TTK4550 Specialization Project

Nonlinear Model Predictive Control for Robot Manipulator Trajectory Tracking

Author:
Martin Albertsen Brandt

Supervisors:
Jan Tommy Gravdahl
Esten Ingar Grøtli
Phillip Maree

Department of Engineering Cybernetics
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

December 17, 2020

Abstract

The trajectory tracking problem is a standard problem for motion planning and control of robot manipulators. Advances in recent years in embedded hardware and numerical optimal control solvers motivates applying Nonlinear Model Predictive Control approaches to solve this problem in real-time. In this report, we study Nonlinear Model Predictive Control for Cartesian space trajectory tracking for robot manipulators in further detail, for its inherent ability to adhere to nonlinear constraint while planning the robot motion. The real-time feasibility of this approach is investigated when using the complete robot dynamics for prediction. Furthermore, we consider how to include collision avoidance and singularity avoidance in the resulting optimal control problem. The developed real-time Nonlinear Model Predictive Control framework was tested in simulation and on a 6 DOF robot manipulator arm. Real-time feasible computation time and low tracking errors were observed. Testing obstacle avoidance of moving objects demonstrated satisfactory accuracy, yet at the cost of increased computation time. The capabilities of the framework were also demonstrated by having the robot grasp a moving object. The results showed how Nonlinear Model Predictive Control based on the full robot dynamics is promising for safety-critical applications with real-time requirements where the robot has to avoid moving obstacles.

Preface

This report presents my specialization project at the Norwegian University of Science and Technology, and is done in collaboration with SINTEF Digital. I started this work on Nonlinear Model Predictive Control for robot manipulators during my summer internship in the Robotics and Control group in SINTEF Digital, and the continuation of this work is presented in this report. I want to thank Research Manager Sture Holmstrøm for giving me the ability to do this while being in such a rewarding work environment with great colleagues. I also want to give a big thanks to my supervisors Prof. Jan Tommy Gravdahl, Dr. Esten Ingar Grøtli and Dr. Phillip Maree for providing valuable feedback and discussions throughout the semester, and especially to Esten who have always been available to discuss my progress and challenges during the past seven months.

Martin Albertsen Brandt
December 17, 2020

Contents

List of Tables

List of Figures

Acronyms

DH	Denavit-Hartenberg
DOF	Degrees of freedom
ERK	Explicit Runge-Kutta
ERK4	Explicit Runge-Kutta of 4th order
IRK	Implicit Runge-Kutta
KKT	Karush-Kuhn-Tucker
MPC	Model Predictive Control
NLP	Nonlinear program
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
QP	Quadratic program
ROS	Robot Operating System
RTDE	Real-time Data Exchange
RTI	Real-time Iteration
SLERP	Spherical linear interpolation
SQP	Sequential Quadratic Programming
URDF	Unified Robot Description Format

Notation

$\boldsymbol{I}_m$	$m \times m$ identity matrix
$\nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$	Gradient of scalar function $\boldsymbol{f}$ with respect to $\boldsymbol{x}$
$\nabla^2_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$	Hessian matrix of scalar function $\boldsymbol{f}$ with respect to $\boldsymbol{x}$
$[\boldsymbol{x}]_\times$	Skew-symmetric matrix of vector $\boldsymbol{x}$
$\det(\boldsymbol{A})$	Determinant of matrix $\boldsymbol{A}$
$\mathrm{tr}(\boldsymbol{A})$	Trace of matrix $\boldsymbol{A}$
$\|\boldsymbol{x}\|$	Euclidean norm of vector $\boldsymbol{x}$
$\|\boldsymbol{x}\|^2_{\boldsymbol{Q}}$	Squared Euclidean norm of vector $\boldsymbol{x}$ weighted by matrix $\boldsymbol{Q} \succ 0$, i.e. $\boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x}$

Chapter 1

Introduction

1.1 Motivation

The emergence of autonomous robots and especially its applications to human-robot collaboration can provide many possible benefits to numerous industries. The ability for robots to safely monitor, cooperate and contribute to production processes in complex environments is evidently useful for many industrial applications. Yet in order for autonomous robots to be effectively used in the industry, many challenges need to be addressed. Requirements for safety, agility and robustness must be met, as well as real-time guarantees. This motivates the development of motion planning and control methods for robots to be able to react in real-time to dynamic environments and changing tasks while operating. How to achieve this is an open and active area of research, and in this work a Cartesian space trajectory tracking and collision avoidance framework based on optimal control is presented, to tackle a small part of this problem.

Optimal control, which considers how to optimally solve such control problems over a time horizon while adhering to constraints, provides a useful framework for motion planning and control. Especially Nonlinear Model Predictive Control (NMPC), which closes the loop by repeatedly solving the discretized nonlinear optimal control problem, is of interest because of the requirements for robustness and real-time feasibility. Furthermore, one of the main benefits of NMPC is that constraints can be considered explicitly when finding the control input. This makes it possible to handle the nonlinear kinematics and dynamics of the robot directly in the optimization problem, and changing tasks, safety constraints and collision-avoidance constraints can be considered explicitly. The main limitation is the computation time, especially for systems with fast dynamics. However, the advances in recent years in embedded hardware and numerical optimization solver architectures makes NMPC interesting to investigate for robotic systems with fast and nonlinear dynamics. In the following, this will be investigated using the 6 degrees of freedom (DOF) UR10e robot

manipulator arm, both in simulation and using the real robot.

1.2 Objective

The main objective of this report is to investigate the performance and real-time capability of NMPC for trajectory tracking for robot manipulator arms. Being able to perform complex and dynamic tasks while reacting to the environment is of great interest for robot manipulators, both because of their use in production lines, and their applications to mobile robots, ships, satellites and more. We will investigate how to incorporate obstacle avoidance, self-collision avoidance and singularity avoidance in the NMPC problem. Moreover, motivated by the emphasis on human-robot collaboration, we will investigate if these methods can be used to grasp and avoid moving objects, by testing in simulation and on a UR10e robot manipulator. The objectives for this work are summarized in the three following research questions that will be investigated:

- How well does a task space trajectory tracking NMPC perform when using the full robot dynamics as constraints, in terms of accuracy and computation time?
- How can collision avoidance and singularity avoidance be included in the NMPC problem while maintaining real-time feasibility?
- Is it possible to use the developed methods to grasp moving objects and avoid moving obstacles in the environment?

1.3 Contribution

In order to answer these questions, a framework for task space trajectory tracking NMPC for robot manipulators was developed. By giving the kinematic and dynamic parameters of a robot manipulator of arbitrary DOF as input, using the Unified Robot Description Format (URDF) standard, a real-time NMPC solver is generated. More typical approaches either use a joint space formulation or simplifies the dynamics by linearization or only considering kinematic relationships, whereas here the full nonlinear dynamics are considered for torque control of the robot. Furthermore, the developed framework uses the Real-time Iteration (RTI) scheme for Sequential Quadratic Programming (SQP) to solve the problem in real-time. This work also considers how to include collision avoidance and singularity avoidance constraints in the NMPC problem, building on the previous works of Zube 2015, Krämer et al. 2020 and Lunni et al. 2017. Simulation and lab test results are presented for trajectory tracking and the singularity avoidance and collision avoidance extensions.

1.4 Outline

The report starts in Chapter 2 by presenting the necessary background theory. Kinematics, dynamics and manipulability for robot manipulators are reviewed, and a brief overview of optimal control and NMPC is given. Finally, the SQP method and the RTI scheme is presented. In Chapter 3, the specific trajectory tracking NMPC problem is proposed, and extensions for singularity avoidance and collision avoidance are considered. The software tools, as well as the UR10e robot used for testing, is also briefly presented. Chapter 4 presents results from testing the NMPC in simulation and on the UR10e robot, by considering only trajectory tracking, as well as tests with collision avoidance and singularity avoidance. A demo of the robot grasping a moving object is also considered. In Chapter 5 the presented results are discussed, and various issues, discrepancies and potential areas of further work are explored. Finally, Chapter 6 contains some concluding remarks for the work presented in the report.

Chapter 2

Theory

2.1 Kinematics and dynamics of robot manipulators

The foundation of control design, motion planning and simulation for robot manipulators starts with how we describe the motion of robots. This is done similarly as to all mechanical systems, by formulating the kinematics (motion as a purely geometric concept) and the dynamics (motion as a consequence of applied forces and torques). In this section, we will start by covering the necessary formalisms for representing the position and orientation of rigid bodies in space, before formulating the kinematics and dynamics of robot manipulators. Furthermore, the concept of manipulability will be presented.

2.1.1 Rigid body rotation representations

When expressing the state of a rigid body in space, it is uniquely defined by its position and orientation (pose), relative to some reference frame. We can trivially describe its position by the vector $\boldsymbol{p} \in \mathbb{R}^3$. How we describe the orientation of a rigid body in space is however not as trivial, and we will therefore in this section cover the different rotation representations used throughout this report. The reader is referred to Siciliano et al. 2010, Fossen 1999, Brekke 2020 and Solà 2017 for further details on rigid body rotations.

2.1.1.1 Rotation matrices

A possible starting point when describing the rotation of a rigid body in some reference frame is to look at how the axes of the reference frame are transformed under a rotation. Given the unit vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ and the rotated unit vectors $\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}'$, we can define the corresponding rotation of an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^3$ as:

$$\boldsymbol{v}' = \boldsymbol{R}\boldsymbol{v} = \begin{bmatrix} \boldsymbol{x}' & \boldsymbol{y}' & \boldsymbol{z}' \end{bmatrix} \boldsymbol{v}, \tag{2.1}$$

where $\boldsymbol{R} \in SO(3)$ is the rotation matrix, which has the properties that $\boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}_3$ and $\det(\boldsymbol{R}) = 1$. Note that a sequence of rotations is given by multiplication, e.g. $\boldsymbol{R}_0^2 = \boldsymbol{R}_0^1\boldsymbol{R}_1^2$ describes a rotation from frame 0 to frame 1, followed by a rotation from frame 1 to frame 2, which is equivalent to a rotation from frame 0 to frame 2 directly.

2.1.1.2 Angle-axis representation

An alternative way to describe rotation in $\mathbb{R}^3$ is to consider a rotation about an axis $\boldsymbol{u}$ by an angle θ. The resulting representation called the angle-axis representation, is related to the rotation matrix by

$$\boldsymbol{R}_{\boldsymbol{u},\theta} = \boldsymbol{I}_3 + \sin\theta[\boldsymbol{u}]_\times + (1 - \cos\theta)[\boldsymbol{u}]_\times^2. \tag{2.2}$$

For some applications, it may be useful to represent the rotation by the vector $\boldsymbol{v} = \theta\boldsymbol{u}$, termed the rotation vector. Note that this transforms the representation from 4 parameters to 3 parameters, which introduces a singularity in $\theta = 0$, for which the rotation is not defined. The angle-axis representation and rotation vector and its relation to the rotation matrix is visualized in Figure 2.1

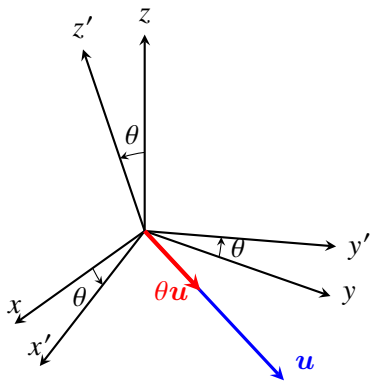


Figure 2.1: Relation between the rotation matrix defined by x', y' and z', the angle-axis representation by $\boldsymbol{u}$ and θ and the rotation vector $\theta\boldsymbol{u}$.

2.1.1.3 Euler angles

Another possible minimal representation is to consider three consecutive rotations, each around one of the coordinate axes. We can then define the Euler angles $\boldsymbol{\Theta} = [\ \varphi\ \ \theta\ \ \psi\ ]^\top$, for some combination of three rotations about X, Y and Z. In the following, the ZYX (roll-pitch-yaw) convention is used. Note that similarly to the rotation vector, this representation also has a singularity. For the ZYX convention, this happens for $\theta = \pm\pi/2$. For further details on rotation matrices, angle-axis and Euler angles the reader is referred to Fossen 1999 and Siciliano et al. 2010.

