in [36].

Formulation
The resulting RMPC optimization problem can be seen in (7.3). Principally it is equivalent to the general formulation in (3.34) as introduced in [36] for additive disturbances.

\[
V_N(x(t)) = \min_{u(\cdot|t)} J_N(x(\cdot|t), u(\cdot|t)) \tag{7.3a}
\]
subject to
\[
x(0|t) = x(t), \tag{7.3b}
\]
\[
x(k+1|t) = f(x(k|t), u(k|t)), \tag{7.3c}
\]
\[
g_j(x(k|t), u(k|t)) + c_j \frac{1 - \rho^k}{1 - \rho} \bar{w} \leq 0, \tag{7.3d}
\]
\[
x(N|t) \in \mathcal{X}_f, \tag{7.3e}
\]
\[
k = 0, \ldots, N-1, \quad j = 1, \ldots, p,
\]
with \( J_N \) as introduced in Section 3.2.3 and repeated here:

\[
J_N(x(\cdot|t), u(\cdot|t)) = \sum_{k=0}^{N-1} l(x(k|t) - x_s, u(k|t) - u_s) + V_f(x(N|t) - x_s). \tag{7.4}
\]

Constraints
In this example we only need to consider polytopic state and input constraints that are given in (7.2). We can write the constraints as \( g_j(x,u) \leq 0 \) for \( j \in \{1, \ldots, p\} \). We need the Jacobians for the required LMIs needed to compute the incremental Lyapunov function (see (4.77)) and again to calculate \( c_j \) for \( j = 1, \ldots, p \) using (4.84). However, since the Jacobian of polytopic constraints is constant, this simplifies to the form in (4.82).

7.2.5 Learning - Combining Regression and Classification
In the beginning we were thinking about how to potentially improve the learning of this approach. By looking at Figure 7.1 we directly thought about ways of potentially not approximating the whole area using regression. For the naive learning approach of approximating the whole state space with regression, dense sampling needs to be given in all areas. Therefore we were thinking about also introducing a supervised classification method, in our case also given by a NN.

The potential improvement would look as follows: In regions of changing policies we would need to have a properly working regression network, in flat regions we would assign the corresponding value of the detected class, according to the classifying network. This would also induce the necessity of only sampling very finely in these regions, where we expect a fast changing policy. The advantage is that a classification approach needs way less examples to produce suitable results compared to the regression. As a motivation we plotted an exemplary classification in Figure 7.3. To make sure that we always apply the regression input when needed, it is also possible to increase the width of the middle region (see Figure 7.4b).

7.2.6 Alternative Sampling
In the last section we introduced the idea of a combination of classifying the regions and using the regression in the interesting regions where the policy is changing. For this idea to be useful, we also need to be able to profit from it, i.e. we need to be able to only sample the interesting regions. For this we propose a sampling consisting of two steps.
Chapter 7. Simulation Example - CSTR

(a) Uniformly sampled points with grid resolution \(8 \cdot 10^{-4}\).
(b) Recursively sampled points up to grid resolution \(2.5 \cdot 10^{-4}\).

Figure 7.2

Simple Uniform Sampling

The simplest approach and to generally reproduce the results is to uniformly sample the MPC also in our case. In our specific case, the finest resolution we sampled uniformly is \(8 \cdot 10^{-4}\). A plot of all the samples can be seen in Fig. 7.2a.

New Recursive Approach

Even though we might have a good classification for the different regions, we still need an approximation for the remaining fast changing area. Therefore we introduce a recursive approach which locally increases the resolution as long as neighboring values are still differing. For this we start with a square, i.e. 4 points and its corresponding computed MPC values. Then if the increase resolution criteria is fulfilled, 5 more points are introduced and its MPC values are computed. The used criteria is given in Algorithm 3. The difficult part about this implementation was to make full use of parallelism and to not recompute things multiple times. Additionally a finest grid resolution is defined in advance. In our case \(2.5 \cdot 10^{-4}\) as in [19] for the whole grid. This means in the interesting regions we will obtain the same grid resolution as in their approach. The resulting grid can be seen in Fig. 7.2b. Overall only 111459 points need to be sampled for the recursive
7.3 Results

In this section we will shortly show the results we could obtain for the simulation example.

7.3.1 New Learning Approach

It can be clearly seen in Fig. 7.1 that most of the regions of this controller lie in saturation. Actually this is a typical appearance for controllers aiming for optimal behavior. Learning all flat regions for good response in regression requires a lot of points even though the function is trivial in these regions. Therefore we came up with the idea of determining these areas in terms of classification. If the points in the set are not classified wrongly, i.e. free of noise, the regions are geometrically clearly defined. Therefore only few points are needed to achieve good classification results for the flat regions. At the boundaries, more points lead to sharper and more precise edges. This is demonstrated in Fig. 7.3 where only 2500 points are used for training and the regions, especially the saturated ones, are still nicely represented. Looking at Fig. 7.4a shows an example which is trained on the uniformly sampled set from Fig. 7.2a. It is visible that the edges are more clearly evaluated on random points compared to Fig. 7.3. With this approach we can already cover most of the areas, since they are lying in saturation, with way less samples than before. To make classification robust, we can increase the width of the continuous region afterwards by adding an additional percentage to the corresponding output label of the middle region for each point. This can be seen in Fig. 7.4b.
Chapter 7. Simulation Example - CSTR

(7.2a) Classification trained on uniformly sampled points with grid resolution $8 \cdot 10^{-4}$ (compare Fig. 7.2a), evaluated on 200,000 random points. As it can be seen, the middle region is less wide than in Figure 7.4b.

(7.2b) Classification trained on the same amount of uniformly sampled points. In this case the middle region is widened afterwards to make sure that we do not erroneously classify an interesting point as saturated.

Figure 7.4

7.3.2 Pure Regression

For the training of the regression network we used the combined data from Fig. 7.2a and Fig. 7.2b. The resulting approximation can be found in Fig. 7.5a. It is clear that the result in the flat regions is not as good as for the uniform sampling with high grid resolution. However, this is not a problem since we can make use of the classification in these regions if we want to.

For the validation we use 70,000 randomly sampled points visualized in Fig. 7.5b.

7.3.3 Simulation

A simulation of the trajectory for the MPC and the AMPC controllers can be seen in Fig. 7.6. It is visible that even though the control input deviates, the trajectory is almost indistinguishable. This can be seen as an interpretable result of why the new validation criterion based on the state evolving is less conservative than the previous one based on the input deviation.

7.3.4 Validation

For this example we sampled 42518 feasible trajectories. In this case, we sampled them really to full convergence, i.e. not only until the state reached the terminal set. The old validation criterion from [19] satisfied the conditions for our learned AMPC along 24785 trajectories, i.e.

$$\hat{\mu}_{\text{old-val-criterion}} = 58.293\%, \text{ for } p = 42518.$$  \hfill (7.5)

Our new validation criterion is significantly less conservative and violated the conditions not along a single trajectory, i.e.

$$\hat{\mu}_{\text{new-val-criterion}} = 100.0\%, \text{ for } p = 42518.$$  \hfill (7.6)

Hence, our new proposed validation criterion outperforms the previous one significantly with respect to conservatism for practical use cases.
7.3. Results

(a) Surface Plot of the Regression over the Set.

(b) Recursively sampled points.

Figure 7.5

(a) Inputs of MPC and AMPC controllers along the trajectory.

(b) State Trajectory of MPC and AMPC controllers along the horizon for $x_{t,0} = [0.1, 0.06]^T$.

Figure 7.6
7.3.5 Overall Comparison

**Other RMPC approach** Basing the work on [36] instead of [33] allows us to give a less conservative bound $\eta$ on the input disturbance, i.e. without considering any other factors. The values for $\eta$ now become:

$$\eta = 1.4 \cdot 10^{-2}$$  \hspace{1cm} (7.7)

instead of

$$\eta = 5.1 \cdot 10^{-3}$$  \hspace{1cm} (7.8)

in [19].

**Other robustness criterion** This new criterion is less conservative than the old one. In our validation we achieved to...

**Overall Speed-Up** Due to parallelization and different sampling we were able to push down the 500 hours for the sampling in [19] to less than 9 hours with our approach. This is mainly due to the recursive sampling, more efficient code and parallelization.

The fact, that we reached a higher admissible error also increases the speed of the overall procedure. However, we didn’t evaluate enough networks with training data originating from lower resolution - and therefore less points which need to be sampled - to give numerical results on this speed up. But clearly, since the network needs to achieve lower precision, less points need to be sampled in the critical regions and therefore additional speed-up can be achieved.
Chapter 8

Introducing Apollo Robot

The robot we are using for showing the abilities of our RMPC and AMPC approach is the Apollo robot which is located at the MPI in Tübingen. An image of the robot can be seen in Figure 8.1. The robot is equipped with two KUKA LBR4+ robotic arms. However, for our experiments we will only address one of them.

8.1 Geometry

In this section we will introduce the geometry of the robotic arm to give an idea about what movements the robot is able to perform. In Figure 8.2 an illustration of the arm is found. For introducing the coordinate systems it is common to use the Denavit-Hartenberg (DH) convention. By doing so, the derivation of the resulting kinematics is less tedious than by arbitrarily introducing the coordinate systems. The coordinate frames seen in Figure 8.2 were chosen according to this convention.

8.2 Robot Description

As seen in Section 6.4, using the robot control framework is not trivial and we need to adhere to it when deciding for our robot description. It is possible to identify a model of the robot. In literature, there are several works about System Identification of the KUKA LBR4+ robot. However, since the integration would have been non trivial with no guarantees of the result satisfying our requirements, we decided to choose a kinematic approach, to do the control with desired accelerations and leave the task of the feedback linearization to the inner loop low-level controllers. For the sake of potential future work, yet we will shortly describe our findings and works regarding system identification of the KUKA LBR4+ robotic arm.

Remark. Even when considering the dynamic model of the robot, the CT prestabilizing feedback \( \kappa \) can always be chosen in a way such that the resulting system dynamics \( f_\kappa \) are also linear.

8.2.1 Identification of the Dynamics Equations

A comprehensive overview of nonlinear system identification can be found in [55]. It is written as a survey where the authors give guidelines in nonlinear system identification and also a lot of background information to better understand the challenges as well as their origin. The work also covers additional topics such as the impact of structural model errors and process noise on the system identification. The article also covers an overview of possible models, such as among others deterministic differential algebraic equation (DAE) models, stochastic state space models, semi-physical modeling, linearization based models, linear parameters varying (LPV) models, black box models and many more. It also shows the design excitation signals validation of the model and collection of examples.
Getting more to the actual topic of interest, the system identification of the KUKA LBR4+ robotic arm, brings us first to the work of Gaz et al. [15]. In this work, the authors present an approach for the model identification of this robotic arm (denoted as KUKA LWR IV in the paper). On that account they present a general procedure for determining the structure as well as a identification of the values of the dynamic coefficients used by the manufacturer (i.e. KUKA). The authors refer to this approach by calling it reverse engineering since they try to match the numerical data provided by the software interface to a suitable symbolic model. The identification results as well as the training and validation results are publicly available.

Another relevant work was done by Bargsten et al. [5], in which the authors describe a full structured approach for model-based controller design of the KUKA LBR4 robotic arm, including the system identification. Also a control application is presented to show validity of the proposed system identification. Along with this paper, also a freely available model toolbox for this robotic arm is published which might be interesting for future work.

Another paper trying to estimate the system dynamics of the KUKA LWR robot can be found in [25]. In this work the authors try to utilize the presence of joint torque sensors data offered by the KUKA robotic arm. Traditional identification methods only take into account the motor position and the motor torques computed as a result of the current reference signal multiplied by the joint drive gains. In this paper by contrast they are directly using the joint torque data measured by sensors and compare the results to the traditional approach as well to using both sources of information. Surprisingly they can show that using motor torques calculated from electric currents lead to the same accuracy as using usually expensive torque sensors.

The last work we want to cite in this chapter is done by Stürz et al. [59]. They also perform system identification, however not on the KUKA LBR4 but on its successor LBR iiwa. Similarly to the robot we are using, it is also equipped with torque sensors in each joint. This paper presents an identification of the minimal set of needed parameters as well as a consistent set of parameters for the physics robot, not neglecting friction. In this case the parameters are obtained by solving a nonlinear optimization problem with additional constraints to ensure physical feasibility.