2.1.1.4 Unit quaternions

Quaternions are a generalization of complex numbers which are often used to represent rotations in 3-dimensional space. In the same way that a complex number on the unit circle can represent a rotation in $\mathbb{R}^2$, quaternions of unit length can represent a rotation in $\mathbb{R}^3$. In the section that follows, a brief overview of unit quaternions and their relation to rotational motion is given, based on Solà 2017 and Brekke 2020.

Quaternions consist of a single real element and three imaginary elements and can be written as the vector

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \boldsymbol{\varepsilon} \end{bmatrix}, \tag{2.3}$$

where $\eta \in \mathbb{R}$ and $\boldsymbol{\varepsilon} \in \mathbb{R}^3$ are the scalar and vector part of the quaternion respectively.

We define the multiplication of the two quaternions $\boldsymbol{q}_1 = [\ \eta_1 \quad \boldsymbol{\varepsilon}_1^\top\ ]^\top$ and $\boldsymbol{q}_2 = [\ \eta_2 \quad \boldsymbol{\varepsilon}_2^\top\ ]^\top$ as

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \begin{bmatrix} \eta_1\eta_2 - \boldsymbol{\varepsilon}_1^\top\boldsymbol{\varepsilon}_2 \\ \eta_2\boldsymbol{\varepsilon}_1 + \eta_1\boldsymbol{\varepsilon}_2 + [\boldsymbol{\varepsilon}_1]_\times\boldsymbol{\varepsilon}_2 \end{bmatrix}. \tag{2.4}$$

The norm of a quaternion is $\|\boldsymbol{q}\| = \sqrt{\eta^2 + \boldsymbol{\varepsilon}^\top\boldsymbol{\varepsilon}}$, and the inverse of a quaternion is defined as

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|^2}, \tag{2.5}$$

where

$$\boldsymbol{q}^* = \begin{bmatrix} \eta \\ -\boldsymbol{\varepsilon} \end{bmatrix} \tag{2.6}$$

is the quaternion conjugate.

A unit quaternion has unit norm $\|\boldsymbol{q}\| = 1$, and is as previously mentioned one of the standard representations of rigid body rotations. Going back to the axis-angle representation, given a unit vector $\boldsymbol{u}$ and angle θ, the unit quaternion is given by

$$\boldsymbol{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \boldsymbol{u}\sin\frac{\theta}{2} \end{bmatrix}, \tag{2.7}$$

analogous to Euler's formula for two-dimensional rotation using complex numbers. In the quaternion formalism, the rotation of a vector $\boldsymbol{x}$ by the θ about the axis $\boldsymbol{u}$ is given by

$$\boldsymbol{x}' = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \otimes \boldsymbol{q}^*. \tag{2.8}$$

Furthermore, the relation between the rotation matrix and the quaternion is as given in Solà 2017

$$\boldsymbol{R} = (\eta - \varepsilon^\top \varepsilon) I + 2\varepsilon\varepsilon^\top + 2\eta[\varepsilon_1]_\times. \tag{2.9}$$

Other conversion formulas between the different formalisms relevant to this work are given in Appendix A.

Note that the unit quaternion representation is not unique, as $\boldsymbol{q}$ and $-\boldsymbol{q}$ represents the same rotation. While the representation is not unique, it is however singularity-free.

Also note that a composition of several rotations, similarly to rotation matrices, is given as the product of these unit quaternions, i.e. $\boldsymbol{q}_{AC} = \boldsymbol{q}_{AB} \otimes \boldsymbol{q}_{BC}$. This lets us define the unit quaternion error as

$$\delta\boldsymbol{q} = \boldsymbol{q}_1^* \otimes \boldsymbol{q}_2, \tag{2.10}$$

as this achieves $\boldsymbol{q}_2 = \boldsymbol{q}_1 \otimes \delta\boldsymbol{q}$.

Finally, we will consider the unit quaternion exponential, logarithm and power, as presented in Solà 2017. The exponential of a unit quaternion is defined as

$$e^{\boldsymbol{q}} = \sum_{k=0}^{\infty} \frac{\boldsymbol{q}^k}{k!} = e^\eta \begin{bmatrix} \cos\|\varepsilon\| \\ \frac{\varepsilon}{\|\varepsilon\|}\sin\|\varepsilon\| \end{bmatrix}, \tag{2.11}$$

and the inverse operation, the unit quaternion logarithm, is defined as

$$\log \boldsymbol{q} = \begin{bmatrix} 0 \\ \theta\boldsymbol{u} \end{bmatrix}, \tag{2.12}$$

where $\boldsymbol{u}$ and θ are defined as in Section 2.1.1.2. Note that the imaginary part of the quaternion logarithm is the rotation vector, such that the logarithm defines a map from quaternions to rotation vectors. Furthermore, the conversion formulas between rotation vectors and unit quaternions are given in Eq. (A.3) and Eq. (A.4). Finally, using these definitions we find the unit quaternion power as

$$\boldsymbol{q}^s = \exp(s \log \boldsymbol{q}) = \begin{bmatrix} \cos s\theta \\ \boldsymbol{u}\sin s\theta \end{bmatrix}, \tag{2.13}$$

which creates the basis for linear interpolation between quaternions. This will be discussed further in Section 3.2.2.

2.1.2 Manipulator kinematics and the Denavit-Hartenberg convention

A robot manipulator is a mechanical system, consisting of a chain of rigid body links, connected by joints. One end of the chain is connected to a static base, while the other is

connected to the end effector, which is a general term for the tool at the end of the arm that interacts with the environment. Analyzing the robot manipulator system we can define the states of the system $\boldsymbol{x} \in \mathbb{R}^{2n}$ as the joint angles $\boldsymbol{q} \in \mathbb{R}^n$ and the joint angular velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, i.e.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}. \tag{2.14}$$

We denote the space spanned by $\boldsymbol{q}$ as the joint space.

However, when interacting with the environment we usually express the tasks for the robot in terms of a desired end effector pose $\boldsymbol{h}(t)$, not a desired joint configuration. The end effector pose space is therefore fittingly called the task space. This motivates the introduction of some mathematical framework for expressing the pose of the end effector using the joint configuration of the manipulator $\boldsymbol{q}$, which is called the forward kinematics. In the following, we will formulate the forward kinematics equations for robot manipulators, using homogeneous transformation matrices.

2.1.2.1 Homogeneous transformation matrices

Motivated by the goal of finding a mapping from the joint space to the task space, we note how the rigid body transformation from the inertial frame to the end effector frame is a series of rigid body transformations along the links of the manipulator. Homogeneous coordinates and homogeneous transformations are convenient mathematical tools to represent this chain of rigid body transformations, and will be introduced in the following. As discussed in Siciliano et al. 2010, we first extend the representation of a point $\boldsymbol{p} \in \mathbb{R}^3$ by appending a unit element, i.e.

$$\tilde{\boldsymbol{p}} = \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix}, \tag{2.15}$$

which is termed homogeneous coordinates. This lets us define the homogeneous transformation matrix as

$$\boldsymbol{T}_1^0 = \begin{bmatrix} \boldsymbol{R}_1^0 & \boldsymbol{r}_1^0 \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in SE(3), \tag{2.16}$$

which transforms a homogeneous coordinate vector $\tilde{\boldsymbol{p}}$ from frame 1 to frame 0 defined by the translation vector $\boldsymbol{r}_1^0 \in \mathbb{R}^3$ and rotation vector $\boldsymbol{R}_1^0 \in SO(3)$. This transformation is then simply written as

$$\tilde{\boldsymbol{p}}^0 = \boldsymbol{T}_1^0 \, \tilde{\boldsymbol{p}}^1. \tag{2.17}$$

Analogous to rotation matrices, a sequence of homogeneous transformations is given by

$$\tilde{\boldsymbol{p}}^0 = \boldsymbol{T}_1^0 \, \boldsymbol{T}_2^1 \, \cdots \, \boldsymbol{T}_n^{n-1} \, \tilde{\boldsymbol{p}}^n. \tag{2.18}$$

It is then straightforward to formulate the forward kinematics of the end effector using a series of homogeneous transformations. We define a fixed coordinate frame in every link, and starting in the end effector frame we consider the sequence of homogeneous transformations from every link to its parent, ending up in the base frame. The homogeneous transformation matrix describing the pose of the end effector with respect to the base frame is then given as:

$$\boldsymbol{T}_e^b(\boldsymbol{q}) = \boldsymbol{T}_0^b \, \boldsymbol{T}_1^0(q_1) \, \boldsymbol{T}_2^1(q_2) \, \cdots \, \boldsymbol{T}_n^{n-1}(q_n) \, \boldsymbol{T}_e^n, \tag{2.19}$$

where $\boldsymbol{T}_0^b$ and $\boldsymbol{T}_e^n$ are static transformations between the base frame and the first link, and the last link and the end effector frame respectively.

2.1.2.2 The Denavit-Hartenberg convention

We have seen how to relate the pose of the end effector to the base frame using homogeneous transformations. However, we have yet to see how we can formulate these homogeneous transformation matrices, and therefore also the end effector pose, as a function of the joint angles. This task comes down to how to define the link frames and the transformations between them. The Denavit-Hartenberg (DH) convention formulates a systematic way of defining these transformations for any robot manipulator.

The transform from frame i to frame $i+1$ following the DH convention is defined as

$$\begin{aligned} \boldsymbol{T}_{i+1}^i &= \begin{bmatrix} \boldsymbol{R}_z(\theta_i) & d_i \boldsymbol{e}_z \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_x(\alpha_i) & a_i \boldsymbol{e}_x \\ \boldsymbol{0}^T & 1 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{R}_z(\theta_i)\,\boldsymbol{R}_x(\alpha_i) & a_i \boldsymbol{R}_z(\theta_i)\,\boldsymbol{e}_x + d_i \boldsymbol{e}_z \\ \boldsymbol{0}^T & 1 \end{bmatrix}, \end{aligned} \tag{2.20}$$

where $\boldsymbol{R}_x$ and $\boldsymbol{R}_z$ denote simple rotations around the x-axis and z-axis, and $\boldsymbol{e}_x = [\,1\,0\,0\,]^\top$ and $\boldsymbol{e}_z = [\,0\,0\,1\,]^\top$. The transformation consists of a translation by d_i and rotation by θ_i about the z-axis, followed by a translation by a_i and a rotation by α_i about the x-axis of the intermediate frame. Given the DH parameters for a robot manipulator, we are then able to determinate the forward kinematics of the end effector as a simple sequence of linear transformations. The reader is referred to Siciliano et al. 2010 for further details regarding homogeneous transformation matrices and the Denavit-Hartenberg convention.

2.1.3 Differential kinematics and manipulability

An important consideration for robot manipulator motion planning and control is how to handle and avoid singular configurations. The concept of manipulability, as first introduced in Yoshikawa 1985, provides a convenient tool for numerically analyzing how close a robot

configuration is to a singular configuration.

First, consider the differential kinematics of a robot manipulator. As discussed in Siciliano et al. 2010, the end effector velocity $\dot{\boldsymbol{h}} \in \mathbb{R}^m$ is related to the joint angle velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ by the linear map given by the Jacobian $\boldsymbol{J}(\boldsymbol{q})$:

$$\dot{\boldsymbol{h}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.21}$$

Furthermore, for the joint angles $\tilde{\boldsymbol{q}}$ for which the Jacobian is rank-deficient, i.e.

$$\text{rank } \boldsymbol{J}(\tilde{\boldsymbol{q}}) < m, \tag{2.22}$$

the configuration of the manipulator is singular. This means the motion of the end effector is limited to some subspace of the full space $\mathbb{R}^m$. Entering such a configuration (or rather entering the vicinity of the singular configuration) is not desirable, as a small end effector velocity could map to large joint angle velocities, which in turn could both damage the joint motors and produce motions that are not safe, e.g. for humans that collaborate with the robot. This motivates constructing some function that measures closeness to singular configurations, which is termed manipulability.

In Yoshikawa 1985 the manipulability index

$$m(\boldsymbol{q}) \equiv \sqrt{\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\top)} = \sigma_1 \, \sigma_2 \, \cdots \, \sigma_m \tag{2.23}$$

is introduced, with $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_m \geq 0$ being the singular values of $\boldsymbol{J}$. The manipulability index has the property that it approaches 0 as the robot configuration approaches a singularity, and increases as the manipulability of the robot increases. It therefore provides a useful tool for measuring the manipulability of the robot.

2.1.4 Manipulator dynamics

To simulate or design a control system for a robot manipulator it is evidently useful to not only study the kinematics of the manipulator, but also the dynamics. As introduced in Section 2.1.2, the states of the system are given by the joint angles $\boldsymbol{q}$ and the joint velocities $\dot{\boldsymbol{q}}$. The control inputs are the joint motor torques $\boldsymbol{\tau}$. It is worth noting here that while the low-level input is the motor torque, many robotic manipulator arms provide joint angle and joint velocity control interfaces.

The general manipulator equations of motion can then be derived as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} + \boldsymbol{F}_s \, \text{sgn}(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{2.24}$$

assuming the manipulator end effector exerts no forces or torques on the environment. Here

$\boldsymbol{M}$ is the square, symmetric and positive-definite inertia matrix, $\boldsymbol{C}$ is the Coriolis and centripetal force matrix, $\boldsymbol{g}$ is the gravity term, and $\boldsymbol{F}_v$ and $\boldsymbol{F}_s$ parametrize the viscous and static (Coulomb) friction torques respectively. The equations of motion can be derived e.g. by Lagrange's equation or using a Newton-Euler based approach such as the recursive Newton-Euler algorithm (see Siciliano et al. 2010).

By rewriting the equations of motion in terms of the state vector $\boldsymbol{x} = [\ \boldsymbol{q}^\top\ \dot{\boldsymbol{q}}^\top\ ]^\top = [\ \boldsymbol{x}_1^\top\ \boldsymbol{x}_2^\top\ ]^\top$ and assuming no friction terms we get the simplified state dynamics

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\tau}) = \begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{M}(\boldsymbol{x}_1)^{-1}(\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{x}_1, \boldsymbol{x}_2) - \boldsymbol{g}(\boldsymbol{x}_1)) \end{bmatrix}. \tag{2.25}$$

We denote this formulation as the inverse dynamics, as opposed to the forward dynamics in Eq. (2.24). It can be derived by explicitly calculating the inverse of the inertia matrix, or using a recursive inverse dynamics method such as the articulated body algorithm or the composite rigid body algorithm (see Featherstone 2008).

2.2 Optimal control

Optimal control theory covers how to find optimal control policies for dynamical systems with respect to some cost function. This approach to motion planning and control provides many benefits which are relevant for autonomous robots, one of which is how constraints in the system, such as bounds on the robot joints, can be considered explicitly when finding the control policy. The branch of numerical optimal control within optimal control theory is concerned with how to apply numerical methods to approximately solve such problems. This is especially of interest, as it has potential for solving complex optimal control problems in real-time. Numerical optimal control will be briefly presented in the following section before we will discuss how to close the loop with NMPC. Then the real-time numerical solver used in this work, SQP using the RTI scheme, will be presented. The section is based on Lars and Jürgen 2011, Johansen 2011, Diehl et al. 2009, Nocedal and Wright 2006, Egeland and Gravdahl 2002 and Gros et al. 2016.

2.2.1 Numerical optimal control

This section will introduce the general optimal control problem (OCP) and then look at various discretization methods in the literature for making this problem computationally

tractable. The continuous-time infinite horizon nonlinear optimal control problem is given as:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \int_0^\infty \ell(\boldsymbol{x},\boldsymbol{u})\,dt \\ \text{s.t.} \quad & \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}), \\ & \boldsymbol{x}(t) \in \mathcal{X}, \quad \forall t, \\ & \boldsymbol{u}(t) \in \mathcal{U} \quad \forall t, \\ & \boldsymbol{x}(0) = \boldsymbol{x}_0, \end{aligned} \tag{2.26}$$

where $\ell(\boldsymbol{x},\boldsymbol{u})$ is the cost function, $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u})$ are the system dynamics, $\mathcal{X}$ and $\mathcal{U}$ are some in general non-convex state and input constraint sets, and $\boldsymbol{x}_0 \in \mathbb{R}^n$ is the initial condition. For the case of robot manipulators, the state and input constraint sets typically include joint angle, joint velocity and motor torque constraints, but other nonlinear constraints will also be considered.

It is desirable to solve Eq. (2.26) analytically and apply the optimal control input $\boldsymbol{u}^*$ to the system. However, finding the exact solution is rarely computationally tractable for an nonlinear program (NLP). Discretization is therefore necessary, and here a distinction is made between indirect and direct methods. Indirect methods deal with how to solve the continuous OCP and then discretize the solution, while direct methods first discretize the problem and then solve it using numerical optimization. In the following, we will explore direct methods further, as they typically provide more real-time capable solvers for NLPs.

First, we constrain ourself to the finite horizon problem, i.e. by only considering optimization over a finite time horizon T. Furthermore, we introduce the concept of sampling $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ with a constant sample time t_s. Finally, the integral of the cost function and the dynamics are discretized with this sample time, such that we end up with the following NLP:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{i=0}^{N-1} \ell(\boldsymbol{x}_i,\boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_N) \\ \text{s.t.} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i,\boldsymbol{u}_i), \quad i = 0,\dots,N-1, \\ & \boldsymbol{g}_i(\boldsymbol{x}_i,\boldsymbol{u}_i) \le 0, \quad i = 0,\dots,N-1, \\ & \boldsymbol{h}_i(\boldsymbol{x}_i,\boldsymbol{u}_i) = 0, \quad i = 0,\dots,N-1, \\ & \boldsymbol{g}_N(\boldsymbol{x}_N) \le 0, \\ & \boldsymbol{h}_N(\boldsymbol{x}_N) = 0, \\ & \boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0, \end{aligned} \tag{2.27}$$

where $\boldsymbol{x} = [\ \boldsymbol{x}_0^\top\ \boldsymbol{x}_1^\top\ \cdots\ \boldsymbol{x}_N^\top\ ]^\top$ is a vector of the states at the sample times and similarly for $\boldsymbol{u} = [\ \boldsymbol{u}_0^\top\ \boldsymbol{u}_1^\top\ \cdots\ \boldsymbol{u}_{N-1}^\top\ ]^\top$. Furthermore, we introduce the final stage cost $\ell_f(\boldsymbol{x}_N)$ and rewrite the state and input constraints more explicitly as a set of nonlinear inequality and

equality constraints using $\boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i)$ and $\boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i)$.

2.2.1.1 Numerical optimal control methods

In the following, a brief overview of the most relevant direct optimal control methods will be presented, based on Johansen 2011. Starting with direct single shooting, the basic idea is to eliminate the state variable $\boldsymbol{x}$ by recursively substituting the discretized system dynamics $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i)$ for $\boldsymbol{x}_{i+1}$, such that we get a reduced problem where the cost and constraints are only a function of the input variable $\boldsymbol{u}$ and the initial condition $\bar{\boldsymbol{x}}_0$.

This implies the use of some numerical integration scheme for the cost and dynamics. The integral of the cost function is typically evaluated using some simple quadrature rule, such as the rectangle rule or the trapezoidal rule. The dynamics can be discretized using a numerical integration scheme such as explicit Runge-Kutta (ERK) or implicit Runge-Kutta (IRK) methods. One typical example is the 4th order explicit Runge-Kutta (ERK4) method, which is given by

$$\begin{aligned}
\boldsymbol{y}_{k+1} &= \boldsymbol{y}_k + \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4), \quad \text{where} \\
\boldsymbol{k}_1 &= h\boldsymbol{f}(t_k, \boldsymbol{y}_k), \\
\boldsymbol{k}_2 &= h\boldsymbol{f}\left(t_k + \frac{h}{2}, \boldsymbol{y}_k + \frac{\boldsymbol{k}_1}{2}\right), \\
\boldsymbol{k}_3 &= h\boldsymbol{f}\left(t_k + \frac{h}{2}, \boldsymbol{y}_k + \frac{\boldsymbol{k}_2}{2}\right), \\
\boldsymbol{k}_4 &= h\boldsymbol{f}(t_k + h, \boldsymbol{y}_k + \boldsymbol{k}_3),
\end{aligned} \tag{2.28}$$

and will be frequently used in later sections. The reader is referred to Egeland and Gravdahl 2002 for further details on these numerical integration schemes.

The direct multiple shooting method is a generalization of direct shooting, where the time horizon $[0,\ T]$ is segmented into $M + 1$ intervals. For every interval $[t_i,\ t_{i+1}]$ the direct shooting method is applied, and an additional constraint is added to ensure continuity between every interval. Finally, for direct collocation the general idea is to approximate the state and input as piecewise polynomials, where the trajectory is then parametrized by a number of knot points between every sample.

2.2.2 Nonlinear Model Predictive Control

After formulating the discrete-time finite horizon OCP in Eq. (2.27) we are able to formulate the NMPC algorithm, as given in Algorithm 1. By solving the OCP at every time step with the current sampled state, and applying the first control input from the solution, we effectively close the loop. This introduces robustness against modeling errors and noise when comparing

to applying the optimal control sequence $\boldsymbol{u}^*$ directly in open loop. Yet this comes at the cost of high computational demands. Theoretical details on NMPC, such as stability, feasibility and robustness, will not be discussed here. See Lars and Jürgen 2011 and Johansen 2011 for further details on this.

Algorithm 1 NMPC

1: **for** For every time step t_k:
2: Sample the current state $\boldsymbol{x}(t_k)$ of the system.
3: Solve Eq. (2.27) using $\bar{\boldsymbol{x}}_0 = \boldsymbol{x}(t_k)$, and denote the solution as $\boldsymbol{x}^*, \boldsymbol{u}^*$.
4: Let $\boldsymbol{\mu}(t_k) = \boldsymbol{u}_0^*$ be the control input at the current time step.

2.2.3 Sequential Quadratic Programming

In order to solve the NMPC problem a nonlinear optimization solver is required, which should be able to solve a general NLP at every time step:

$$\min_{\boldsymbol{z}} \boldsymbol{f}(\boldsymbol{z}) \quad \text{s.t.} \quad \boldsymbol{g}(\boldsymbol{z}) \leq 0,\ \boldsymbol{h}(\boldsymbol{z}) = 0. \tag{2.29}$$

In the following, the SQP method is presented, yet the reader should be aware that many other numerical methods exist, of which interior point solvers such as IPOPT (Wächter and Biegler 2006) are also frequently used for numerical optimal control. The reader is referred to Diehl et al. 2009 for further details on interior point solvers.

In order to formulate the principle behind SQP, we start by stating the first-order optimality conditions for Eq. (2.29), known as the Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} \nabla_{\boldsymbol{z}} \mathcal{L}\left(\boldsymbol{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*\right) &= 0, \\ \boldsymbol{g}\left(\boldsymbol{z}^*\right) &\leq 0, \\ \boldsymbol{h}\left(\boldsymbol{z}^*\right) &= 0, \\ \boldsymbol{\mu}^* &\geq 0, \\ g_i\left(\boldsymbol{z}^*\right) \mu_i^* &= 0, \quad i = 1, \ldots, n_g, \end{aligned} \tag{2.30}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ are the Lagrange multipliers corresponding to the inequality constraints $\boldsymbol{g}$ and the equality constraints $\boldsymbol{h}$ respectively, n_g is the number of inequality constraints and $\mathcal{L}$ is the Lagrangian function defined as

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \boldsymbol{f}(\boldsymbol{z}) + \boldsymbol{\lambda}^\top \boldsymbol{h}\left(\boldsymbol{z}\right) + \boldsymbol{\mu}^\top \boldsymbol{g}\left(\boldsymbol{z}\right). \tag{2.31}$$

The SQP method is based on solving the KKT system Eq. (2.30) using Newton's method for nonlinear systems, i.e. iteratively solving its linearized system of equations. It can be

shown that this is equivalent to iteratively solving the following quadratic program (QP):

$$\begin{aligned} \min_{z} \quad & \nabla_z \boldsymbol{f}^\top(z_k)(z - z_k) + \frac{1}{2}(z - z_k)^\top \nabla_z^2 L(z_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)(z - z_k) \\ \text{s.t.} \quad & \boldsymbol{h}(z_k) + \nabla_z \boldsymbol{h}^\top(z_k)(z - z_k) = 0, \\ & \boldsymbol{g}(z_k) + \nabla_z \boldsymbol{g}^\top(z_k)(z - z_k) \leq 0, \end{aligned} \tag{2.32}$$

where z_k is the current iterate of the approximation of the solution z^*. We call one such iteration a Newton step, as it corresponds to a single step of Newton's method.