8.3 General Kinematics of KUKA LBR4+

As shortly mentioned before, we decided to do the control task on the kinematic model and therefore leave some of the tasks to the inner loop consisting of the low level controllers. The exact model
8.3. General Kinematics of KUKA LBR4+

Figure 8.2: Illustration of the geometry and local coordinates of the KUKA LBR4 robotic arm. Image taken from the Notes on the KUKA LWR4 dynamic model document from Coppelia Robotics GmbH and Massimo Cefalo (http://www.coppeliarobotics.com/contributions/LBR4p_dynamic_model.pdf)

Denavit-Hartenberg reference frames

V-rep default reference frames
(for each link the reference frame is centered in its bounding box)

Kinematic parameters:

\[ l_0 = 11 \text{ cm} \]
\[ l_1 = l_2 = l_3 = l_4 = 20 \text{ cm} \]
\[ l_5 = 19 \text{ cm} \]
\[ \text{offset} = 7.8 \text{ cm} \]
Chapter 8. Introducing Apollo Robot

will be introduced in the next chapter. Nevertheless we will present the position of the end-effector already in this section. More information about the derivation of kinematics can be found in these two standard references [56, 57].

We aim for the position of the end effector. This highly nonlinear function can be derived using the DH convention as introduced in Section 8.1 and is given in (8.1).

\[
o_{\text{end-eff}}(\Theta) = \begin{bmatrix}
-0.4 \cos(\Theta_1) \sin(\Theta_2) - 0.39 \sin(\Theta_4) \sin(\Theta_1) \sin(\Theta_3) \\
0.39 \sin(\Theta_4) (\cos(\Theta_1) \sin(\Theta_3) + \cos(\Theta_2) \cos(\Theta_3) \sin(\Theta_1)) \\
0.4 \cos(\Theta_2) + 0.39 \cos(\Theta_2) \cos(\Theta_4) + 0.31 \\
- \cos(\Theta_1) \cos(\Theta_2) \cos(\Theta_3) - 0.39 \cos(\Theta_1) \cos(\Theta_4) \sin(\Theta_2) \\
- 0.4 \sin(\Theta_1) \sin(\Theta_2) - 0.39 \cos(\Theta_4) \sin(\Theta_1) \sin(\Theta_2) \\
+ 0.39 \cos(\Theta_3) \sin(\Theta_2) \sin(\Theta_4)
\end{bmatrix} (8.1)
\]

Remark. Also the orientation of the end effector could be included in the output function. Its dimensionality would then expand to 6 (instead of 3). However, since we are not using the orientation for our control task we decided not to show the derivation here.

The derived position of the end-effector given in (8.1) denotes the position of the origin of coordinate systems 5, 6 and 7 in Figure 8.2, i.e. the middle of the last spherical part of the robot. However, in our use case we are interested in the position of the middle of the palm of the barrett hand. If the last three joints are fixed at \(\Theta_{5,6,7} = 0\), we can write its position only dependent on \(\Theta_i, i = 1, \ldots, 4\) as in the following:

\[
o_{\text{hand-palm}}(\Theta) = \begin{bmatrix}
-0.4 \cos(\Theta_1) \sin(\Theta_2) - 0.578 \sin(\Theta_4) \sin(\Theta_1) \sin(\Theta_3) \\
0.578 \sin(\Theta_4) (\cos(\Theta_1) \sin(\Theta_3) + \cos(\Theta_2) \cos(\Theta_3) \sin(\Theta_1)) \\
0.4 \cos(\Theta_2) + 0.578 \cos(\Theta_2) \cos(\Theta_4) + 0.31 \\
- \cos(\Theta_1) \cos(\Theta_2) \cos(\Theta_3) - 0.578 \cos(\Theta_1) \cos(\Theta_4) \sin(\Theta_2) \\
- 0.4 \sin(\Theta_1) \sin(\Theta_2) - 0.578 \cos(\Theta_4) \sin(\Theta_1) \sin(\Theta_2) \\
+ 0.578 \cos(\Theta_3) \sin(\Theta_2) \sin(\Theta_4)
\end{bmatrix} (8.2)
\]

This is also the position we are using for controlling the robot later on.

8.4 SL Specifics and Constraints

8.4.1 Joint Offset DH convention to SL

We do all of our control actions on the DH model derived in Section 8.3. Though, the angles are defined differently in SL and therefore need conversion. The shift between the two angle definitions can be seen in Table 8.1. With these shifted values we can obtain the angles for SL using the transformation in (8.3).

\[
\Theta_{\text{joint,SL}} = \Theta_{\text{joint,DH}} + \Delta \Theta_{\text{joint}}
\]

Table 8.1: Shift between angles in SL and our DH convention.

<table>
<thead>
<tr>
<th>Joint</th>
<th>lbr-0</th>
<th>lbr-1</th>
<th>lbr-2</th>
<th>lbr-3</th>
<th>lbr-4</th>
<th>lbr-5</th>
<th>lbr-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Delta \Theta_{\text{joint}}) [rad]</td>
<td>0</td>
<td>-1.57</td>
<td>1.05</td>
<td>0</td>
<td>-1.57</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

8.4.2 Joint Limits

As a matter of principle the angular joint limits are given by the constraints of the hardware, since the joints are not able to do more than one full rotation. Nevertheless, for our angular joint limits
Table 8.2: Angular joint limits of the KUKA LBR4 defined in SL’s Joint Space.

<table>
<thead>
<tr>
<th>Joint</th>
<th>lbr-0</th>
<th>lbr-1</th>
<th>lbr-2</th>
<th>lbr-3</th>
<th>lbr-4</th>
<th>lbr-5</th>
<th>lbr-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min [rad]</td>
<td>-2.96</td>
<td>-3.1</td>
<td>-1.9</td>
<td>-2.09</td>
<td>-3.1</td>
<td>-2.09</td>
<td>-2.96</td>
</tr>
<tr>
<td>Max [rad]</td>
<td>2.96</td>
<td>-0.1</td>
<td>4.0</td>
<td>2.09</td>
<td>1.35</td>
<td>2.09</td>
<td>2.96</td>
</tr>
</tbody>
</table>

we use the joint constraints as given in SL, to not violate any internal constraints. Those angular joint limits observed from SL are given in Table 8.2. The angular velocity is not really limited in the same way and is not clearly restricted in SL. Nevertheless, after reaching a certain velocity the KUKA controller kicks in and stops the robot. Based on experience the robot is able to move at high speeds before reaching that point. Therefore, we arbitrarily define these values as design parameters, also dependent on the speed of our controller. We define the values as it can be seen in Table 8.3. If we use a more comprehensive system state dynamics as seen in the next chapter.

Table 8.3: Angular velocity joint limits, used as design parameters.

<table>
<thead>
<tr>
<th>Joint</th>
<th>lbr-0</th>
<th>lbr-1</th>
<th>lbr-2</th>
<th>lbr-3</th>
<th>lbr-4</th>
<th>lbr-5</th>
<th>lbr-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max [rad/s]</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
</tr>
</tbody>
</table>

in (9.2) we can also enforce bounds on the acceleration. In this case we choose the ones seen in Table 8.4.

Table 8.4: Angular acceleration joint limits, used as design parameters.

<table>
<thead>
<tr>
<th>Joint</th>
<th>lbr-0</th>
<th>lbr-1</th>
<th>lbr-2</th>
<th>lbr-3</th>
<th>lbr-4</th>
<th>lbr-5</th>
<th>lbr-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max [rad/s²]</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

**Remark.** As it can be seen in the next chapter, we assume to be in a position to control the angular velocity or the angular acceleration (depending on the model) of the joints directly. Therefore, the limitations in angular velocity and angular acceleration are a powerful and important way to limit the model uncertainty.

### 8.4.3 Robot frame to DH convention

Additionally, since reference and obstacles are given in the robot frame we need to transform them to our DH robot arm coordinate system. The robot frame coordinate system is located between the shoulders of the robot, with z axis pointing upwards and x-axis pointing horizontally.

Let us denote the transformation subscripts as following: $\mathbf{x}_{\text{arm}} = T_{\text{arm-apollo}} \cdot \mathbf{x}_{\text{apollo}}$. In our case, the transformation between the robot frame and the DH robot arm is static. This static transformation for points in the Apollo frame to points in the DH frame is derived in the following by inspection:

\[
R_{\text{arm-apollo}} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1/2 \\ \sqrt{3}/2 \end{pmatrix}, \quad R_{\text{arm-apollo}} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -\sqrt{3}/2 \\ 1/2 \end{pmatrix}, \quad (8.4)
\]

\[
R_{\text{arm-apollo}} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \quad (8.5)
\]

This results in the following rotation matrix

\[
\Leftrightarrow R_{\text{arm-apollo}} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = R_{\text{arm-apollo}} = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & -\sqrt{3}/2 & 0 \\ \sqrt{3}/2 & 1/2 & 0 \end{pmatrix}. \quad (8.6)
\]
Additionally we can observe the following translation vector by inspection:

\[
t_{\text{arm-apollo}} = 0.05813 \begin{pmatrix} 0 \\ 1/2 \\ -\sqrt{3}/2 \end{pmatrix}.
\]  
(8.7)

### 8.5 Controlling the Robot

As seen in Section 6.4 we use SL for the robot control. SL integrates low-level PID controllers for controlling the state and acceleration by taking a reference and transforming it to voltage commands for the motors of the KUKA LBR4+ robotic arms. In our experience they work reliably if a smooth and continuous state trajectory is set as reference. We are not using the underlying dynamic model of the robotic arms but leave that to the low level controllers. The solution of the MPC controller grants access to the optimal input \(u^*(0|t)\) in the form of a reference velocity \(\dot{x}_r\) (compare (9.1)) or acceleration \(\ddot{x}_r\) (compare (9.2)). Since the low-level PID controllers takes a reference state \(x_r\) and velocity \(\dot{x}_r\) as input, we need to infer the state (compare (9.1)) or the state and the velocity (compare (9.2)) from the obtained input \(u^*(0|t)\). We can take the desired acceleration into account by using the inverse dynamics function of SL to obtain a feedforward command (compare Figure 6.1) which is directly applied to the KUKA robotic arm.

**Simplified System (9.1)** For this formulation introduced in (9.1) the input \(u^*(0|t)\) describes the desired velocity. According to the system definition we directly get as output of the optimization problem the following:

\[
\dot{x}_r = u^*(0|t).
\]  
(8.8)

As the velocity reference might be discontinuous in this simple setting it is not easy to get a reasonable estimate for the acceleration. We decided to set it to zero:

\[
\ddot{x}_r = 0.
\]  
(8.9)

Using the inverse dynamics function still offers some gain, e.g. due to gravity compensation. E.g. this helps the low level controllers in positions where the arm is close to horizontal. The desired speed can easily be targeted in real time (i.e. 1ms) in the MPC sampling time interval with the following policy:

\[
x_r^+ = A_{d,1ms} x_r + B_{d,1ms} (u^*(0|t) + K_\delta x_{\text{real}}).
\]  
(8.10)

\(A_{d,1ms}\) and \(B_{d,1ms}\) denote the matrices of the discretization corresponding to a sampling time of 1ms. At the beginning of an interval, of course we then set \(x_r = x_{\text{real}}\).

**Full System (9.2)** In this setting the output of the system directly describes the acceleration

\[
\ddot{x}_r = u^*(0|t).
\]  
(8.11)

The desired velocity and state can be calculated accordingly:

\[
x_r^+ = A_{d,1ms} x_r + B_{d,1ms} (u^*(0|t) + K_\delta x_{\text{real}}),
\]  
(8.12)

where \(x_r\) contains both the angle and the angular velocity and \(A_{d,1ms}, D_{d,1ms}\) denote the discretization for the system, given a sampling time of 1ms.

**Remark.** The exact discretization for a piecewise constant input signal \(u^*(0|t)\) over an arbitrary long time horizon would be

\[
x^+ = A_{d,d} x + B_{d,d} u^*(0|t),
\]  
(8.13)
where $A_{cl,d}$ and $B_{cl,d}$ denote the matrices for the exact discretization of the continuous time system
\[ \dot{x} = (A + BK_δ)x + Bu^*(0|t). \]
Therefore normally the expression in (8.10) and in (8.12) would only be an approximation due to a non constant input term $K_δx_{real}$. However, in this specific case we are calculating the next state after 1ms. And since the internal rate of the framework is also given as 1ms, $x_{real}$ is constant within this interval.
Chapter 9

Experimental Results on Apollo Robot

In this chapter we present our results on the robotic system we introduced in the last chapter. Our goal is to control the position of the arm under constraints in the state and input space as well as constraints regarding the position of the end-effector. In Section 9.1 we will frame this task as a control problem. In Section 9.2 we will introduce the RMPC design and the learning architectures before we will present the results in Section 9.3.

9.1 System Formulation

9.1.1 System State Dynamics

In the following, we will present two ways of formulating the problem as a control task. One simplified one where we assume to be able to directly control the velocity of the joints, and another more realistic one where we use the joint accelerations as control signal. The motivation for the first one is mainly to reduce dimensions to be in a more comfortable position to sample the state space for the learning approach. The robot has 7 degrees of freedom, nevertheless, we only control the first 4 joints and keep the joints 5-7 constant. This is sufficient for controlling the position but becomes a restriction when control of the end-effector orientation would be desired.

Simplified Formulation

For the kinematic control for Apollo we define the system state vector and its derivative as in (9.1), since we assume being able to control the velocity of the joints directly relying on the low level joint controllers.

\[
\begin{align*}
  x &= \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \Theta_1 \\ \Theta_2 \\ \Theta_3 \\ \Theta_4 \end{pmatrix}, \\
  \dot{x} &= f(x, u) = \begin{pmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \\ \dot{\Theta}_3 \\ \dot{\Theta}_4 \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix}.
\end{align*}
\]  

(9.1)

We define the corresponding output function as the end-effector position of the robotic arm as this is the item of interest. Its arithmetic expression was derived in Section 8.3 in (8.2).

In practice this formulation of the state dynamics only makes sense under the following assumption.

Assumption 9.1. The underlying controller which controls the speed of the robot joints is fast enough, i.e. also possibly discontinuous or fast changing velocity commands are reached in the system by applying the underlying low level controllers.
Remark. As we will see in Section 9.2.1 this model is realistic for low absolute velocities, leading to smaller steps in the velocity reference. Additionally it becomes more and more unrealistic the more we decrease the sampling time of the MPC controller, since the rise time of the controller becomes less negligible.

9.1.2 Full Formulation

Alternatively it is possible to use the accelerations as input signal. This has two big advantages:

1. It is more related to direct dynamics control.

2. The speed profile is automatically continuous, and hence is more realistic and easier controllable.

For this purpose a new state dynamics formulation is needed.

\[
\begin{align*}
x &= \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix}, \quad \dot{x} = f(x, u) = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \end{pmatrix},
\end{align*}
\]

(9.2)

with the same output function.

Remark. It is worth mentioning that this formulation would be completely equivalent to the pure dynamic model if an exact feedback linearization would be applied to the latter. However, for this to be viable an model with a good quality needs to be given.

9.1.3 Output Function

We define the position of the end palm of the robot hand as the output of the system. This function was derived using the DH convention in the last chapter and is given as seen in (9.3).

\[
y = h(x, u) = \begin{pmatrix} -0.4 \cos(\Theta_1) \sin(\Theta_2) - 0.578 \sin(\Theta_4)(\sin(\Theta_1) \sin(\Theta_3) \\ 0.578 \sin(\Theta_4)(\cos(\Theta_1) \sin(\Theta_3) + \cos(\Theta_2) \cos(\Theta_3) \sin(\Theta_1)) \\ 0.4 \cos(\Theta_2) + 0.578 \cos(\Theta_2) \cos(\Theta_4) + 0.31 \\ - \cos(\Theta_1) \cos(\Theta_2) \cos(\Theta_3) - 0.578 \cos(\Theta_4) \cos(\Theta_1) \sin(\Theta_2) \\ - 0.4 \sin(\Theta_1) \sin(\Theta_2) - 0.578 \cos(\Theta_4) \sin(\Theta_1) \sin(\Theta_2) \\ + 0.578 \cos(\Theta_3) \sin(\Theta_2) \sin(\Theta_4) \end{pmatrix}
\]

(9.3)

9.2 Setup

9.2.1 Determination of the Disturbance

To be in a proper and useful position for designing the RMPC controller we first need to find out how big the disturbance of the system is, i.e. what was denoted as \( d_w \) earlier. For this we randomly sample trajectories and analyze disturbance values for both system formulations.

Simplified Formulation

First we will have a look at the simplified formulation from (9.1).
Figure 9.1: Disturbance (i.e. error) and trajectory of $\Theta_1$ of the real robot when limiting the velocity to $0.7\text{rad/s}$. It is especially large when the system accelerates or brakes abruptly, e.g. in the very beginning.

**Disturbance of Real Robot** For designing the RMPC we first wanted to find out how big the real disturbance on the system can get. For this purpose we sampled the state space on the real robot to get an idea of how big the deviation of the real system is compared to our assumed nominal system. To get an idea about this, please look at Figure 9.1. We only plotted the measurements on $\Theta_1$ since the magnitude of the error is about the same for all joints. It is likely that the error becomes larger with increasing demanded accelerations, since the system is not able to accelerate instantly (and without delay) to the desired velocity. Looking at Figure 9.1 unveils that Assumption 9.1 is at least partly fulfilled for low velocities and a slow sampling rate of $0.4\text{s}$, since the disturbance remains moderate.

When trying to design a suitable RMPC controller the deviation of the disturbed function $x_{k+1} = f_w(x, u, w)$ compared to the nominal system $x_{k+1} = f(x, u)$ is the important characteristic. To guarantee stability we need to be in a position to account for all possible disturbances along the horizon.

**Disturbance as Function of the Velocity** We came up with the idea that very likely the disturbance is related to the velocity; higher velocity means higher disturbances. In this case the error seems to be mainly created due to high intractable acceleration. Nevertheless, high acceleration steps can also be avoided by limiting the velocity.

We sample random trajectories in simulation and plotted the $\infty$-norm of the disturbance against the $\infty$-norm of the joint velocities (in $\text{rad/s}$). This is visualized in Figure 9.2. To be in the position to design a suitable and sufficient RMPC controller we limited the speed of the joints to a value only giving rise to manageable disturbances. In this case we perform a reverse approach: we check for which disturbances we are still able to design a suitable controller. Then afterwards we limit the velocity s.t. we can make sure to keep the disturbance within these bounds.

**Remark.** Note that for this thesis we are only properly looking at the disturbance values in simulation. We found the model to be relatively accurate in this respect and since we are doing the validation in simulation only, we are mainly interested in this value in this thesis.
Figure 9.2: Disturbance in relation to the joint velocities for the simulation.

(a) Disturbance of $\Theta_1$ for the simulation.

(b) Disturbance of $\dot{\Theta}_1$ for the simulation.

Figure 9.3

Full Formulation

This time we use the full description with 8 states as seen in (9.2).

Disturbance of the Real Robot In the very beginning we wanted to find out how the real system is behaving compared to our assumptions. For this we limited the joint velocity to $0.7 \frac{\text{rad}}{\text{s}}$ and the angular acceleration to $0.7 \frac{\text{rad}}{\text{s}^2}$. In Figure 9.3 we can see the deviation on the state (angle and angular velocity) of the first joint of the real robot.

Disturbance as Function of the Velocity and Acceleration We proceed similar to the last subsubsection for the simplified formulation. However, this time we are not only plotting the disturbance against the velocity, but against the velocity and the acceleration. This makes sense, since these are the values of interest we are able to control. The corresponding plots for the control problem formulation can be seen in Figure 9.4a and Figure 9.4b.

All in all, it also becomes clear that the error of this full formulation is much smaller than the one of the simplified case. The explanation for this is that it is physically possible to control the accelerations by applying a specific torque on the electric motors leading to a continuous angle and angular velocity profile. In contrast, it is not possible to directly control the velocity, and the shorter the sampling time gets, the more negative impact is created by the not infinitesimal rise
Chapter 9. Experimental Results on Apollo Robot

(a) Disturbance as a function of the velocity and the acceleration for the simulation from perspective 1.
(b) Disturbance as a function of the velocity and the acceleration for the simulation from perspective 2.

Figure 9.4

time of the low level controllers, hence leading to a deviation from the expected state, in our case taken into account as disturbance.

9.2.2 RMPC Design

The main theory for the used RMPC scheme was introduced in Section 3.2 and the needed knowledge and calculations for its design were introduced in Chapter 4. Since our goal is to control the output function and we do not have explicit knowledge about the state $x_s$ leading to that desired output, we include the dynamic set point tracking which we introduced earlier (Section 3.1) in our RMPC scheme. Additionally we need to consider constraints on the state, input and output functions. The constraints on the output function are of nonlinear nature, due to the fact that the output function itself is nonlinear.

Formulation

The resulting RMPC optimization problem in this case is the one seen in (9.4) and is analogous to the one derived in Section 4.5. It contains everything we need to be in a position to track the output function and therefore to control the robot while satisfying all constraints and guaranteeing recursive feasibility under disturbances.

\[
V_N(x(t), y) = \min_{u(\cdot|t), x, y, \alpha} J_N(x, y; u(\cdot|t)^*, y_s, x_s, u_s, \alpha) \tag{9.4a}
\]

subject to

\[
x(0|t) = x(t), \tag{9.4b}
\]

\[
x(k+1|t) = f(x(k|t), u(k|t)), \tag{9.4c}
\]

\[
g_j(x(k|t), u(k|t)) + c_j \frac{1 - \rho^k}{1 - \rho} \bar{w} \leq 0, \tag{9.4d}
\]

\[
x_s = f(x_s, u_s), \quad y_s = h(x_s, u_s), \tag{9.4e}
\]

\[
\frac{\bar{w}}{1 - \rho} \leq \alpha \leq \frac{g_j(x_s, u_s)}{c_j}, \quad \forall j = 1, ..., p \tag{9.4f}
\]

\[
|x(N|t) - x_s|^2 \leq (\alpha - \delta f)^2, \tag{9.4g}
\]

\[
k = 0, ..., N - 1, \quad j = 1, ..., p,
\]
Figure 9.5: Visualization of the nonlinear output constraint which avoids that the robot could hit itself.

with $J_N$ as introduced in Section 3.1.2 and repeated here:

$$J_N(x, y; u(\cdot|t), y_s, x_s, u_s) = \sum_{k=0}^{N-1} l(x(k|t) - x_s, u(k|t) - u_s) + V_f(x(N|t) - x_s, y_s) + V_O(y_s - y_t). \quad (9.5)$$

In this case the function $f(\cdot, \cdot)$ described either the dynamics in (9.1) or in (9.2). The function $h(\cdot, \cdot)$ denotes the output function in (9.3) describing the 3D position in the cartesian space.

**Constraints**

As seen in (9.4d), we include the constraints in a robust way (i.e. with some constraint backoff) in our optimization problem. We will introduce the possible constraints in the following.

**State and Input Constraints** For the state constraints see Table 8.3 with the transformation in Table 8.1 and for the full formulation from (9.2) Table 8.3. For the input constraints consider either (for the simplified formulation (9.1)) Table 8.3 or (for the full formulation (9.2)) Table 8.4. All of these constraints are of polytopic nature and can be written as $g_j(x, u) \leq 0$ for $j \in \{1, \ldots, p-3\}$ since there will be 3 output constraints.

More interesting are the constraints on our output function since they describe the space where the robot is supposed to move later on in the application. In the following we will derive the function $g_q(x, u)$ for $q = p-2$, $g_r(x, u)$ for $r = p-1$ and $g_p(x, u)$ and their derivatives with respect to $x$, i.e. the state of the system. $y_1(x)$, $y_2(x)$ and $y_3(x)$ denote the first, second and third entry of the system output (i.e. the $x$, $y$ and $z$ coordinates of the robot hand palm) which are given as a function of the system state. With (4.84) we can then calculate the $c_j$ values corresponding to those constraints.

**Robot Body Avoidance** We use the constraint description to make sure that the robot does not hit its own body with the end effector. $y^b$ denotes the base of the quadratic function which defines where the robot is allowed to move and where it is not. A visualization of the constraint can be seen in Figure 9.5.
In the coordinate system of the robotic arm this can be written as the following condition:

\[ y_3 - y_b^3 \geq -(y_2 - y_b^2)^2. \]  

We want to bring this condition to the expected form for \( g_q(x,u) \): \( g_q(x,u) \leq 0 \). With this prerequisite we get:

\[
\begin{align*}
g_q(x,u) &= -y_3 + y_b^3 - (y_2 - y_b^2)^2 + 2y_2y_2 \\ &= -y_3 - y_2^2 + 2y_b^2y_2 + y_b^3 - (y_b^2)^2 \\ &= -y_3 - y_2^2 + y_b^3 \geq 0.
\end{align*}
\]

Now we are interested in the Jacobian of this function (see (4.84) for more information), which can be written as the following:

\[
\frac{\partial g_q(x,u)}{\partial x} = -\frac{\partial y_3}{\partial x} - 2y_2 \frac{\partial y_2}{\partial x}.
\]

**Obstacle Avoidance** Additionally we use quadratic constraints to avoid obstacles in the operation space. In principle we are able to handle any obstacle which can be described by a differentiable function. In this specific case we do the calculations for two types: horizontal and vertical obstacles. A visualization of a vertical obstacle can be seen in Figure 9.6. \( y^o \) denotes the apex of the function which defines the obstacle. In our case we define the obstacles with the help of quadratic functions. The constant \( C \) defines the steepness of the quadratic function, we choose it to be 2.