The Hessian $\nabla_z^2 L(z_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)$ typically requires high computational effort to compute exactly, and it must be positive definite for the problem Eq. (2.32) to be convex, which is not guaranteed for z far away from the optimum. The Gauss-Newton method is one possible solution to these problems, where the Hessian of $\boldsymbol{f}$ is approximated by

$$\boldsymbol{H} = \nabla \boldsymbol{r}(\boldsymbol{x}) \nabla \boldsymbol{r}(\boldsymbol{x})^\top, \tag{2.33}$$

given that the cost is of a sum of squares form, i.e. $\boldsymbol{f}(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2$. Further details on Gauss-Newton, as well as other implementation aspects of SQP, such as globalization strategies, are not considered here. The reader is referred to Nocedal and Wright 2006 for further details on these topics. One aspect of SQP implementations which is of vital importance to NMPC specifically are warm starts, and this will however be considered in further detail in the following section.

2.2.4 SQP Real-time Iteration scheme

For NMPC problems the solution at a certain time step is usually very similar to the solution at the previous step. This motivates developing methods for warm starting the optimization solver using the previous solution. One such method, the RTI scheme, will be briefly presented in the following section. The section is based on Gros et al. 2016.

The RTI scheme makes use of the fact that shifting the previous SQP solution by one time step produces an initial guess that is already very close to the solution at the current time step. Furthermore, when doing this shifting procedure to warm start the solver, we can also assume that performing a single Newton step of the SQP algorithm provides a reasonable approximation of the converged SQP solution. The combination of warm starts by shifting and only doing a single Newton step is the essence of the RTI scheme, and can in some sense be regarded as running normal SQP while updating the information the solver has about the current state of the system at every iteration. Also note that the Gauss-Newton Hessian approximation given in Eq. (2.33) is usually used in SQP RTI. One additional trick is however done before arriving at the final algorithm.

A problem with the approach so far, which moreover is a general problem for all real-time

Model Predictive Control (MPC) solvers, is that there is a significant delay between when the system state $\boldsymbol{x}(t_k)$ is sampled, and when the solver is finished and the first control input of the solution $\boldsymbol{u}_0^*$ can be applied to the system. This feedback delay results in the solver using outdated information of the system state and can reduce the stability margins of the closed-loop system. Gros et al. 2016 refer to this issue as the real-time dilemma.

The consequences of the real-time dilemma is mitigated in RTI by splitting the algorithm into two phases: the preparation phase and the feedback phase. In the preparation phase the shifting and linearization steps are executed and the QP is partially formed, and in the feedback phase the QP is fully formed and solved. Since all the computations done in the preparation phase are completely independent of the initial condition $\bar{\boldsymbol{x}}_0$, the system state $\boldsymbol{x}(t_k)$ only needs to be sampled after performing the preparation step. The basic loop of the preparation and feedback phase is shown in Figure 2.2, which illustrates how the RTI scheme minimizes the feedback delay. Since in practice the preparation phase typically takes longer to do than the feedback phase (Diehl et al. 2009), the feedback delay can be drastically reduced. We then arrive at the basic SQP RTI algorithm which will be used in the following sections, as given in Algorithm 2.

Figure 2.2: Preparation phase and feedback phase loop of the RTI scheme.

Algorithm 2 SQP RTI

Preparation phase:

1: Shift previous solution $\boldsymbol{x}_{k-1}^*, \boldsymbol{u}_{k-1}^*$ to construct initial guess $\boldsymbol{x}_k^0, \boldsymbol{u}_k^0$.

2: Evaluate the terms in the QP Eq. (2.32), and form the QP omitting $\bar{\boldsymbol{x}}_0$.

Feedback phase:

3: Input $\bar{\boldsymbol{x}}_0 = \boldsymbol{x}(t_k)$ into the QP and solve it to get $\Delta\boldsymbol{x}_k^*, \Delta\boldsymbol{u}_k^*$.

4: Apply the full Newton step to get the NMPC solution at the current time step:

$$(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*) \leftarrow (\boldsymbol{x}_k^0, \boldsymbol{u}_k^0) + (\Delta\boldsymbol{x}_k^*, \Delta\boldsymbol{u}_k^*).$$

Chapter 3

Design and implementation

3.1 Robot manipulator trajectory tracking NMPC

Given some desired end effector pose trajectory, we want to apply the NMPC approach to solve the trajectory tracking problem for a general robot manipulator arm. This standard problem is solved by considering a cost function that penalizes tracking error, typically the quadratic Euclidean distance cost function

$$\ell_{TT}(\boldsymbol{x}, \boldsymbol{u}) = \|\boldsymbol{x} - \boldsymbol{x}_d(t)\|^2 + \|\boldsymbol{u} - \boldsymbol{u}_d(t)\|^2, \tag{3.1}$$

where $\boldsymbol{x}_d(t)$ and $\boldsymbol{u}_d(t)$ are the time-varying desired state and input trajectories. In this section, we will start by considering how to formulate the tracking error for the end effector pose, and then discuss the various formulations for the trajectory tracking NMPC problem for robot manipulators, before arriving at the specific formulation that will be implemented and tested.

Since we will consider following an end effector pose trajectory, i.e. a task space formulation of the trajectory tracking problem, certain changes have to be made to the standard cost function in Eq. (3.1). First let $\boldsymbol{h}_d(t) \equiv [\ \boldsymbol{h}_{d,p}(t)^\top\ \boldsymbol{h}_{d,o}(t)^\top\ ]^\top \in \mathbb{R}^7$ denote the desired pose trajectory consisting of the position $\boldsymbol{h}_{d,p}(t) \in \mathbb{R}^3$ and unit quaternion $\boldsymbol{h}_{d,o}(t) \in \mathbb{R}^4$. By using the forward kinematics we can formulate the pose of the end effector, and thus find the pose tracking error.

The end effector position $\boldsymbol{h}_p(\boldsymbol{q})$ is derived from the last column of the homogeneous transformation matrix from the robot frame to the end effector frame $\boldsymbol{T}_e^b(\boldsymbol{q})$ given in Eq. (2.19):

$$\begin{bmatrix} \boldsymbol{h}_p(\boldsymbol{q}) \\ 1 \end{bmatrix} = \boldsymbol{T}_e^b(\boldsymbol{q}) \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^\top. \tag{3.2}$$

The end effector orientation is given by the rotation quaternion $\boldsymbol{h}_o(\boldsymbol{q})$, which is found from converting the rotation matrix $\boldsymbol{R}_e^b$ to a unit quaternion according to Eq. (A.11), or alternatively

by performing a similar chain of frame transformations as in Eq. (2.19) with a unit quaternion in conjunction.

With the end effector pose given by the kinematics, we can then formulate the pose tracking error. Starting with the position error we have

$$\boldsymbol{e}_p(\boldsymbol{q}) \equiv \boldsymbol{h}_{d,p}(t) - \boldsymbol{h}_p(\boldsymbol{q}). \tag{3.3}$$

For the orientation error the choice is not quite as trivial, and many possible formulations exist, with different properties. One possibility is to consider the quaternion error

$$\delta\boldsymbol{q} = \boldsymbol{q}_d(t)^* \otimes \boldsymbol{q}_a, \tag{3.4}$$

where $\boldsymbol{q}_d(t)$ and $\boldsymbol{q}_a$ are the desired and actual quaternions respectively. We can then find an error function based on making the quaternion error approach the identity unit quaternion $[\ \pm 1\ 0\ 0\ 0\ ]^\top$. As discussed in Siciliano et al. 2010, we can define the orientation tracking error as

$$\boldsymbol{e}_o(\boldsymbol{q}_d, \boldsymbol{q}_a) \equiv \eta_d\boldsymbol{\varepsilon}_a - \eta_a\boldsymbol{\varepsilon}_d - [\boldsymbol{\varepsilon}_d]_\times\boldsymbol{\varepsilon}_a, \tag{3.5}$$

i.e. the vector part of the error quaternion. From the unit constraint it is seen that $\boldsymbol{e}_o(\boldsymbol{q}_d, \boldsymbol{q}_a) = \boldsymbol{0}$ will then uniquely track the desired trajectory exactly. The reader is referred to Jackson et al. 2021 for a brief discussion of other possible formulations of orientation error, including the exponential map, Rodrigues parameters and Modified Rogrigues parameters. Other possible error metrics based on quaternions and rotation matrices are presented in Huynh 2009, of which the geodesic distance between the quaternions

$$\boldsymbol{e}_{o,\text{geo}}(\boldsymbol{q}_d, \boldsymbol{q}_a) \equiv \| \log \delta\boldsymbol{q} \| \tag{3.6}$$

also seems promising. However in the following, the vector part of the quaternion error in Eq. (3.5) will be used as it is easier to compute. Therefore the orientation error in the NMPC problem is given by

$$\boldsymbol{e}_o(\boldsymbol{q}) \equiv \eta_d(t)\boldsymbol{\varepsilon}(\boldsymbol{q}) - \eta(\boldsymbol{q})\boldsymbol{\varepsilon}_d(t) - [\boldsymbol{\varepsilon}_d(t)]_\times\boldsymbol{\varepsilon}(\boldsymbol{q}). \tag{3.7}$$

From here on we denote $\boldsymbol{e}(\boldsymbol{q}) \equiv [\ \boldsymbol{e}_p(\boldsymbol{q})^\top\ \boldsymbol{e}_o(\boldsymbol{q})^\top\ ]^\top$ as the pose tracking error using this orientation error formulation. We can then formulate the weighted quadratic tracking error cost as $\|\boldsymbol{e}(\boldsymbol{x})\|^2_{\boldsymbol{Q}}$.

In addition to penalizing the tracking error over the horizon, various other terms may be considered. One may penalize the derivate of the tracking error $\dot{\boldsymbol{e}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ to generate smoother motion. Other cost terms that tries to accomplish additional tasks may also be considered, such as singularity avoidance and collision avoidance, which will be discussed further in

Section 3.3 and Section 3.4.

Regularization terms are also necessary to construct a well-behaved optimization problem with unique minima. The most evident regularization term is to penalize the input, e.g. using the weighted quadratic cost $\|\boldsymbol{u}\|^2_{\boldsymbol{R}}$. For torque control of a robot manipulator this is evidently useful as it reduces energy usage. Adding a regularization term in terms of the derivative of the torque input $\dot{\boldsymbol{u}}$ is also of interest, as it will help avoid oscillatory and high jerk behavior. The reader is referred to Nie and Kerrigan 2018 for how this may be implemented in practice.