The calculations for the horizontal obstacle can be done as in the following:

\[
\begin{align*}
y_3 - y_b^3 &\leq Cy_1^3 - C(y_1^2)^2 + C(y_2 - y_b^2)^2. \\
\text{The corresponding function } g_r(x,u) \text{ is defined as follows:} \\
g_r(x,u) &= y_3 - y_b^3 - Cy_1^3 - C(y_1^2)^2 + 2Cy_1^3y_1 - Cy_2^2 - C(y_2^2)^2 + 2Cy_2^3y_2 \\ &= y_3 - Cy_1^3 - Cy_2^2 + 2Cy_1^3y_1 + 2Cy_2^3y_2 - Cy_1^3 - C(y_2^2)^2 - y_b^3 \leq 0.
\end{align*}
\]
For the derivative we get:
\[
\frac{\partial g_r(x, u)}{\partial x} = \frac{\partial y_1}{\partial x} - 2C(y_1 - y_1^o)\frac{\partial y_1}{\partial x} - 2C(y_2 - y_2^o)\frac{\partial y_2}{\partial x}.
\]
(9.14)

Now we do the calculations for the vertical obstacle:
\[
y_1 - y_1^o \geq -C \left((y_3 - y_3^o)^2 + (y_2 - y_2^o)^2\right)
\]
and therefore
\[
g_p(x, u) = -y_1 - Cy_2^2 - Cy_3^2 + 2Cy_2y_2^o + Cy_3y_3^o + y_1^o - C (y_2^o)^2 - C (y_3^o)^2 \leq 0.
\]
(9.16)

For the derivative we get:
\[
\frac{\partial g_p(x, u)}{\partial x} = - \frac{\partial y_1}{\partial x} - 2C(y_2 - y_2^o)\frac{\partial y_2}{\partial x} - 2C(y_3 - y_3^o)\frac{\partial y_3}{\partial x}.
\]
(9.17)

It is obvious that we still have the location of the obstacle in this derivatives of \(g_r\) and \(g_p\).

However, since the function is linear in this argument and we are interested in the extrema for finding \(c_r\) and \(c_p\) we only need to consider the maximum and minimum values of \(y_1^o\) and \(y_2^o\). Since the robot arm only has a total length of 1.1m we choose the following bounds:

\[-1.2 \leq y_1^o \leq 1.2, \text{ and } -1.2 \leq y_2^o \leq 1.2, \text{ and } -1.2 \leq y_3^o \leq 1.2.
\]
(9.18)

**RMPC Offline Computations**

We need to do some offline calculations to get the required parameters for our RMPC controller. Of course we designed many RMPC controllers along the way: for 4 dimensions, for 8 dimensions, with a sampling time of 0.4s for the RMPC robot control, with a sampling time of 0.01s and 0.04s for the controller we used to train the neural network and so on. We will demonstrate the design exemplary for one specific configuration: sampling time of \(h = 0.4s\) and 8 dimensions, i.e. the full formulation from (9.2). The code for doing the offline computations as well as the results for the other designs can be found in the code coming with this thesis.

**Remark.** As already described earlier, we assume for all of the calculations CT prestabilization due to the fact that it can be applied at a frequency of 1ms. If this would not be possible, the prestabilization would also need to be included in the calculations in DT.

1. The choice of the CT contraction rate is very important for the behavior of the system. Higher contraction rates enable the system to compensate for larger disturbances. However, when the contraction rate is significantly higher than the time constant of the system or the sampling time this can lead to jerky behavior. In this case we choose a contraction rate of \(\rho_c = 2.5\) (i.e. the time constant corresponding to the RMPC sampling rate). To have some buffer to always finish our calculations in the sampling time we choose a time horizon of \(T = 1.2\), i.e. 3 steps into the future. According to Figure 9.4a we see that the maximum DT error is given as \(d_w = 0.06\) (\(rad\) or \(rad/s\)). For a sampling time of 0.4s this is equivalent to \(d_{w,c} = 0.15\). We put some tolerance on top and choose \(d_{w,c} = 0.25\).

2. After having decided for the important design parameters, we need to calculate suitable \(P_3\) and \(K_3\) (refer to Section 4.6.1). For this we consider the condition on the contraction rates as well as the constraints on our state and input (see Section 8.4). Furthermore we require \(\tilde{w}_c\) to be smaller than a specific value, 0.32 in our case. This is the smallest bound on \(\tilde{w}_c\) which
still preserves feasibility of the optimization problem. We get the following two matrices:

\[
P_\delta = \begin{pmatrix}
4.7311 & 0 & 0 & 0 & 0.9776 & 0 & 0 & 0 \\
0 & 4.7311 & 0 & 0 & 0 & 0.9776 & 0 & 0 \\
0 & 0 & 4.7311 & 0 & 0 & 0 & 0.9776 & 0 \\
0 & 0 & 0 & 4.7311 & 0 & 0 & 0 & 0.9776 \\
0.9776 & 0 & 0 & 0 & 0 & 0.3911 & 0 & 0 \\
0 & 0.9776 & 0 & 0 & 0 & 0 & 0.3911 & 0 \\
0 & 0 & 0.9776 & 0 & 0 & 0 & 0.3911 & 0 \\
0 & 0 & 0 & 0 & 0.9776 & 0 & 0 & 0.3911 \\
0 & 0.3911 & 0 & 0 & 0 & 0.9776 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.3911 & 0 & 0.9776 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.3911 & 0.9776 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9776
\end{pmatrix}, \quad (9.19)
\]

\[
K_\delta = \begin{pmatrix}
-12.0983 & 0 & 0 & 0 & -5 & 0 & 0 & 0 \\
0 & -12.0983 & 0 & 0 & 0 & -5 & 0 & 0 \\
0 & 0 & -12.0983 & 0 & 0 & 0 & -5 & 0 \\
0 & 0 & 0 & -12.0983 & 0 & 0 & 0 & -5
\end{pmatrix}, \quad (9.20)
\]

3. As a next step we compute the required values for \(c_j\). We have the following constraints:

- Constraints 1, ..., 16 are the polytopic state constraints according to Table 8.2 transformed to our DH convention used for the control (Table 8.1).
- Constraints 17, ..., 24 are the input constraints. As we have seen in the last section we limit those to \(8 \text{ rad/s}^2\) for each of the 4 joints.
- Constraint 25 is the constraint on the output function to avoid the robot hitting itself (compare (9.9)).
- Constraint 26 is the constraint on the output function to avoid the robot hitting any horizontal obstacle at position \(y^o\) (compare (9.13)).
- Constraint 27 is the constraint on the output function to avoid the robot hitting any vertical obstacle at position \(y^o\) (compare (9.16)).

The resulting values for \(c_j\) are summarized in Table 9.1, 8.4 and 9.2.

Table 9.1: Values for \(c_j\) of the state and input constraints, i.e. for \(j = 1, ..., 24\).

<table>
<thead>
<tr>
<th>(j)</th>
<th>1, 2</th>
<th>3</th>
<th>4</th>
<th>5, 6</th>
<th>7, 8</th>
<th>9 ... 24</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_j)</td>
<td>0.2234</td>
<td>0.4498</td>
<td>0.4322</td>
<td>0.2242</td>
<td>0.3164</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 9.2: Values for \(c_j\) of the output constraints, i.e. for \(j = 25, 26, 37\).

<table>
<thead>
<tr>
<th>(j)</th>
<th>25</th>
<th>26</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_j)</td>
<td>0.7906</td>
<td>4.2143</td>
<td>4.5535</td>
</tr>
</tbody>
</table>

4. We check that \(\rho_c\) and \(w_c\) satisfy our requirements.

5. We calculate \(\bar{w}_c,\text{allowed} = \frac{\rho_c}{\rho_c,\text{max}} = 2.5\), which is bigger than the \(\bar{w}_c\) we imposed. This means that a design with a \(\bar{w}_c\) value being approximately 5 times bigger is still feasible.

6. We check whether \(\bar{w}_c,\text{allowed, output} = \frac{g_j(x_{\text{test}}, u_{\text{test}})}{c_j}\) is bigger than \(\bar{w}_c\) for realistic \((x_{\text{test}}, u_{\text{test}})\).

7. We compute the maximum tube size \(\bar{s}_f = \frac{1-e^{-\rho_c T}}{\rho_c}w_c = 0.1188\).

8. We define the matrices \(Q\) and \(R\) as following: \(Q = I_{8 \times 8}\), \(R = I_{4 \times 4}\).
9. Finally, we compute the terminal cost, which is the exact infinite horizon cost using the corresponding $K_\delta$-controller. This results in:

$$P_f = \begin{pmatrix} 16.1534 & 0 & 0 & 0 & 6.0905 & 0 & 0 & 0 \\ 0 & 16.1534 & 0 & 0 & 0 & 6.0905 & 0 & 0 \\ 0 & 0 & 16.1534 & 0 & 0 & 0 & 6.0905 & 0 \\ 6.0905 & 0 & 0 & 0 & 3.8181 & 0 & 0 & 0 \\ 0 & 6.0905 & 0 & 0 & 0 & 3.8181 & 0 & 0 \\ 0 & 0 & 6.0905 & 0 & 0 & 0 & 3.8181 & 0 \\ 0 & 0 & 0 & 6.0905 & 0 & 0 & 0 & 3.8181 \end{pmatrix}. \quad (9.21)$$

10. Compute the DT equivalents of the parameters: $\rho_d = e^{-\rho_c h} = 0.3679$, $\bar{w}_d = \frac{1-e^{-\rho_c h}}{\rho_c} \bar{w}_c = 0.0791$.

### 9.2.3 Learning

#### Curse of Dimensionality

At this place we do a small calculation example to underline one main problem for the learning. Including parallelization, we assume a computing time of $0.05 \frac{\text{seconds}}{\text{sample}}$. This results in $72,000 \frac{\text{samples}}{\text{hour}}$ or $1,728,000 \frac{\text{samples}}{\text{day}}$. If we want the calculations not to take longer than 7 days we can approximately evaluate a maximum of

$$\#_{\text{max}} = 1.2 \cdot 10^7 \text{ samples}. \quad (9.22)$$

#### Sample State Space

If we now use the better system description from (9.2) considering only the first 4 joints, we already get up to 8 state dimensions and up to 3 reference dimensions, fully neglecting the possibility of having additional obstacle positions. With the maximum number of samples from (9.22) it is clear that in any case only few samples can be gridded per dimension. E.g., if we look at the simplified system description from (9.1) and we want to sample the 3D reference we get a sampling space of 7 dimensions (4 state and 3 output). This leads to the following amount of samples for each dimension:

$$\#_{\text{dim1}} = (1.2 \cdot 10^7)^{1/7} = 10.26 \frac{\text{samples}}{\text{state dimension}}. \quad (9.23)$$

For the more comprehensive full formulation this only leads to

$$\#_{\text{dim1}} = (1.2 \cdot 10^7)^{1/11} = 4.4 \frac{\text{samples}}{\text{state dimension}}. \quad (9.24)$$

samples per dimension. However as we have shortly explained in Section 3.3.2 neural networks are often still able to learn suitable function mappings.

#### Sample in Output Space

Another approach is to sample in the output space, i.e., we set realistic end-effector positions and then calculate the input for a given reference and state. By doing so, we have only 3 dimensions we are sampling over. In principle this leaves us space to get

$$\#_{\text{dim2}} = (1.2 \cdot 10^7)^{1/3} = 228.9 \frac{\text{samples}}{\text{output dimension}}. \quad (9.25)$$

samples for each of the six dimensions. in practice we would implement this by simulating realistic trajectories. Our hope is then, that we cover most of the relevant state space areas for being able to learn a working policy. This is similar to the work in [28], where they also sampled full trajectories for learning the approximation.
Chapter 9. Experimental Results on Apollo Robot

Sampling

For the sampling we first started with simplified systems to get a feeling about how well the network is able to approximate the implicit MPC policy function. We started with a 2D-2D configuration, i.e. only enabling the first two joints while providing as a two dimensional reference lying in a plane in front of the robot. After we found out that we are able to properly learn an approximation of the RMPC control policy we increased the dimensions. Most of the sampling rounds we started were 4D-3D, i.e. all 4 joints and a 3 dimensional output reference for the simplified formulation in (9.1) without any obstacles.

Random Sampling  On the one hand, we chose to randomly sample the possible dimensions. We had two main reasons in mind:

1. By sampling it randomly, the NN has at least a rough idea of how to approximate each region. Since in our validation a sampled trajectory counts as violated as soon as on AMPC step along this trajectory does not guarantee stability we especially strongly interested in avoiding outliers.