Analogous to these two terms, but for the joint space, one may also consider minimizing joint acceleration $\ddot{\boldsymbol{q}}$ or joint jerk $\dddot{\boldsymbol{q}}$ (see Zhao et al. 2018), to force the solution to be smoother in joint space. It is worth noting that costs on the state variables $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ directly may also be of relevance. The former could help mitigate self-collisions, by forcing the motion of the robot to stay within some vicinity of a "default" joint configuration. The latter could also help in generating smooth motion in the joint space, like acceleration and jerk costs. For the basic trajectory tracking NMPC we will consider quadratic cost terms on tracking error $\boldsymbol{e}(\boldsymbol{q})$, input $\boldsymbol{u}$ and joint acceleration $\ddot{\boldsymbol{q}}$. The specific NMPC problem can therefore be formulated as:

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{i=0}^{N-1} \left(\|\boldsymbol{e}(\boldsymbol{x}_i)\|^2_{\boldsymbol{Q}} + \|\ddot{\boldsymbol{q}}_i\|^2_{\boldsymbol{R}_a} + \|\boldsymbol{u}_i\|^2_{\boldsymbol{R}_u} \right) + \|\boldsymbol{e}(\boldsymbol{x}_N)\|^2_{\boldsymbol{S}} \\
\text{s.t.} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \quad i = 0, \ldots, N-1, \\
& \boldsymbol{x}_{\min} \leq \boldsymbol{x}_i \leq \boldsymbol{x}_{\max}, \quad i = 0, \ldots, N, \\
& \boldsymbol{u}_{\min} \leq \boldsymbol{u}_i \leq \boldsymbol{u}_{\max}, \quad i = 0, \ldots, N-1, \\
& \ddot{\boldsymbol{q}}_{\min} \leq \ddot{\boldsymbol{q}}_i \leq \ddot{\boldsymbol{q}}_{\max}, \quad i = 0, \ldots, N-1, \\
& \boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0,
\end{aligned}
\tag{3.8}
$$

where $\boldsymbol{u} = \boldsymbol{\tau}$ is the joint motor torque inputs and $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ are the discretized dynamics, found by discretizing Eq. (2.25), e.g. by ERK4. The joint acceleration $\ddot{\boldsymbol{q}}$ is given by the second term of the dynamics. We have inequality constraint on $\boldsymbol{x}$ and $\boldsymbol{u}$ in order to ensure that joint angle, joint velocity and joint torque limits are satisfied. Furthermore, constraints on $\ddot{\boldsymbol{q}}$ are added to ensure that the generated motion has sufficiently low acceleration.

Other formulations exist, such as considering velocity control of the robot manipulator directly, where the basic integrator dynamics $\dot{\boldsymbol{q}} = \boldsymbol{u}$ are used instead. This formulation is used in Arbo et al. 2017 and Zube 2015, and is for most applications perfectly sufficient, especially for static base systems. Furthermore, replacing the highly nonlinear dynamics constraints by linear constraints evidently makes the problem simpler and thus faster to solve. However, a torque input formulation of the NMPC problem with full nonlinear dynamics is interesting, since it allows for torque minimization and more dynamic motion. The benefits and limitations of this formulation will be further discussed in Chapter 5.

Finally, notice the terminal cost on the tracking error in Eq. (3.8), weighted on the

terminal cost matrix $\boldsymbol{S}$. This term may be used to approximate the remaining cost from N to infinity in order to approximate the infinite horizon optimal control problem, as discussed in Lars and Jürgen 2011. Designing terminal costs appropriately, as well as terminal constraints, may have stabilizing effects. Several formulations of terminal costs and constraints were experimented with during initial tests of the NMPC, but yielded no noticeable improvements in performance or stability, and is therefore not considered in detail from here on. It is however a potential area of further research for this work.

3.1.1 Slack variables

Finally, we will consider extending the trajectory tracking NMPC problem Eq. (3.8) to use slack variables were applicable. The main concept behind slack variables is that certain (hard) inequality constraints can be extended to be soft by adding slack variables s, meaning that the constraints may be violated but at a high cost when violated. Consider the slack variable extension of the general inequality constraint:

$$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \leq 0 \quad \rightarrow \quad \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) - \boldsymbol{s} \leq 0, \quad \boldsymbol{s} \geq 0, \tag{3.9}$$

By adding a linear and quadratic cost on the slack variable, i.e.

$$\ell_s(\boldsymbol{s}) = \boldsymbol{z}^\top \boldsymbol{s} + \boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{s}, \tag{3.10}$$

with $\boldsymbol{z}$ and $\boldsymbol{W}$ being the linear and qudratic slack weights, one can improve the feasibility of the NMPC problem, as the problem will no longer be infeasible if the system enters a state outside the original feasible set of Eq. (3.8). For the trajectory tracking NMPC problem we can then apply this principle to make the limits on $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ soft, as the chosen limits for these variables typically will be far lower than their actual maximum values.

3.2 Trajectory blending

So far a trajectory tracking controller has been developed, which tracks the target trajectory appropriately given that it's initial configuration is close to the initial target configuration. However, for safety-critical applications, it's important to also consider the approach towards the target trajectory. A typical example where this is useful is pick-and-place operations of moving objects, where the robot starts from some default configuration far away from the target and it is desirable to smoothly and safely approach the target. Feeding the target trajectory directly into the tracking controller will however create a large error signal, which might lead to overshoot. In this work, trajectory blending methods are introduced to solve this general problem of generating smooth motion from one pose trajectory to another. Note that other approaches could be considered as well, such as filtering the desired trajectory.

3.2.1 Position trajectory blending

The position $\boldsymbol{h}_p(t, s)$ is blended from the initial position $\boldsymbol{p}_0(t)$ to the position reference $\boldsymbol{p}_r(t)$ by interpolation

$$\boldsymbol{h}_p(t, s) = \boldsymbol{p}_0(t) + \beta(s)(\boldsymbol{p}_r(t) - \boldsymbol{p}_0(t)) \tag{3.11}$$

at every time step. The characteristics of the blended motion are dictated by the monotonically increasing function $\beta(s) : [0, 1] \mapsto [0, 1]$. The most basic case is for $\beta(s) = s$ where we get linear interpolation. The first such function that we will consider is the logistic function

$$\beta(s) = \frac{a}{1 + e^{-ks}} - b, \tag{3.12}$$

where the parameter k shapes the steepness of the blending and a and b are chosen such that the conditions $\beta(0) = 0$ and $\beta(1) = 1$ are satisfied, as discussed in Myhre 2016. This is satisfied for

$$a = 2\,\frac{1 + e^{-k}}{1 - e^{-k}}, \quad b = \frac{1 + e^{-k}}{1 - e^{-k}}.$$

This choice of blending function allows approximately linearly motion towards the target in the beginning and then the motion will slow down as we approach the target. One downside of this choice of $\beta(s)$ is that it will result in a step in the initial velocity, which will require a step in the initial joint motor torques to track correctly. This might damage the joint motors, and since the controller will try to minimize acceleration it will also lead to an initial jump in the tracking error that the controller needs to catch up with.

A possible choice to remedy this problem is to use the third-order exponential function

$$\beta(s) = c\,(1 - e^{-(\alpha s)^3}) \tag{3.13}$$

as in Rymansaib et al. 2013, where $\alpha > 0$ is a parameter that will change the shape of the resulting curve and c is chosen such that $\beta(1) = 1$. It can be shown that this is achieved for

$$c = \frac{1}{1 - e^{-\alpha^3}}. \tag{3.14}$$

The two blending shape functions are compared for different values of k and α in Figure 3.1.

3.2.2 Quaternion trajectory blending

Blending orientation trajectories is not quite as trivial as for position trajectories. Since the quaternion is constrained by the unit length constraint we cannot use the linear interpolation expression Eq. (3.11) directly. One simple option is to normalize the quaternion after linear interpolation (QLB). In the following, we will use spherical linear interpolation (SLERP), which linearly interpolates between the initial quaternion and the desired quaternion along

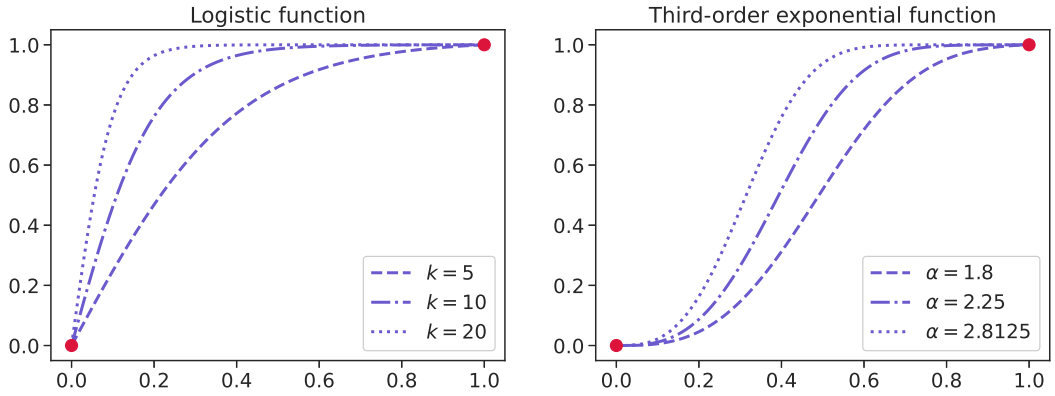


Figure 3.1: Logistic function and third-order exponential function for different values of k and α.

the 4-dimensional unit hypersphere

$$\boldsymbol{h}_o(\boldsymbol{q}_0, \boldsymbol{q}_d, s) = \boldsymbol{q}_0 \otimes (\boldsymbol{q}_0^{-1} \otimes \boldsymbol{q}_d)^{\beta(s)}, \tag{3.15}$$

using the quaternion properties defined in Section 2.1.1.4. The resulting operation results in a rotation with a constant angular velocity around a fixed axis for $\beta(s) = s$. Again by choosing a smooth, monotonically increasing $\beta(s)$ such as Eq. (3.12) or Eq. (3.13) we can achieve smooth motion between the initial orientation and the desired orientation.

An equivalent expression for SLERP (see Solà 2017) which is more suited for implementation in code is

$$\boldsymbol{h}_o(\boldsymbol{q}_0, \boldsymbol{q}_d, s) = \boldsymbol{q}_0 \frac{\sin((1-\beta(s))\Delta\theta)}{\sin(\Delta\theta)} + \boldsymbol{q}_d \frac{\sin(\beta(s)\Delta\theta)}{\sin(\Delta\theta)}, \tag{3.16}$$

where $\Delta\theta = \arccos(\boldsymbol{q}_0^\top \boldsymbol{q}_1)$.

In Figure 3.2 a simple example is considered, where we want to approach the trajectory of a pendulum while keeping a fixed orientation reference. Both the logistic function Eq. (3.12) and the third-order exponential Eq. (3.13) is studied. From the resulting blended reference trajectories, we see how the logistic function will result in a nonzero initial target velocity, while the exponential function will start with zero velocity before ramping up, as previously discussed.

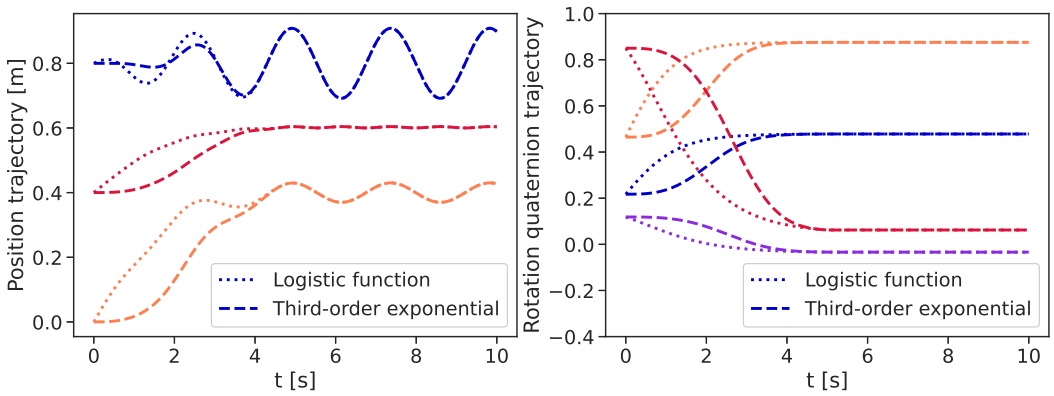


Figure 3.2: Blended position and orientation trajectories for pendulum motion with fixed orientation. The blending is done by linear interpolation of position and SLERP interpolation of the quaternion, where both a logistic function blend and a third-order exponential blend is shown.