2. We chose random sampling over gridding, since this easily allows the addition of further points afterwards. For gridding, introducing more points would not be trivial and you need to wait every time until the whole space is sampled. Otherwise this would lead to an imbalanced sample point distribution.

Sampling Trajectories  When trying to fit the network to the randomly sampled policy points from the last paragraph, we encountered some difficulties (more information in Section 9.3.3). Therefore we decided to additionally sample random trajectories, i.e. to repeatedly choose a random starting point and to sample the subsequent trajectory until steady state is reached. This has the big advantage that our samples better represent the robot use case and therefore lead to a smaller approximation error. By doing so, it is possible to reach better performance and to simplify the validation.

Network Architecture

For the network architecture we only tried standard fully connected neural networks. We varied them a lot with respect to size, deepness, the loss function, the training procedure and the batch size. The following plots are all taken with respect to the training of the 4-dimensional simplified formulation.

Configuration  For the activation functions within the network we chose ReLu activations. Only in the last layer, typical for regression problems, we chose a linear activation. As optimizer we trusted in Adam [29] as most people are doing nowadays. Our error function was a mean squared error (MSE).

Network Size  For the performance in general it seems that a specific network size is required. However, after a specific epoch (approximately epoch 11 for this example) the performance improvement saturates. This can be seen in Figure 9.7 where we show the training result of different network configurations from 157 hidden units (5,760 trainable parameters) through to 2700 hidden units (1,103,804 trainable parameters). It is clear that bigger network lead to better performance until a specific point of stagnation. All networks shown in this plot had between 4 and 8 hidden layers.

Depth of the Networks  Even though we reached nice convergence and proper results we were not able to get the training error down to 0. However, this is also not necessarily needed since we only need to get good enough results such that our assumptions from the RMPC design are satisfied. Nevertheless, we tried to further decrease the error by choosing some really deep networks.

Figure 9.7: Training loss for different network configurations of different size. The bigger the network, the lower the loss. This training was done on the simplified 4-dimensional system without any prestabilization, i.e. the overall values the network should learn have moderate values in magnitude (between $-2.3$ and $2.3$).

Figure 9.8: Training loss for two networks with very different depth. The deeper network converges slower but reaches about the same performance. In this case, again the learning for the simplified 4-dimensional system is shown. However, with prestabilization which explains the higher loss compared to Figure 9.7.

Deep networks usually have higher potential expressiveness but take longer to train. As it can be seen in Figure 9.8 their learning curve looks typical for training deep networks. However, for this specific example they are not able to achieve significant higher performance than shorter networks with similar number of hidden neurons which train significantly faster.

**Overfitting**  In general, we did not encounter too big problems regarding overfitting. An exemplary comparison of the training and validation set can be seen in Figure 9.9.

**Regularization**  Regularization helps to keep the size of the weights smaller, compared to the case without regularization. Since overfitting is not a big problem in our case there was also not real need for regularization. Anyhow, we tried it but could only encounter decrease in performance.

**Network Training**  

The network training turned out to be quite important. As we use the prestabilized dynamics our system input becomes the following:

\[ u_{sys}(\tau|t) = v_{RMPD} + K_dx(\tau|t), \forall \tau \in [0, T). \]  

(9.26)
Chapter 9. Experimental Results on Apollo Robot

Figure 9.9: Visualization of the performance on the training and validation set. We didn’t face big problems in terms of overfitting for this learning problem. The performance on the whole validation set is taken after the epoch while the performance on the training set is the average of the error during the training interval of one epoch. This explains why the error of the validation set is sometimes significantly smaller.

(a) Value of the loss function on the training set when directly learning $v^{MPC}$

(b) Value of the loss function on the training set when learning $u^{sys}$.

Figure 9.10

Since we impose limits on $u^{sys}$, this also means that the absolute values of the entries of $v^{RMPC}$ can become relatively large (i.e. values in the scale of 100, 200). This becomes especially problematic for a strong prestabilization we impose for fast operation.

Naively using $v^{RMPC}$ for the training often produces decent results but still retains a bigger MSE error since the overall output can be increased by several magnitudes. We know that the values of $v^{RMPC}$ need to include an offset to compensate for $K\delta x(0|t)$ at the beginning of the interval. This term $K\delta x(0|t)$ keeps $u^{sys}(0|t)$ and $u^{sys}(h|t)$ within the bounds. We can equalize for this RMPC internal compensation by simply adding $K\delta x(0|t)$ to $v^{RMPC}$ which is equivalent to directly learning $u^{sys}(0|t)$ (see (9.26)).

The big advantage of this is that we enable the network to get a better precision with respect to the actual unknown core of the approximated policy. The comparison of the error of the two plots can be seen in Figure 9.10. As a consequence, in the online implementation we then need to subtract $K\delta x(0|t)$ to get $v^{RMPC}$.

9.2.4 Validation

For the validation we use the results from Section 5.4. For this thesis we do the validation in simulation but also considering the model mismatch, the delay of the controller as well as additional
errors, e.g. induced by the integrators. In the next section (9.3) we will show our approach to find AMPCs for the simplified formulation (9.1) as well as for the full formulation (9.2).

Sampling Trajectories

The validation in simulation is done as following: We use the feedback gain $K_\delta$ to steer the system to a random state; the starting point of the trajectory. When we are approximately in a steady state, we start the trajectory control trying to reach a random output set point. As soon as we reached the state well enough, we sample the next trajectory and so on.

Criterion

The validation is done according to Section 5.3.2, in particular using the indicator function of (5.19). In our validation procedure we need the different trajectories to be i.i.d. Since the resulting trajectory is deterministic with respect to the start state of the trajectory and the desired end-effector position, also the trajectory is i.i.d. for i.i.d. sampled start points and desired references.

In our specific case, we not only have the error due to the erroneous approximation of the neural network as in [19], but also additional errors such as the model mismatch. We have two possible ways of how to deal with this.

Full Validation along Trajectories  This formulation is most straightforward. We sample our trajectories in deterministic simulation as described before and then validate the following criterion:

$$|f_w(x, \pi_{\text{approx}}) - f(x, \pi_{\text{MPC}})|_{p_3} = |d_w|_{p_3} \leq \bar{w}, \ \forall x \in X_i.$$  \hfill (9.27)

Bound Additional Disturbances  For the other formulation we need the following assumption.

**Assumption 9.2.** All additional disturbances, including model mismatch, time delay and integrator errors, are bounded according to the following condition:

$$|f_w(x, \pi_{\text{MPC}}, d) - f(x, \pi_{\text{MPC}})|_{p_3} = |d_{w, \text{model}}|_{p_3} \leq \bar{w}_{\text{model}}, \ \forall x, \ \text{with } \pi_{\text{MPC}} = u^*(0|t).$$  \hfill (9.28)

We can now sample the trajectory in the same way as described before. However, at every time step we are not comparing the expected following state $x^{+} = f(x, \pi_{\text{MPC}})$ with the observed system state $x^{+}_w = f_w(x, \pi_{\text{approx}}, d)$, but instead with the expected state when applying the approximated input $x^{+}_{w, \text{approx}} = f(x, \pi_{\text{approx}})$ to the nominal system.

**Theorem 9.1.** Assume we have the admissible bound $\bar{w}_{\text{model}}$ for our RMPC controller error. If

$$|f(x, \pi_{\text{approx}}) - f(x, \pi_{\text{MPC}})|_{p_3} \leq \bar{w}_{\text{approx}} = \bar{w} - \bar{w}_{\text{model}},$$  \hfill (9.29)

with $\bar{w}$ from our RMPC design and $\bar{w}_{\text{model}}$ from Assumption 9.2 holds, then the guarantees of our underlying RMPC design also hold for the deployed AMPC.

**Proof.**

$$|f_w(x, \pi_{\text{approx}}, d) - f(x, \pi_{\text{MPC}})|_{p_3}$$

$$= |f_w(x, \pi_{\text{approx}}, d) - f(x, \pi_{\text{approx}}) + f(x, \pi_{\text{approx}}) - f(x, \pi_{\text{MPC}})|_{p_3}$$

$$\leq |f_w(x, \pi_{\text{approx}}, d) - f(x, \pi_{\text{approx}})|_{p_3} + |f(x, \pi_{\text{approx}}) - f(x, \pi_{\text{MPC}})|_{p_3}$$

$$\leq \bar{w}_{\text{model}} + \bar{w}_{\text{approx}} \overset{(9.29)}{=} \bar{w}. \hfill (9.33)$$

And therefore according to our RMPC design, we know that the requirements are satisfied.
9.2.5 Implementation

In the beginning we planned to use ACADO for the solving of our RMPC problem. It has several advantages such as faster computations and performance functionalities such as the real time iteration (see Section 6.2.1 for more information). Since our optimization problem is relatively complex with many additional non-standard decision variables and the nonlinear and non-quadratic output function we encountered problems. Therefore we decided to use the Casadi C++ API for our experiments.

For the RMPC implementation, being at time step k, we always predict the state at step k+1 and solve the RMPC problem for this predicted step. then in the next interval this input is applied. For the AMPC implementation we compute the reference in the beginning of the interval based on the current state and apply it during this interval. This is reasonable since the evaluation of the AMPC takes less than 1 ms.

9.3 Results

We started our robot experiments with the state dynamics formulation in (9.1) in the hope that Assumption 9.1 is fulfilled. We used this approach to set up our pipeline and test it in simulation as well as on the real world robot. We observed a computation time of approximately 250 ms why we decided to choose a sampling time of 400 ms. By making the code more efficient but at the same time introducing more complex functionalities we kept this sampling time for the normal RMPC control till the very end. For the AMPC approach we planned to use the simplified formulation due to advantages in the number of dimensions. Nevertheless, in the end we also implemented an AMPC version based on the full description.

In the following we will only explain the final version of the RMPC controller in more detail which is based on the full 8 dimensional state description. Additionally we will show results for the AMPC approaches based on both formulations.

9.3.1 RMPC Control of the Apollo Robot

One of the main goals of this Master Thesis is to propose a well working RMPC based control scheme for the Apollo robot.

Workflow

We worked iteratively, i.e. we started by controlling the joint angels without considering the position of the robotic hand. For the simulation we started based on very simple integrator simulations. For an example of the simplified formulation, refer to Figure 9.11. In a next step we integrated the output set point tracking before we continued to the more advanced simulation of SL (e.g. also including inertias etc.), see Figure 9.12. Moving to SL was particularly difficult, since our software framework already needed to meet the standards regarding command communication and real time behavior.

In parallel we were also developing the code for calculating the offline parameters (compare Section 9.2.2). From a theoretical perspective there were also some challenges such as the nonlinear output constraints and the RPI terminal set for non static set points. During the project we were also thinking about trying to integrate robust collision avoidance. However, having our desired demonstration, we then included the obstacles as described in Section 9.2.2.

Final Result

For making the final RMPC controller for the Apollo robot work involved all steps described in this thesis, especially Chapter 4 and this chapter introduce a lot of special cases and show how to do the calculations. The result is a working framework which controls the robot in a dynamic way being able to robustly satisfy all constraints in the state, input and output, and hence it is even able to avoid obstacles. The framework is written in a modular way making it easy to also
9.3. Results

(a) Simulation of MPC for angle set point tracking with $T = 20$, $\Delta t = 0.5$ and $u_{max} = 0.05$. The horizon is not long enough to reach the desired position. The horizon is of course long enough.

(b) Simulation of MPC for angle set point tracking with $T = 20$, $\Delta t = 0.5$ and $u_{max} = 1$. In this case the horizon is not long enough to reach the desired position.

Figure 9.11

(a) Real set point tracking to reach the endeffector position $[0, 0, 11/10]^T$. $T = 10$, $\Delta t = 0.5$.

(b) Simulation in SL. The white lines depicts the trajectory of the robotic arm in the last few seconds.

Figure 9.12

include further constraints, consider multiple obstacles or to aim for even faster movements by further decreasing the sampling time. In terms of timing we certainly admit for improvement.

The final result can be observed online\textsuperscript{1}.

9.3.2 AMPC Control of the Simplified Formulation

Workflow

In the beginning of this work we were skeptical regarding the curse of dimensionality (compare Section 9.2.3). This was also the reason why we had in mind to model the system very simplistic only containing a 4 dimensional state vector.

During the development of the RMPC controller (we did it for the simplified as well as for the full formulation in parallel) this formulation turned out to be quite decent. Therefore we also decided to train an AMPC controller from this simplified description. From the very beginning our main goal was to show that the AMPC formulation can bring a huge gain in performance.

\textsuperscript{1}https://www.youtube.com/playlist?list=PLG18yuSJCUApnS-UDaTjnhCtIfZmAfXJ
Figure 9.13: Disturbance $d_w$ due to model mismatch for the choice of 10\(\text{ms}\) as sampling time. Unfortunately, the disturbances are too big to give any useful guarantees.

Therefore after building a sample AMPC controller without prestabilization and a sampling time of 0.4s we directly moved to the faster variant: prestabilization and a sampling rate of just 0.01s.

As described before, we do the offline calculations under the assumption of having continuous time prestabilizing feedback. Therefore, also the allowed threshold on the disturbance is in continuous time. As described in Section 9.2.2 we thought that $d_{w,c} = 0.25$ should be a good option. At the same time we were able to easily find a RMPC design for this choice.

Afterwards we designed an RMPC controller one more time. This time we chose $d_{w,c} = 1.3$. Our goal for this design was to make the underlying RMPC controller as robust as possible. And with $\rho_c = 100$, this $d_{w,c}$ marked the last possible value.

**Validation**

Special about the validation is that this is really the first time we see the system operating at this speed, since we were not able to simulate it realistically using the original RMPC controller due to lack of speed.

When we investigated the AMPC controller trained with the version compensating for $d_{w,c} = 0.25$ we were quite surprised. The error was much bigger than we expected. The main difficulty lies in the fact, that we are not able to test the error in advance without training the NN as a result of the lacking speed of the RMPC computations.

In Figure 9.13 it is possible to see the overall disturbance over time, occurring from the model mismatch only (under the assumption that our RMPC control command is perfect). This influence is the $\infty$-norm of the disturbance during the validation process. The maximum errors are approximately 0.04. For comparison: $d_{w,c} = 0.25$, approximately means in this case $d_w = 0.0025$ even though we are not looking for the infinity norm in the validation but rather on the incremental Lyapunov function (Section 5.4).

This was the reason why we started another design round including stronger prestabilization and a much higher admissible value for the disturbance, i.e. $d_{w,c} = 1.3$.

Since we are validating trajectories, we would need a percentage close to zero. Also for the redesigned AMPC controller the results in this case are not good enough. The model error is just too big. The explanation for this is that Assumption 9.1 is to lesser extent valid, the smaller the sampling time gets.
That is also the reason why we decided to use the full formulation for designing the AMPC controller after all. For more information on this, refer to Section 9.3.3.

**Final Result**

The final result can be observed online\(^2\). The result and its dynamic behavior is really impressive. However, as described before we are not in a position to give any statistical guarantees.

**9.3.3 AMPC Control of the Full Formulation**

**Workflow**

After the two detailed iteration steps for the simplified formulation described in the last section, we decided to give it another try with the full eight dimensional formulation. We expected to be able to bring down the model mismatch significantly, since the model is physically tractable and therefore the CT disturbance should be more or less invariant to the sampling time.

We designed the controller for disturbances of \(d_{w,c} = 1.0\) together with a contraction rate of \(\rho_c = 10\). Furthermore, we designed this controller for a sampling time of \(T = 40\) ms. This means exactly a speed up by a factor of 10 compared to the RMPC-based control.

**Validation**

As introduced in Section 5.4, for the validation we need to sample i.i.d. trajectories. However, in this framework it is not as simple as for the other case, since also the velocity is included in the starting point of the coordinate system. In our case we are setting the velocity in the starting point to zero and are therefore only able to give statistical guarantees for full trajectories starting with zero velocity.

Since we do not do any predictions for the usage of the AMPC controller but to really calculate the input in the current interval based on the real system state, the validation even incorporates the non zero computational delay of the network, which is even neglected for most practical RMPC computations. Thanks to the fact that the evaluation takes less than 1 ms, the influence of this delay on the whole interval of 40 ms is luckily rather small.

The validation procedure itself is way more promising in this case since the model mismatch is much smaller, also for the worst case scenario. Nevertheless, doing the validation is still not simple since it is more difficult for the neural network to learn the policy compared to the simplified formulation. This is mainly due to two reasons:

1. The whole problem is now higher dimensional, since we have now a 8 dimensional formulation of the state vector compared to the 4 dimensional one in the simplified case. This leads to the fact that our sampled data regarding the sample dimensions is even more sparse than in the simplified formulation.

2. Since we operate at high sampling rates now (i.e. either 10 ms or 40 ms) the underlying controller is able to control the system in a much more dynamic way. Looking at the data, it seems that the controller is acting like a bang-bang kind of controller when the corresponding position in the task space of the state is far away from the reference set point. Since we can operate at high speeds this distance is much smaller than for the case of a sampling time of 400 ms, hence also the proportion of the controller acting at its input limits is much higher. This is confirmed by the experiments, which show that the NN AMPC controller performs well in the beginning of each trajectory and poorly towards the end. We explain this by the fact that the NN has mainly seen state and reference far away and therefore inputs being in the constraints, since we sampled randomly in state and reference.

In this case it might be helpful to use the classification approach again as introduced in Chapter 7 for the chemical stirred-tank reactor.

As a result we decided to sample trajectories, which we expect to be extremely beneficial.

\(^2\)https://www.youtube.com/playlist?list=PLG18yu6JCUpmS-ODeTjnhrCt1hFZmAfXJ
Sampling Trajectories  For the sampling of the data points we decided for a combination of randomly sampling data points as well as sampling trajectories. We sampled approximately 45,000,000 randomly chosen independent points and additional 45,000,000 points by sampling over trajectories. For the final phase of the training we then decided to use 45,000,000 million more points, by sampling over trajectories, resulting in overall approx. 135,000,000 million points. This took overall 30 days of sampling on 2 computers. The points sampled over trajectories turned out to be practically more relevant.

Training  The training turned out to be more difficult than expected. This is mainly due to the previously mentioned curse of dimensionality. As you can see in Table 9.3, we didn’t manage to get down the mean squared error (MSE) significantly below 0.7 for the validation set, even though we applied the technique of not learning $v_{MPC}$ directly, but rather $u_{sys}(0|t)$. Since we are not really interested in the loss of our trained network but more in the worst case scenarios, we also tried to train the network using different loss functions, e.g. the mean-quartic-error (MQE). Unfortunately this didn’t work.

On the other side, since our training data contains only a little noise, training with bigger batches can lead to significantly lower loss on the training set, since we are doing the gradient descent in more relevant directions. Unfortunately, the network is not able to generalize this decrease in the error also to the validation set. Therefore, using too big batches leads to more overfitting. The full overview of our attempts can be seen in Table 9.3.

Final Result

The final result can be observed online. After sampling the trajectories, we used approximately 30,000,000 random data points and 60,000,000 data points coming up from the trajectory sampling.

\footnotesize{\begin{itemize}
\item[3]https://www.youtube.com/playlist?list=PLGl8yuSJCUApmS-QDcTjnhCtIhfZmAfXJ
\end{itemize}}
Table 9.3: Overview of all our training attempts. It can be seen that deeper networks usually lead to better performance, but take longer to train, compare [1]. Furthermore, using bigger batches increases the speed of convergence on the training set but not necessarily improves performance on the validation set. Using another loss than the very typical mean squared error (MSE) did not turn out to help in increasing performance of the network.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training Procedure</th>
<th>Epochs (+ means continued training)</th>
<th>Batch Size</th>
<th>MSE / MQE Training Set</th>
<th>MSE / MQE Validation Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small dataset (90 mio. data-points)</td>
<td>Shallow Network (≈ 3.5 mio. trainable parameters)</td>
<td>40</td>
<td>512</td>
<td>1.5129</td>
<td>1.7843</td>
</tr>
<tr>
<td></td>
<td>Shallow Network</td>
<td>+40</td>
<td>512</td>
<td>1.4087</td>
<td>1.6167</td>
</tr>
<tr>
<td></td>
<td>Shallow Network</td>
<td>+80</td>
<td>512</td>
<td>1.4639</td>
<td>1.9086</td>
</tr>
<tr>
<td></td>
<td>Deep Network (&lt; 2 mio trainable parameters)</td>
<td>40</td>
<td>512</td>
<td>0.7727</td>
<td>1.0025</td>
</tr>
<tr>
<td></td>
<td>Deep Network</td>
<td>+40</td>
<td>512</td>
<td>0.6672</td>
<td>0.9649</td>
</tr>
<tr>
<td></td>
<td>Deep Network</td>
<td>+80</td>
<td>512</td>
<td>0.6585</td>
<td>0.9279</td>
</tr>
<tr>
<td></td>
<td>Deep Network, bigger layers (≈ 4.7 mio. trainable parameters)</td>
<td>40</td>
<td>512</td>
<td>0.6340</td>
<td>0.9736</td>
</tr>
<tr>
<td></td>
<td>Deep Network, bigger layers</td>
<td>+40</td>
<td>512</td>
<td>0.5303</td>
<td>0.9025</td>
</tr>
<tr>
<td></td>
<td>Deep Network, bigger layers</td>
<td>+80</td>
<td>512</td>
<td>0.5333</td>
<td>0.7232</td>
</tr>
<tr>
<td></td>
<td>Deep Network, bigger layers, MQE loss</td>
<td>40</td>
<td>512</td>
<td>34.55 / 523.0649</td>
<td>34.56 / 532.2351</td>
</tr>
<tr>
<td></td>
<td>Deep Network, bigger layers, regularized</td>
<td>20</td>
<td>512</td>
<td>1.3284</td>
<td>1.6915</td>
</tr>
<tr>
<td></td>
<td>Deep Network, even bigger layers (≈ 6 mio. trainable parameters)</td>
<td>40</td>
<td>512</td>
<td>0.7321</td>
<td>1.1084</td>
</tr>
<tr>
<td></td>
<td>Even Deeper Network</td>
<td>14</td>
<td>512</td>
<td>34.5533</td>
<td>34.5676</td>
</tr>
<tr>
<td></td>
<td>Deep Network (based on Deep Network, bigger layers, after 80 (+40) epochs)</td>
<td>+80</td>
<td>2048</td>
<td>0.4088</td>
<td>0.7467</td>
</tr>
<tr>
<td></td>
<td>Deep Network (based on Deep Network, bigger layers, after 80 (+40) epochs)</td>
<td>+80</td>
<td>16384</td>
<td>0.3638</td>
<td>0.8157</td>
</tr>
<tr>
<td></td>
<td>Deep Network (based on Deep Network, bigger layers, after 80 (+40) epochs)</td>
<td>+80</td>
<td>65536</td>
<td>0.3532</td>
<td>1.2388</td>
</tr>
<tr>
<td></td>
<td>Deep Network (based on Deep Network, bigger layers after 80 (+40) epochs), only updated if validation performance improved</td>
<td>+80</td>
<td>16384</td>
<td>0.3592</td>
<td>1.0641</td>
</tr>
<tr>
<td></td>
<td>Deep Network (based on Deep Network, bigger layers after 80 (+40) epochs), MQE loss from this point</td>
<td>+80</td>
<td>512</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>Big Dataset (135 mio. data-points)</td>
<td>Deep Network (based on Deep Network, bigger layers), regularized after 80 (+40) epochs</td>
<td>+80</td>
<td>16384</td>
<td>0.5415</td>
<td>1.0456</td>
</tr>
<tr>
<td></td>
<td>Deep Network, bigger layers</td>
<td>80</td>
<td>2048</td>
<td>0.3638</td>
<td>0.8157</td>
</tr>
</tbody>
</table>
Chapter 10

Conclusion

10.1 Summary

In this work we presented a novel way to design a RMPC as well as an explicit AMPC for the control of a real world robotic platform. We integrated several important results from literature which helped us to achieve the desired behavior. We also broke down the RMPC scheme from [36] to the for us necessary components. Additionally, we extended the scheme by introducing terminal ingredients for dynamic set point tracking. We also guided the reader through the complex concepts of this approach and how the derivations and computations are done in a practical example.

We verified the importance of the concept of AMPC with guarantees introduced by Hertneck et al. [19] and enhanced it by using a better validation procedure to be in a position for providing statistical guarantees more easily. We checked our framework by simulating the same example as given in the corresponding work [19]. For this we introduced more advanced concepts for sampling and doing the regression ending up with a huge decrease in offline computation time.

Furthermore, we implemented the RMPC scheme by Koehler et al. [36] for the very first time on a complex physical system. We showed good performance on the Apollo robot while being able to guarantee robust constraint satisfaction, including nonlinear output constraints for obstacle avoidance. The framework is designed in a very modular way, making it easy to integrate and include additional constraints, objectives or concepts. The main drawback of this approach is the speed of the evaluation of our implementation. Nevertheless, including the full robot control in one optimization problem (i.e. also the motion planning) while guaranteeing robust constraint satisfaction is superior to other concepts in many respects. Usually, many different software components are needed and safety can not be guaranteed in the way we are doing it. The lack in computational simplicity motivates the usage of our AMPC approach.