3.3 Singularity avoidance

Singularity avoidance is a useful extension for a robot manipulator trajectory tracking controller for addressing safety concerns. This is especially motivated by the task space formulation of the tracking problem in Eq. (3.8), which does not take into account singularities, while a joint space trajectory could be directly designed to avoid the singular configurations of the robot. Especially for redundant robot manipulators this is useful as they have additional DOF they could use to optimize for manipulability while still tracking the desired pose trajectory. In this section, we consider one possible way to avoid singular configurations while tracking a trajectory, based on the manipulability index introduced in Section 2.1.3, and how to implement this in practice with QR decomposition.

3.3.1 Singularity avoidance cost function

In order to design a cost term for avoiding singularities, we first simplify the computation by considering the square of the manipulability index in Eq. (2.23)

$$\tilde{m}(\boldsymbol{q}) \equiv m(\boldsymbol{q})^2 = \det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\top), \tag{3.17}$$

in order to avoid the square root, which would introduce unnecessary non-differentiability in the cost function. A simple cost term to consider is then

$$\ell_{m1}(\boldsymbol{q}) \equiv -Q_m \tilde{m}(\boldsymbol{q})^2, \tag{3.18}$$

where $Q_m > 0$ is the singularity avoidance weight and the negative sign is introduced since we want to maximize manipulability. A square cost is considered as a sum of squares form is simpler to implement, but a cost that is linear in $\tilde{m}(\boldsymbol{q})$ is also a valid option.

In Lunni et al. 2017 a few other possible cost terms are explored, such as minimizing the conditioning number of $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\top$, which can be written as

$$\ell_{m2}(\boldsymbol{q}) \equiv Q_m \frac{\sigma_1}{\sigma_\mathrm{m}}, \tag{3.19}$$

with σ_1 and σ_m being the largest and smallest singular value of $\boldsymbol{J}$ respectively, as introduced in Section 2.1.3. Alternatively the rational expression

$$\ell_{m3}(\boldsymbol{q}) \equiv Q_m \frac{1}{\tilde{m}(\boldsymbol{q})^2} \tag{3.20}$$

is also possible. When comparing the cost terms we observe how Eq. (3.19) and Eq. (3.20) approaches infinity as the robot approaches a singular configuration, while Eq. (3.18) approaches zero which might increase the risk of entering a singular configuration. Furthermore, qualitatively Eq. (3.18) and Eq. (3.20) seem easier to implement, given a way to calculate $\tilde{m}(\boldsymbol{q})$ in a straightforward manner. This will be discussed in the following section.

3.3.2 Implementation using QR decomposition

One possibility for efficient computation of $\tilde{m}(\boldsymbol{q})$ is by using QR decomposition to factorize $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\top$ such that

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\top = \boldsymbol{Q}\boldsymbol{R}, \tag{3.21}$$

where $\boldsymbol{Q}$ is orthogonal and $\boldsymbol{R}$ is upper triangular. We have that

$$\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\top) = \det(\boldsymbol{Q})\det(\boldsymbol{R}), \tag{3.22}$$

and since $\boldsymbol{Q}$ is orthogonal we know that $|\det(\boldsymbol{Q})| = 1$. Finally, since $\boldsymbol{R}$ is upper triangular we get that $\det(\boldsymbol{R}) = \mathrm{tr}(\boldsymbol{R})$, such that

$$\tilde{m}(\boldsymbol{q})^2 = \mathrm{tr}(\boldsymbol{R})^2. \tag{3.23}$$

This provides a convenient and computationally efficient expression for evaluating the manipulability index.

3.4 Collision avoidance

When considering using robot manipulators in dynamic environments, such as environments with requirements of human-robot collaboration, it is a necessity that the robot is not only able to follow a desired trajectory but also dynamically avoid moving objects such as humans. The NMPC approach to solving the trajectory tracking problem can easily be extended to consider collision avoidance as well. By adding additional constraints and cost terms we can make the controller plan its motion around objects in its environment, as well as avoid self-collisions. In the following section, one approach to extend the original problem Eq. (3.8) will be considered, adapted from the work of Zube 2015 and Krämer et al. 2020. The reader is referred to these, as well as Cascio et al. 2009 and Homsi 2016 for further details on implementing collision avoidance in MPC approaches for robot manipulator planning and control.

3.4.1 Obstacle avoidance

The obstacle avoidance approach in this work is based on approximating the body of the robot manipulator by a set of n_R robot spheres, which is denoted by L_R. Each sphere is described by its corresponding link l_i, the translation from the link frame origin to the sphere center $\boldsymbol{p}_{R,i}$ and the radii $R_{R,i}$. A simple illustration of this concept for a single link of the robot is shown in Figure 3.3. Furthermore, a set of n_O obstacle points L_O in the cartesian space is considered, which may be static or dynamic (in the sense that they follow some known trajectory that varies with time). The obstacle spheres have center points $\boldsymbol{p}_{O,j}$ and radii $R_{O,j}$. Extending the optimal control problem Eq. (3.8) to consider obstacle avoidance is possible by formulating constraints that restrict the minimum distances between all possible pairs of robot points and obstacle points to avoid collisions.

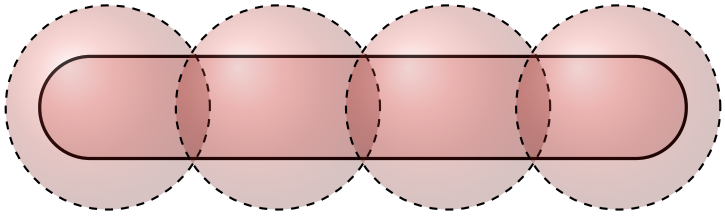


Figure 3.3: Sphere approximation of robot link l_i.

It should be noted that many other approaches exist, such as approximating the robot links as ellipsoids as in Castillo-Lopez et al. 2018, line-swept spheres as considered in Krämer et al. 2020 or as a set of polyhedra as in Gerdts et al. 2012. The decision of using spheres which is also considered in Zube 2015 is motivated by its simplicity. The resulting collision avoidance constraints are then analytic, smooth, computationally efficient and easy to implement for a numerical optimization solver. This however comes at the drawback

of needing many sphere instances to cover the robot body precisely, when comparing to ellipsoids or line-swept spheres for example. This also makes it a non-trivial task to choose how to optimally approximate the robot body with as few spheres as possible and with as little extra volume outside the robot body as possible. One approach is to construct a sphere tree representation of the robot, as in Zhao et al. 2018, but this will not be explored further here.

The kinematics of the points on the robot body are given by:

$$\begin{bmatrix} \boldsymbol{p}_{R,i}(\boldsymbol{q}) \\ 1 \end{bmatrix} = \boldsymbol{T}_{l_i}^{b}(\boldsymbol{q}) \begin{bmatrix} \boldsymbol{r}_i \\ 1 \end{bmatrix}, \tag{3.24}$$

with $\boldsymbol{T}_{l_i}^{b}$ being the homogeneous transformation matrix from the frame of link l_i that the control point is located on to the robot base frame. The obstacle positions are given by $\boldsymbol{p}_{O,j}(t)$, which may be a constant reference or a time-varying trajectory.

The geometry of the obstacle avoidance problem for a single robot point $\boldsymbol{p}_{R,i}$ and obstacle point $\boldsymbol{p}_{O,j}$ is shown in Figure 3.4. The squared Euclidean distance between the robot point and the obstacle point is then

$$d_{i,j}(\boldsymbol{q})^2 = \|\boldsymbol{p}_{R,i}(\boldsymbol{q}) - \boldsymbol{p}_{O,j}(t)\|^2. \tag{3.25}$$

Furthermore, collisions are only considered between a robot point and an obstacle point if the distance is less than the activation threshold $\beta_{i,j}$, as depicted in Figure 3.4. $\beta_{i,j}$ is given by

$$\beta_{i,j} = R_{R,i} + R_{O,j} + \beta_{\text{min}}, \tag{3.26}$$

with β_{min} being the distance between the sphere surfaces for which collisions will no longer be considered. This threshold is introduced to simplify the optimization landscape in complex environments, as well as to reduce computational cost. Especially for mobile manipulator systems this extension is useful, as they might traverse environments consisting of hundreds of obstacles during operation, yet only need to consider a few of those obstacles at a given location in the environment.

We first introduce hard constraints on the distance between every pair of robot spheres and obstacles:

$$d_{i,j}(\boldsymbol{q})^2 > \delta_{O,i,j}^2, \quad \forall\, (i,j) \in L_R \times L_O, \tag{3.27}$$

with

$$\delta_{O,i,j} = R_{R,i} + R_{O,j} + d_{O,\text{min}} \tag{3.28}$$

being the minimum distance between the two points, and $d_{O,\text{min}}$ being a parameter for the minimum allowed distance between the sphere surfaces.

In order to define the cost function for the extended NMPC problem with obstacle

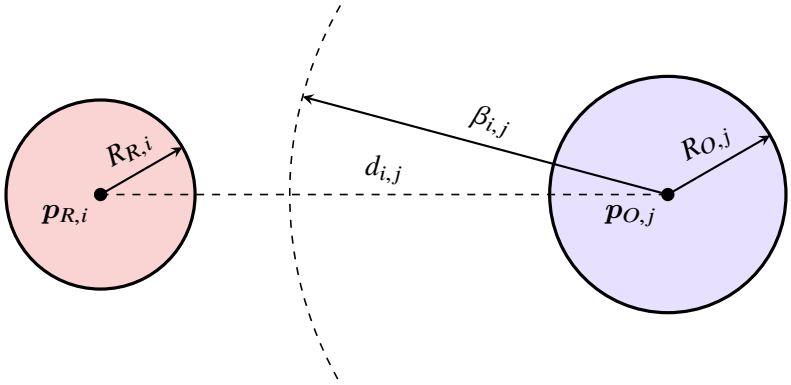


Figure 3.4: Obstacle avoidance problem for a single robot sphere in $p_{R,i}$ with radius $R_{R,i}$ and obstacle sphere in $p_{O,j}$ with radius $R_{O,j}$. The distance between the points $d_{i,j}$ and the activation threshold $\beta_{i,j}$ are also shown.

avoidance, we first consider the cost used in Zube 2015:

$$G_{z,i,j}(\boldsymbol{q}) = \frac{1}{d_{i,j}(\boldsymbol{q})^2 - \delta_{O,i,j}^2}. \tag{3.29}$$

Note that adding an activation threshold for this cost function results in a jump along the activation border given by $\beta_{i,j}$. The cost term was therefore modified to

$$G_{i,j}(\boldsymbol{q}) = \frac{\beta_{i,j}^2 - \delta_{O,i,j}^2}{d_{i,j}(\boldsymbol{q})^2 - \delta_{O,i,j}^2} - 1, \tag{3.30}$$

such that the property $\delta_{O,i,j} = \beta_{i,j} \implies G_{i,j}(\boldsymbol{q}) = 0$ was achieved. This cost will create a repulsive potential around the sphere obstacle that grows to infinity as the distance goes towards $\delta_{O,i,j}$, and is zero outside the outer sphere with radius $\beta_{i,j}$.

In order to make the cost go to zero outside the activation radius $\beta_{i,j}$ we use a similar approach as the one described in Tran et al. 2018. They propose to multiply the cost by a logistic function in order to achieve this boundary property. The same approach is used here, just written in terms of the square of the distance to simplify implementation:

$$F_{i,j}(\boldsymbol{q}) = \frac{1}{1 + \exp(\frac{d_{i,j}(\boldsymbol{q})^2 - \beta_{i,j}^2}{\gamma^2})}. \tag{3.31}$$

Here γ is used to shape the steepness of the curve. Since the logistic function, as depicted in Figure 3.5 for different values of γ, approaches a 0/1 barrier function around $\beta_{i,j}$ as γ goes to zero, it achieves the desired effect.

The combined collision avoidance cost for a single robot sphere and obstacle is then

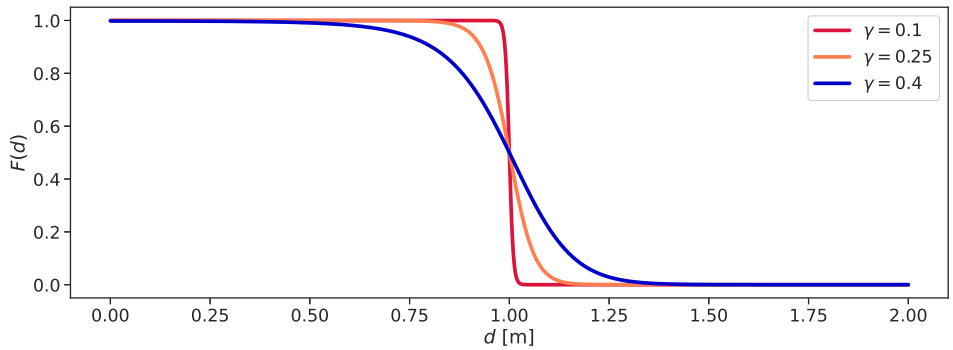


Figure 3.5: Logistic function $F(d)$ acting as a smooth barrier function, for activation threshold $\beta_{i,j} = 1.0$ and different values of γ.

given by:

$$C_{i,j}(\boldsymbol{q}) = Q_O F_{i,j}(\boldsymbol{q})^2 G_{i,j}(\boldsymbol{q})^2, \tag{3.32}$$

where $Q_O > 0$ is the weighting parameter. An example of how the cost function looks is given in Figure 3.6, with $Q_O = 0.03$, $\delta_{O,i,j} = 0.1$ and $\beta_{i,j} = 1$.

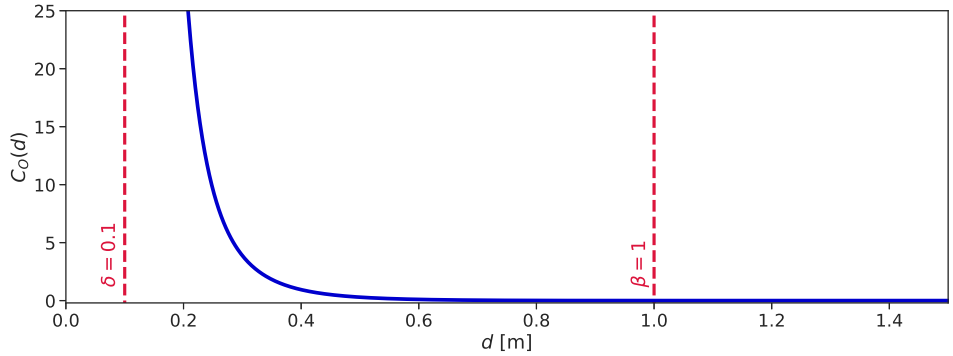


Figure 3.6: Collision avoidance cost for a single robot point and obstacle pair, with weight $Q_O = 0.03$, minimum distance $\delta_{O,i,j} = 0.1$ and activation threshold $\beta_{i,j} = 1$.

It should be noted that the cost term is not absolutely needed, since the hard constraints on distance in Eq. (3.27) will ensure no collisions. Then the robot would however only keep the minimum distance to the obstacle, which could be problematic, especially for moving obstacles. Adding the cost term helps the robot to traverse the environment in a safer way by maximizing distance to the obstacles while following the trajectory.

3.4.2 Self-collision avoidance

This framework is easily extended for self-collision avoidance as well by considering the set of pairs of links that potentially cause self-collisions $L_S \subset L_R \times L_R$. We will add distance constraints on all robot point pairs in this set. The self-collision constraint is then similarly as

before given by

$$d_{i,j}(\boldsymbol{q})^2 > \delta_{R,i,j}^2, \quad \forall\, (i,j) \in L_S, \tag{3.33}$$

with

$$d_{i,j}(\boldsymbol{q})^2 = \|\boldsymbol{p}_{R,i}(\boldsymbol{q}) - \boldsymbol{p}_{R,j}(\boldsymbol{q})\|^2, \quad \delta_{R,i,j} = R_{R,i} + R_{R,j} + d_{R,\min}, \tag{3.34}$$

and $d_{R,\min}$ being the minimum allowed distance between two robot sphere surfaces.

Observe how adding additional robot spheres leads to n_O new obstacle avoidance constraints linearly. For self-collision however, the number of constraints grows exponentially with n_R. A tradeoff therefore needs to made between the number of robot spheres and reduction in computation time from adding additional self-collision constraints.

Finally, we can write the extended NMPC problem with obstacle avoidance and self-collision as

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{i=0}^{N-1} \left(\|\boldsymbol{e}(\boldsymbol{x}_i)\|_{\boldsymbol{Q}}^2 + \|\ddot{\boldsymbol{q}}_i\|_{\boldsymbol{R}_a}^2 + \|\boldsymbol{u}_i\|_{\boldsymbol{R}_u}^2 + \frac{1}{n_R} \sum_{(j,k)\in L_R \times L_O} C_{j,k}(\boldsymbol{x}_i) \right) \\
& + \|\boldsymbol{e}(\boldsymbol{x}_N)\|_{\boldsymbol{Q}}^2 + \frac{1}{n_R} \sum_{(j,k)\in L_R \times L_O} C_{j,k}(\boldsymbol{x}_N) \\
\text{s.t.} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \quad i = 0, \ldots, N-1, \\
& \boldsymbol{x}_{\min} \leq \boldsymbol{x}_i \leq \boldsymbol{x}_{\max}, \quad i = 0, \ldots, N, \\
& \boldsymbol{u}_{\min} \leq \boldsymbol{u}_i \leq \boldsymbol{u}_{\max}, \quad i = 0, \ldots, N-1, \\
& \ddot{\boldsymbol{q}}_{\min} \leq \ddot{\boldsymbol{q}}_i \leq \ddot{\boldsymbol{q}}_{\max}, \quad i = 0, \ldots, N-1, \\
& d_{j,k}(\boldsymbol{x})^2 > \delta_{O,i,j}^2, \quad \forall\, (j,k) \in L_R \times L_O, \quad i = 0, \ldots, N, \\
& d_{j,k}(\boldsymbol{x})^2 > \delta_{R,i,j}^2, \quad \forall\, (j,k) \in L_S, \quad i = 0, \ldots, N, \\
& \boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0,
\end{aligned} \tag{3.35}$$

where we sum over all possible obstacle collisions in the cost and normalize by n_R to make the cost invariant to how many control points are used to model the manipulator. It should be emphasized that neither Zube 2015 nor Krämer et al. 2020 considers the full dynamics or time-varying obstacle trajectories in the NMPC problem, which comprise the main contributions to NMPC-based obstacle avoidance in this work. Finally, it is not shown for brevity in Eq. (3.35), but slack is added to the obstacle avoidance and self-collision avoidance constraints for better feasibility properties.

3.5 Software tools

In the following, the software tools used to implement the NMPC framework for robot manipulators are presented. The framework is written in Python and has the overall goal of

letting the user input a URDF model of the robot manipulator, and then generating a real-time feasible solver for the NMPC problem Eq. (3.8). This is achieved by using the urdf2casadi library to generate the forward kinematics and forward dynamics as analytic expressions, and then using the optimal control library acados to generate the solver, as illustrated in Figure 3.7. This will be discussed in further detail in the following.

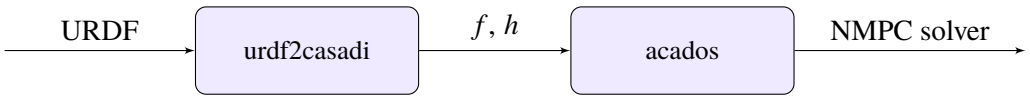


Figure 3.7: Architecture of the developed robot manipulator NMPC framework.

3.5.1 CasADi

CasADi (Andersson et al. 2019) is an open-source framework for nonlinear numerical optimization and optimal control. It is based on generating expression graphs from symbolic expressions, in order to compute algorithmic derivatives. The framework implements various differentiable operations for these graphs, such as exponential, logarithmic and trigonometric functions, and linear algebra based operations like matrix multiplication, trace and QR-decomposition, most of which are used in this NMPC implementation. One can solve a wide range of linear and nonlinear optimization problems, especially OCPs, using a set of different solvers. CasADi is written in C++, with Python and MATLAB interfaces available.

3.5.2 urdf2casadi

urdf2casadi (Johannessen et al. 2019) is a Python library for generating forward kinematics and forward and inverse dynamics of robots. It uses a robot modeled in the URDF format as input, which is an XML file format for representing robot models, used in the Robot Operating System (ROS). Using the URDF file the library generates symbolic CasADi expressions for the kinematics and dynamics, which can then be used for simulation and optimization.

3.5.3 acados

With the forward kinematics and dynamics given by urdf2casadi, the NMPC was then implemented using the optimal control library acados (Verschueren et al. 2019). It provides SQP and SQP RTI solvers for solving OCPs, which are targeted for fast embedded applications. It is implemented in C, and provides Python and MATLAB interfaces that allow C code generation of solvers for embedded systems. Furthermore, it provides compatibility with CasADi, such that the cost function and system model can be provided as CasADi expressions when generating the solver. These properties made the library ideal for combining with the

urdf2casadi library, with minimal need for developing additional interfaces between the libraries.

3.5.4 PyBullet

The Python package PyBullet (Coumans and Bai 2016–2020) was used to test the developed NMPC algorithms in simulation. PyBullet lets the user load robots into the simulation environment from URDF, and simulates the robot dynamics with joint limits and collision detection while allowing joint position, joint velocity and joint torque control.

3.6 Implementation on UR10e

The developed methods were tested on a 6 DOF Universal Robots UR10e robot. In Table 3.1 the DH parameters for the robot is given, and in Table 3.2 the joint angle, joint velocity and joint torque limits are given. Note that the base, shoulder, elbow and wrist 1-3 joints corresponds to q_1 - q_6. Furthermore, all the limits are symmetric, such that $q_{\min} = -q_{\max}$, $\dot{q}_{\min} = -\dot{q}_{\max}$, $\tau_{\min} = -\tau_{\max}$. A visualization of the UR10e in PyBullet is shown in Figure 3.8.

Table 3.1: DH parameters for the UR10e robot

Joint	θ [rad]	a [m]	d [m]	α [rad]
Base	0	0	0.1807	$\pi/2$
Shoulder	0	-0.6127	0	0
Elbow	0	-0.57155	0	0
Wrist 1	0	0	0.17415	$\pi/2$
Wrist 2	0	0	0.11985	$-\pi/2$
Wrist 3	0	0	0.11655	0

Table 3.2: Joint limits for the UR10e robot. The maximum joint angle $q_{\max}$, joint angle velocity $\dot{q}_{\max}$ and joint torque $\tau_{\max}$ are presented for every joint on the 6 DOF robot (*E-Series From Universal Robots*, *Max. Joint Torques* 2015).

	Base	Shoulder	Elbow	Wrist 1	Wrist 2	Wrist 3
$q_{\max}$ [deg]	360	360	360	360	360	360
$\dot{q}_{\max}$ [deg/s]	120	120	180	180	180	180
$\tau_{\max}$ [Nm]	330	330	150	56	56	56

The NMPC controller communicates with the UR10e robot using the Real-time Data Exchange (RTDE) interface, which provides a 500 Hz interface to the UR controller over an

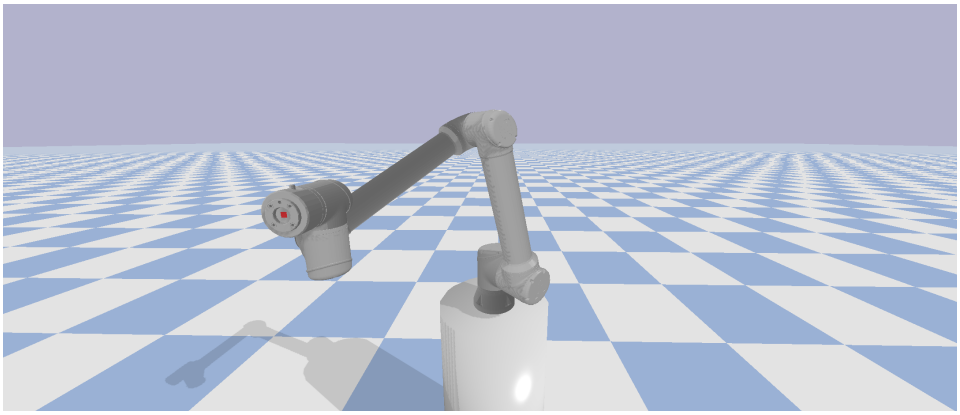


Figure 3.8: UR10e robot in PyBullet simulation environment.