One drawback of approximating the control policy is the reduction in flexibility. E.g. introducing a second or third obstacle in our RMPC formulation can be achieved by adding an additional code to the optimization. In contrast, the AMPC controller needs to be provided with new training data and needs to be retrained after every minor modification. This is the same drawback as for general explicit MPC.

We achieved an order-of-magnitude performance improvement by learning an AMPC controller from two formulations, a simplified one describing the states by the joint angles only and a more comprehensive formulation also including the velocities and using the accelerations as input. For the first formulation we were not able to give useful guarantees on full trajectories due to the large model mismatch. The full formulation leads to a lower model mismatch and hence, to a more promising starting point for the AMPC. However, at the time of submitting this thesis we didn’t manage to find a sufficiently precise NN approximation, mainly due to the curse of dimensionality. However, we still see big opportunities for future work, by using more tailored learning procedures. The evaluation of the neural network takes less than $1ms$ while solving the RMPC problem takes $200 – 300ms$. For the sampling we were able to highly parallelize this evaluation (i.e. sampling multiple points at a time), however for the practical implementation we did not find a simple way
to accelerate this process. Therefore our design easily allows a speed up by a factor of 300 in theory. For our demonstration we chose a sampling time of 10ms for the simplified case, and a sampling time of 40ms for the more comprehensive one. Especially for well defined problems without too many unknowns, e.g. for flight control of quadrotors, this approach has a lot of potential.

10.2 Future Work

The extensions and implementations of the RMPC scheme of Koehler et al. [36] allowed us to do full robot control while guaranteeing stability and robust constraint satisfaction on Apollo Robot. Additionally, this master thesis significantly advanced performance and functionality of the underlying AMPC approach compared to the start of the project [19]. At the same time, we identified promising aspects for further improvements and interesting questions for future work.

To make the practical implementation even more meaningful and relevant for the robotics community, an extension to the real world robot dynamics instead of the kinematic description would be worthwhile. Moreover, including exact collision avoidance instead of our approximated description of the obstacles based on differentiable functions, would reduce conservatism and as a result improve the performance. Another potential extension would be to introduce an admissible non-constant model mismatch $d_w$ for the RMPC design, e.g. $d_w(\dot{\Theta})$. We expect the consequent RMPC design to be significantly less conservative. Using parametrized matrices $K_\delta$, $P_\delta$ could reduce the conservatism even further.

Also very interesting would be to directly tackle the expensive computational demands of the RMPC optimization problem even further. Potential solutions could be to formulate the problem in a more efficient way or to use real-time iteration approaches (see A.3.2). This could result in a higher practical relevance of the RMPC approach, reducing the need of our presented approximation and hence, to be more flexible.

Furthermore, we see potential in further improving the used learning techniques to even better overcome the problem of the high dimension. By further removing outliers in the control signal, the validation would be directly applicable.

Really interesting but challenging would also be to find an approximation which generalizes better. One potential outcome could then be that the network is able to handle different numbers of obstacles without having observed similar situations during training before.

Additionally, it would be interesting to investigate the learning part for the high dimensional robotic use case even more. An example would be to find out whether a classification approach as introduced in 7.2.5 would achieve better performance over the standard regression approach.

10.2.1 Relation to Guided Policy Search

As part of this master thesis we also identified the connection of guided policy search (GPS) [39, 38, 37, 26] to our deployed approach. Especially one paper attracted our attention [68], since it explains an approach of doing MPC guided policy search.

Particularly useful for understanding the main idea of GPS, is the work in [50]. During the emergence of this thesis, we talked several times about GPS and tried to understand in more detail the advantages of GPS over our approach and how we could potentially try to introduce some kind of guarantees to the GPS algorithm. A promising direction for future work is to compare the generalization properties of GPS to the ones from our work. Especially interesting will be to investigate, in which situations the improvement of the reference policy might be helpful. Additionally, the surrogate cost needs to be adjusted online in GPS to ensure convergence. This is something we don’t need in our formulation, since it is already given as a result of our statistical guarantees. For cases such as our real world use case, advantages of GPS are not obvious and are worth exploring in future work.

We can say with confidence that at the current state our approach differs significantly from the work in GPS, since we don’t include any of its characteristic properties. For future work combining the advantages of both approaches (for the case of MPC guided policy search [50]) might be really promising.
Appendix A

User Manual

A.1 Introduction

This is the User Manual which is aside with this Master Thesis. It should help to understand the important steps to reproduce the results presented in this thesis. This includes an overview of the existing code and its functionalities, the installation of the needed frameworks as well as the requirements for deploying the corresponding code on the robot.

In this user manual not all intermediate implementations are described but only those ones which can be useful to the reader. The goal in every single step during the development was to keep the software as modular as possible, i.e. uncoupling the theoretical concepts from the used frameworks, s.t. a change to another framework would be as easy as possible.

A.2 Overview

Most of the code was written in C++ due to real time or at least time-critical demands. However, also Python and MATLAB code is used for training the ML networks, plotting results and performing offline calculations. All programming languages are separated and can be obtained from the name of the corresponding Git repository.

A.2.1 Git Repositories

All repositories are given on the git-amd server and can be accessed through gitlab\textsuperscript{1}.

The used Github repositories are the following (order depending on the point in time of the core implementation):

- `ampc_task`
- `cstr_ampc_cpp`
- `apollo_ampc_cpp`
- `apollo_sampling_cpp`
- `ampc_python`
- `ampc_malab`

We will go into more detail about all of them in Section A.4.

\textsuperscript{1}https://git-amd.tuebingen.mpg.de/
A.3 Frameworks

As described earlier in Chapter 1, the main part of this Master Thesis is to develop and implement an RMPC/AMPC control approach for Apollo robot (see Fig. 1.1) and to deploy it in practice. Due to the time critical demands of the robot control - all hardware related command signals need to be updated at a frequency of $1kHz$ - the whole robotic control framework (Sec. 6.4) is implemented in real-time C++. To be compatible with this framework and to fulfill the demanding requirements, we decided in the very beginning to implement the code which will actually run on the robot in C++. Some other offline calculations, e.g. for training the network, validating theoretical ideas or designing the RMPC controllers are implemented in Python and MATLAB.

A.3.1 Robotic Control Framework

As described earlier in Section 6.4, we use the Simulation Laboratory framework [54] for passing the signals to the robot. Information about the functioning of SL can be found in the official user manual [54]. Using SL is not trivial since it is not written in the most lucid way. However, with the help of my supervisor Vincent Berenz it was manageable. After having implemented the main code base, it runs fast and reliable. Its biggest advantage is that almost no modifications need to be done when moving from simulation to real robot experiments. Therefore, all interfaces and communication issues can already be checked offline in advance.

Important to notice is that the ROS-Apollo version on the git-and server only works with Ubuntu 14.04 at the time of this thesis. The more general, catkinized version also runs with Ubuntu 16.04.

We included additional ROS communication, real-time safe threading as well as overwriting configurations files when launching the program for reaching our goals.

Creating your own Task

SL works great as long as you adhere closely to the official guidelines SL provides, i.e. writing the code within the frame of a so called SL task. How to create such a task is well described in the AMD wiki.

Examples within our Code Base

The best example for how to write your code within SL is the ampc_task package. For more information on this, refer to A.4.1.

A.3.2 MPC Framework 1 - ACADO

For the control tasks in our chemical stirred-tank reactor simulation example, we used the ACADO Toolkit\(^2\). The manual is given in [22]. This framework is tailored for solving optimal control problems and its usage seems to be very intuitive in the beginning, however, as soon as more special demands appeared in my case I also hit on problems. The user manual is not complete and the official support seems to have been stopped in the meanwhile. Nevertheless it’s an incredibly fast problem solver which is absolutely sufficient for most standard model predictive control tasks.

Installation

The usage of ACADO is relatively straightforward. All information can be found on the official installation page\(^3\). The source code can be cloned from GitHub. The way we installed it, was to download this source code, compile it at a specific location, and then install it, i.e. by moving the header files and shared libraries to the correct locations. This can be done as following:

\(^2\)https://acado.github.io
\(^3\)https://acado.github.io/install_linux.html
Real-Time Iteration (RTI) Scheme

Especially interesting for real-world use of complex MPC schemes is the real-time iteration scheme [23, 14, 63]. It uses the fact that optimal solutions of an MPC control problem are usually continuous, e.g. the solution doesn’t change too much between neighboring points and convergence is a lot faster when using neighboring solutions as warmstart for the new solution attempt. We tried it and it’s not especially hard to use, however, since we were aiming for truly converged solutions we abandoned this solution procedure.

Examples within our Code Base

Examples can be found in cstr_ampc_cpp, also how to include the code within a catkin project. The RTI scheme is used within cstr_ampc_cpp/RTI whereas the normal ACADO scheme is used within cstr_ampc_cpp/check_condition, cstr_ampc_cpp/simulation and cstr_ampc_cpp/sample_optimal. The ACADO integrator for simulation purposes can be found in cstr_ampc_cpp/simulation.

A.3.3 CasADi

After we encountered too big problems with ACADO, we started to use CasADi [3] not only in MATLAB but also in C++. CasADi\(^4\) is very well documented for MATLAB and Python, however, really poorly for its C++ version. The example code which comes with the software only shows very simple use cases. It took quite some effort to find efficient ways for implementing our desires. Since all other APIs are also using the C++ backend we knew that our goals must also be implementable in C++, just not exactly the syntax.

Installation

Since all versions of CasADi come with a C++ backend, each of them also delivers all principally needed header files and libraries. Therefore, the easiest way to use CasADi in C++ is to get one of the official releases and then link against the corresponding libraries. In our case, we always used the Python realase by

\[$\textit{pip install casadi}\$\]

However, this only worked in a straight forward manner for us when using Ubuntu 14.04. Nevertheless, since we didn’t put much focus on running it for Ubuntu 16.04, we still think that there must be a way. Another possibility is to build CasADi from source, which definitely would have performance advantages. Since, also third party libraries such as iPOPT need to be included as well, we decided not to do it during this project.

Examples within our Code Base

Certainly the most comprehensive example of how to formulate complex MPC problems in C++ CasADi - including constraint tightening, additional decision variables and matrix algebra - can be found in our MPC predictor, which is for example in ampc_task/predictor/src/mpc_predictor_casadi.cc.

\(^4\text{https://web.casadi.org}\)
A.3.4 Machine Learning Framework - Tensorflow

Since this project contains a big learning part, a suitable ML framework is needed. The whole framework should be implemented in C++, therefore, desirably also the ML framework should have a C++ API to avoid unnecessary programming language interfaces. We are using NNs for the approximation of our AMPC controller. One of the most powerful and most common frameworks for the usage of NNs is TensorFlow which also provides a C++ API. We decided to use TensorFlow in the beginning. Alternatives would have been PyTorch or Caffe.

In our case we are using TensorFlow 1.13. There are multiple ways of using TensorFlow. In the following we will try to describe the most important findings. Multiple APIs are provided by TensorFlow. The recommended and most comprehensive one is the Python API. Especially for training and creating custom models this is the way to go. For the training we mainly use the KERAS NN library which is part of TensorFlow’s core library in the meanwhile and offers a higher-level set of abstractions, which makes it easier to rapidly develop NN-ML models. Despite the skeptical opinion in the web regarding the C++ API, we decided to deploy it for the inference in our own C++ programming framework.

Details

CUDA If one wants to make use of one or multiple GPUs, CUDA needs to be installed on the system. CUDA 10.0 is only supported by new GPUs with driver versions 410.x or higher and only usable with TensorFlow versions newer than 1.13.0. We always used CUDA 9.0 and it fulfilled all our requirements. The official information on how to set up CUDA can be found on the TensorFlow webpage\(^5\). However, when running Ubuntu 16.04 there is a nice script prepared by Vincent Berenz which was recently published as open source software. Additionally also CUDNN needs to be installed to use TensorFlow with CUDA. There exist two main ways of using TensorFlow.

Prebuilt Version The usual way of using TensorFlow is to have recourse to the prebuilt version available online. For Ubuntu this is usually done by using pip\(^6\). If the motivation is to use the Python API this is certainly the easiest solution and usually works. However binary files are then not optimized to the computer and can therefore have worse performance compared to the case of building TensorFlow from source. This version also works with the C-API, however, whose usage we would clearly discourage.

Building TensorFlow from Source Building TensorFlow from source\(^7\) is more tedious and takes longer. However, the resulting framework will offer better performance. Additionally and for us even more importantly, this is necessary to use the C++ API. Unusual is the fact that Bazel\(^8\) is required to build TensorFlow. We recommend to install Bazel in the /home-folder since the installation might block a significant amount of disk space. After having compiled the package it is possible to built a .whl package. This can then be installed on the system by using pip resulting in the same behavior as the prebuilt package but with better performance. To be in a position to use the C++ API in a catkin workspace additional steps are required. Then it’s possible to create besides the general shared library libtensorflow_framework.so also a shared library called libtensorflow_cc.so to use the C++ API. This step is described in more detail in the next Sec. because it’s not explained in the official installation guide. In the catkin workspace, we can then link against this shared library and include all needed header files. Strange is that bazel creates some of the headers during compilation, so you will not find all of them in the TensorFlow software package.

---

\(^5\)https://www.tensorflow.org/install/gpu
\(^6\)https://www.tensorflow.org/install/
\(^7\)https://www.tensorflow.org/install/source
\(^8\)https://docs.bazel.build/versions/master/install.html
Appendix A. User Manual

Installation

Prebuilt Version To install the pip package:

```
$ # Prerequisite: Python development environment
$ sudo apt update
$ sudo apt install python-dev python-pip
$ sudo pip install -U virtualenv # system-wide install
$ # Install tensorflow (possibly in virtual environment)
$ pip install --upgrade tensorflow
```

Building TensorFlow from Source After having installed Bazel you can compile the TensorFlow source folder.

The important part here is that we build the shared library libtensorflow_cc.so. For this we can manually build the required shared library by using

```
$ bazel build --config=opt --config=cuda //tensorflow:tensorflow_cc # --config=cuda only for gpu
$ # Sometimes the previous doesn't work, then:
$ bazel build --config=opt --config=cuda //tensorflow:libtensorflow_cc.so
```

Additionally it is necessary to include all required headers. For more information on that, have a look at our coded examples.

Debugging While building in bazel the error: 'int'objectattribute'__doc__'isread-only. Then we need to uninstall enum and install enum34 (pip install enum34).

Examples within our Code Base

We use TensorFlow almost within all of our repositories. Examples for the training with the Python API can be found in ampc_python/2_apollo/nn/.

On the other side examples for the inference in C++ can be found in ampc_task/predictor/src/ampc_predictor_tf.cc. It is important to note, that before being able to use models trained in KERAS, they need to be frozen. This is for example done in the end of the Python file ampc_python/2_apollo/nn/neural_network_predict_train_multiple.py.

A.4 Developed Software Packages

A.4.1 ampc_task

This is the the package needed for doing the RMPC and/or AMPC control on Apollo robot. As described earlier, SL requires real-time safe C++ code. There are several possibilities for including non real-time safe code - such as CasADi or TensorFlow in our case - in a real-time framework. We decided to do this by spawning real-time safe threads, i.e. provided by Xenomai in our case. Furthermore, we also include non-real-time-safe communication over ROS in our package, again by using suitable threads.

The package file has a configuration file (in ampc_task/config/ampc_config.yaml) where the mode of operation as well as important parameters (e.g. the sampling time) can be set. The big advantage of handing over details about the configuration in this way is, that no recompilation is needed after doing any changes.

Furthermore, we created two suitable starting scripts called ampc.bash and ampc_real_robot.bash which can be found on the AMD server under start_robot/config/. They allow not only to launch all needed packages (such as SL and Vicon), but also to copy our needed configuration files, e.g. for the robot low-level gains, to the correct position.
Details on Implementation

We decided to create an instance of the so called *ComputeControl* class within our SL task. This class contains several variables as members, which can be used to do the actual robot control. We also use these, i.e. the variable $v_{MPC}$ in particular, to pass the calculated inputs of the RMPC and AMPC respectively. The class spawns two real-time threads, of which one of them creates instances of the non-real-time-safe predictors, i.e. *MpcPredictor* and *AMPCPredictor*. During operation these class instances are used to obtain the computed results and update the corresponding member variables of the *ComputeControl* class instance. Additionally, we also include ros communication, generation of trajectories and others within this repository.

An overview about the most important code snippets can be found in the following. The location of the code is given with respect to the repository ampc_task.

cmake  Includes FindCASADI.cmake, a file which is needed to find the corresponding CasADi libraries within our operating system. More information on the inclusion of the other packages can be found in the CMakeLists.txt.

config  This folder is really important, since it contains the main config file ampc_config.yaml. Here parameters such as the sampling time, the used TensorFlow model or the mode of operation can be defined. Furthermore, the parameters needed for the operation of SL are defined in the corresponding .cf-files. Those ones are copied to the right locations before runtime, when using our provided .bash scripts (see Section A.4.1).

framework  In here, the main framework is defined, from initializing the class instance of ComputeControl over spawning the real time threads to calling the input computations.

predictor  In the predictor, the two ways of doing the control are defined, i.e. by using the RMPC as well as the AMPC controller. The header file predictor.h defines all information needed for building the interfaces with the underlying classes, while the actual code is given in ampc_predictor_tf.cc for the AMPC and in mpc_predictor_casadi.cc for the RMPC implementation.

ros  Within this folder, the communication to ros is defined, i.e. the visualization of our reference and the obstacles as well as the communication to the vicon_bridge, in order to get information of the pose of the objects within the scene. This pose is currently updated with a frequency of 25 Hz, but can easily be changed within the code.

task/src  In here, the actual code for the SL task is defined. Since the structure is mainly predefined by SL there was not much room for changes.

trajectories  In this folder, code for the generation of the trajectories is given. The trajectories can be used to either sample the state space as realistically as possible or to validate our underlying AMPC controller.

A.4.2 apollo_ampc_cpp

This package covers all of the standard Apollo related code. It was used to build up the framework, to do the first simulations as well as to validate the AMPC controller. It also shows a standard catkin package which includes ACADO, CasADi as well as TensorFlow in C++. 
Details on Implementation

**src/calc_disturbance**  This code snippet is used to check whether the assumptions were violated during the sampling of the trajectory. It takes the .csv file as input which is created by the ampc_task during trajectory sampling operation. It then counts for how many trajectories the conditions are violated and shows how big the corresponding error on the input is.

**src/plot**  This is in principle the python matplotlib library for plotting graphs in C++. However, since it was just wrapped to C++, it doesn't support all functionalities and should therefore only be used for the prototyping.

**src/simulate**  This software package was used to check the correct functioning of both the RMPC as well as the AMPC controller.

A.4.3  **apollo_sampling_cpp**

We used the apollo_sampling_cpp code for sampling of the data points we then used as groundtruth for the training. The code was particularly designed to run fast in a parallel fashion as well as to be reliable. Once, we've run the code for more than 20 days on two computers with all available threads doing computations.

Details on Implementation

**src/mpc_predictor_casadi**  This is again the CasADi RMPC controller as described earlier. It is used to solve the optimization problems for random points or along trajectories, depending on the mode of operation.

**src/rmpc_sampling**  This executable offers the functionality for sampling points. In the beginning of the code you can define whether you want to do it for the 4-dimensional or 8-dimensional case, whether you want to sample randomly or using trajectories and much more.

A.4.4  **cstr_ampc_cpp**

This repository contains all C++ code we have for the chemical stirred-tank reactor (CSTR). It is used from simulation over sampling to validation.

Details on Implementation

**src/sample_optimal**  This is the code we used for sampling the solution points which we can then use as a groundtruth for the training. This code offers 3 different ways of doing the sampling: uniformly, recursively and randomly. For each of the 3 implementations an executable is computed.

**src/check_conditions**  This collection of code is used to do the validation for the CSTR controller. It also provides functionalities such as measuring the duration of a NN iteration and an integrator implementation for integrating the trajectory.

**src/simulate**  This code is used to simulate the CSTR using both, the RMPC as well as the AMPC controller. In the end of this code, the trajectory for both controllers are directly created.

**src/RTI**  This is an code example where we used the real-time iteration (RTI) scheme of ACADO. It worked, however, since we were interested in the fully converged solutions, we didn’t use it for the sampling.
A.4.5 ampc_python

This repository contains all of the Python code written for this Master Thesis. It was used for plotting and training the NNs for the chemical stirred-tank reactor (CSTR) simulation example as well as for the robot experiment.

Details on Implementation

The repository has two larger sub-directories: 1_cstr and 2_apollo.

1_cstr Here the NN model definitions for the classification as well as for the regression are given. Furthermore, also code for training those models is provided. The latest version for the training is given in neural_net_predict_train_2.py. Furthermore, in plotting/ the code for plotting the results of the regression and the classification is given.

2_apollo In this directory we have several sub-folders. 2_apollo/nn defines all of the learning models and trains the networks. 2_apollo/cluster provides scripts which can be used on the MPI Tuebingen cluster. 2_apollo/utility can be used to create a bigger .npy file out of the many .csv files created by the sampling. Last but not least, 2_apollo/plotting provides scripts for plotting the training progress and for plotting a 3 dimensional trajectory.

A.4.6 ampc_matlab

This repository provides the MATLAB code which was implemented during the project. Two main directories are given: apollo/ and cstr/.

cstr/ In cstr/, the code for the offline calculations of our simulation example is provided. This might be of interest, because also calculations for nonlinear system dynamics are covered.

apollo/ In apollo/, code for writing the optimization problem in CasADi, deriving the end-effector position of Apollo, determining the disturbance level as well as for the RMPC offline calculations is given. The most interesting part are probably the offline calculations which we used for all designs for the 4-dimensional as well as the 8-dimensional cases.

A.5 Deploying the Code on Apollo Robot

For running the code on Apollo robot, not many additional steps compared to running it in simulation with SL are needed.

Check Real-Time Requirements

In the beginning it is necessary to run the SL simulation again on the Xenomai machine, to check whether real-time behavior is given. For this the following command can be run:

$ watch -n 1 cat /proc/xenomai/stat

Changing numbers indicate, that the real-time behavior can not be fulfilled. If real-time behavior is fulfilled, we can go to the next step.

Setup of Apollo Robot

In the next step Apollo robot needs to be set up. This is well explained in the corresponding Wiki entry\(^9\). In short what needs to be done in the KUKA console is the following:

\(^9\)//atlas.is.localnet/confluence/display/AMDW/How+to+start+Apollo+and+control+it+from+SL
1. Tell the robot arm that it is equipped with a Barrett hand.
2. Perform the gravity compensation by "massaging" the robot arm.
3. Go to position control and select the suitable script.
4. Start SL by using our ampc_real_robot.bash script.

During the experiments it is important to always keep the emergency stop in your hand and to always be aware to press it.

A.6 Direct Re-usability of the Code

We tried to make it as easy as possible to reuse our code afterwards. On the one hand we created a treep project - Treep is the MPI open source package manager - in an early stage of the project. On the other hand, we are currently working on a docker image of our C++ software, to make it possible to run the code despite complex dependencies.

A.6.1 Treep

Treep is an open-source package manager provided by the Max Planck Institute for Intelligent Systems\textsuperscript{10}. By providing suitable project files, it helps to install full projects consisting of several repositories. It also directly places the source code at the right position. For the AMD repositories, this file is given on the AMD server in treep_amd_clmc. For our project for the Apollo robot - i.e. consisting of SL, our ampc_task and many more - we also created a corresponding project called AMPC. It includes all necessities for using the Vicon system, SL, as well as our own code. Nice about treep is that it also allows to include full projects within other projects, e.g. in our AMPC project we include the full ROS-Apollo project. The full project can be installed as following:

\begin{verbatim}
$ pip install treep
$ mkdir apollo_ws
$ cd apollo_ws
$ git clone git@git-amd.tuebingen.mpg.de:amd-clmc/treep_amd_clmc.git
$ treep --clone AMPC
\end{verbatim}

Then you are ready to use the provided catkin workspace within ./workspace.

A.6.2 Docker

Since we are using several different software packages which are not fully straightforward to install, we are currently working on a docker image of our code. This image will contain not only ampc_task but all C++ code which requires packages such as ACADO, CasADi or TensorFlow. The big advantage of docker is, that it allows to build containers which can even emulate whole operation systems with almost no loss in computational speed. The docker image will enable usage of our code in almost every unix-environment. More information on docker can be found online\textsuperscript{11}.

\textsuperscript{10}https://pypi.org/project/treep/
\textsuperscript{11}https://www.docker.com
Bibliography


Title of work:
Learning-based Approximate Model Predictive Control with Guarantees
Joining Neural Networks with Recent Robust MPC

Thesis type and date:
Master Thesis, 31.08.2019

Supervision:
Johannes Köhler
Dr. Vincent Berenz
Prof. Dr. Raffaello D’Andrea
Dr. Sebastian Trimpe

Student:
Name: Julian Nubert
E-mail: nubertj@ethz.ch
Legi-Nr.: 14-939-151
Semester: FS 2019

Statement regarding plagiarism:
By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Declaration of Originality:

Hakodate, 1. 9. 2019: J. Nubert