IP connection, and is therefore suitable for real-time control of the robot. The open-source ur_rtde Python package (Lindvig 2020) was used to communicate with the robot over the RTDE interface. Since RTDE only provides rotation vectors when sampling the end effector pose of the robot, the conversion formulas between unit quaternions and rotation vectors in Eq. (A.3) and Eq. (A.4) were used. Universal Robots also provides the simulation environment URsim for testing communication and control of the robot, which was extensively used during testing of the control system.

Chapter 4

Results

The developed trajectory tracking NMPC was tested in the PyBullet simulation environment and in a real lab environment with a UR10e robot with a gripper attachment. In this section, results of the trajectory tracking controller following a pendulum trajectory in simulation and in the lab are first presented. Then results from a grasping test of a moving object are presented, before results considering singularity avoidance and collision avoidance are shown.

4.1 Trajectory tracking NMPC results

4.1.1 Pendulum trajectory in simulation

The trajectory tracking NMPC was first tested in simulation, with the task of tracking the motion of a linearized pendulum. This choice of trajectory was motivated by considering a pick-and-place operation of an object on a swinging hanger in an industrial setting. By approximating the motion of the hanger as the motion of a linearized pendulum we get a simple test case for the controller. We want to approach the pendulum over an approach horizon of T_a seconds, before tracking the position of the pendulum, while keeping a constant Euler angle reference $\mathbf{\Theta}_d$. Linear interpolation of position and SLERP interpolation of the quaternion with a third-order exponential as in Eq. (3.13) was used, with $\alpha = 1.8$.

For finding the desired trajectory in Cartesian space we consider the simple pendulum in Figure 4.1. The pendulum angle trajectory is given by the solution of the linearized pendulum differential equation

$$\theta(t) = \theta_0 \cos(\sqrt{\frac{g}{\ell}}t), \tag{4.1}$$

where θ_0 is the maximum amplitude, g is the gravity and ℓ is the pendulum length. By considering the transformation from spherical coordinates with origin in $\boldsymbol{p}_0 + [\ 0\ 0\ \ell\ ]^\top$ to

Cartesian coordinates we get

$$\boldsymbol{p}_d(\theta) = \begin{bmatrix} x_d(\theta) \\ y_d(\theta) \\ z_d(\theta) \end{bmatrix} = \boldsymbol{p}_0 + \ell \begin{bmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ 1 - \cos(\theta) \end{bmatrix}, \tag{4.2}$$

which provides a simple initial trajectory to test the performance of the trajectory tracking controller. Note that the orientation reference $\boldsymbol{\Theta}_d$ was picked such that the end effector will approach directly towards the pendulum, i.e. by the angle φ in the xy plane.

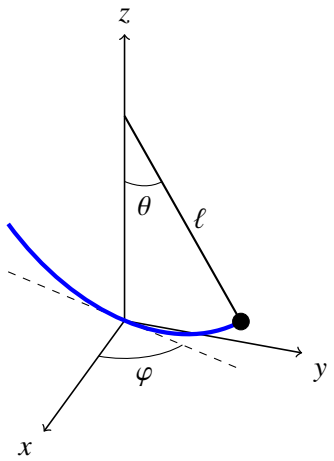


Figure 4.1: Pendulum with angle θ and length ℓ in 3D Cartesian space constrained to plane with rotation angle φ.

The solver was configured to use the ERK4 method in Eq. (2.28) to discretize the dynamics, using the Gauss-Newton Hessian approximation. Furthermore, acados formulates the numerical optimal control problem using the multiple-shooting method. The SQP RTI solver in acados was used with a tolerance of 10^{-4}, with the interior point QP solver HPIPM (Frison and Diehl 2020) for solving the QP at every time step in the SQP RTI algorithm. This configuration was empirically found to be a good tradeoff between efficiency and accuracy for solving the OCP in Eq. (3.8), and was used for all the subsequent tests in this chapter.

The URDF for the UR10e robot (from *ROS Industrial*) was loaded into the simulation environment and tested with motor torque control. An image of the simulation environment for testing the performance of the NMPC when tracking a pendulum motion can be seen in Figure 4.2. For this test, as well as for all the subsequent tests in this chapter, a computer with an AMD Ryzen 9 3900X CPU using 16 GB RAM was used. A MPC time horizon of $T_f = 0.4\,\text{s}$ with $N = 20$ steps was used, such that the sampling time was $t_s = T_f/N = 20\,\text{ms}$. Other relevant parameter configurations are given in Table 4.1, of which the addition of a quadratic cost term in $\dot{\boldsymbol{e}}_p$ with weight matrix $\boldsymbol{Q}_d$ is especially relevant. Also note how the

joint limits $q_{\max}$ and torque limits $\tau_{\max}$ are omitted, as the physical limits given in Table 3.2 were used for this and all subsequent tests. The joint velocity limits $\dot{q}_{\max}$ were however lowered from the physical maximum values to ensure a slower and smoother motion. Also note how adding additional safety margins around $q_{\max}$ and $\tau_{\max}$ is generally a good idea for safe deployment in a real industrial setting but was not included here.

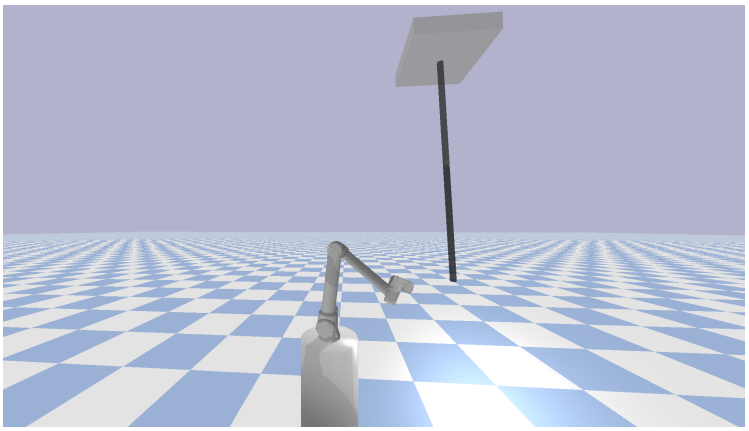


Figure 4.2: PyBullet simulation setup for tracking the motion of a pendulum.

Table 4.1: Configuration of NMPC parameters for test on pendulum trajectory in simulation.

Parameter	Value
N	20
T_f [s]	0.4
T_a [s]	5
$\boldsymbol{\Theta}_d$ [rad]	$[-\frac{\pi}{2}, 0.0, -\frac{\pi}{2} + 1.3]$
$\boldsymbol{q}_0$ [deg]	$[100, -84, -140, 10, 87, 0]$
$\boldsymbol{p}_0$ [m]	$[0.4, 0.8, 3.2]$
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	3.5
$\boldsymbol{Q}_p$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_d$	$10^1 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_q$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{R}_a$	$10^{-2} \cdot \boldsymbol{I}_6$
$\boldsymbol{R}_u$	$10^{-4} \cdot \boldsymbol{I}_6$

In Figure 4.3 the control input from the NMPC solver $\boldsymbol{u}^*$ is presented, and in Figure 4.4 the joint angles and joint velocities resulting from applying $\boldsymbol{u}^*$ in simulation are given. The corresponding end effector pose trajectory is given in Figure 4.5. Notice the approach in the

first $T_a = 5$ s where the initial pose of the end effector is blended with the desired trajectory, to generate a smooth motion towards the target.

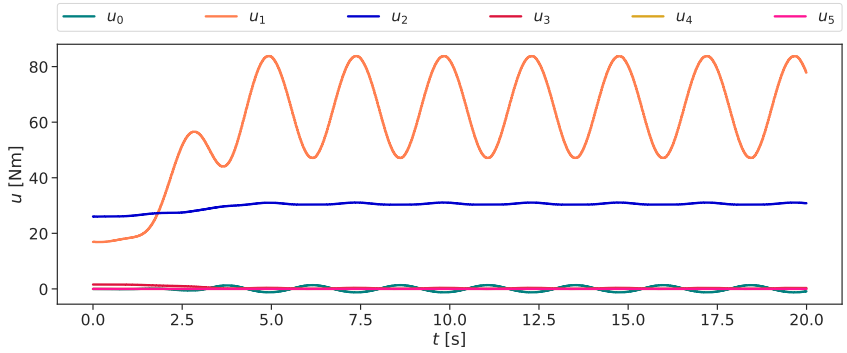


Figure 4.3: Computed control input $\boldsymbol{u}^*$ from NMPC solver for trajectory tracking test in simulation.

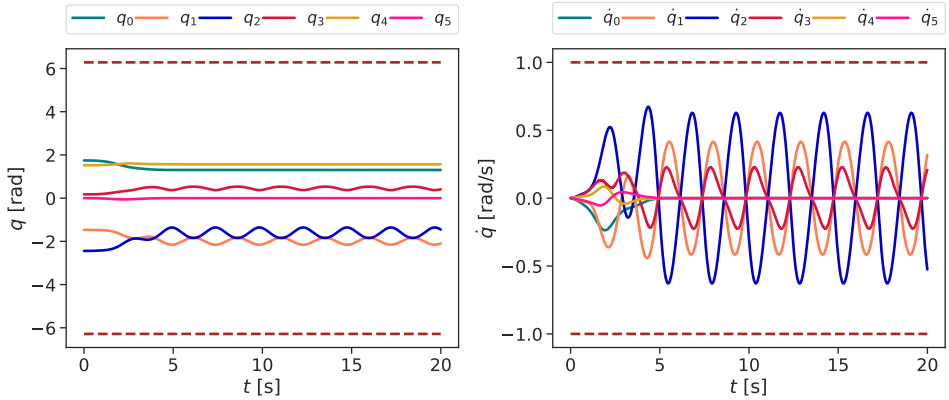


Figure 4.4: Resulting joint angle $\boldsymbol{q}$ and joint velocity $\dot{\boldsymbol{q}}$ trajectory for trajectory tracking test in simulation. The dashed lines indicate the joint limits.

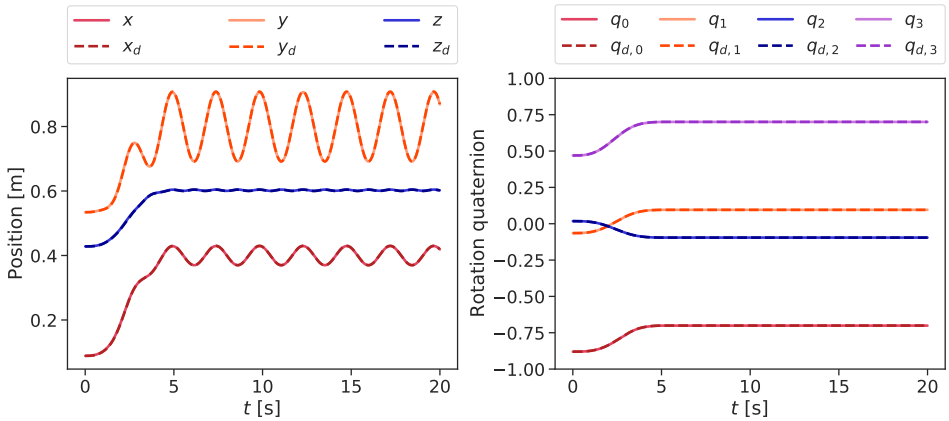


Figure 4.5: Desired end effector pose trajectory and resulting pose from simulation.

The tracking error is shown in Figure 4.6, and it can readily be observed that the worst-case

norm of the position tracking error is about 4 mm, while the worst-case norm of the tracking error in Euler angles is about 0.2 deg. Furthermore, it is observed that the error enters a cycle with the period of the pendulum trajectory, which is not asymptotically decreasing. It can be seen that the error peaks correspond to the maximum angles of the pendulum, which are the points of maximum acceleration.

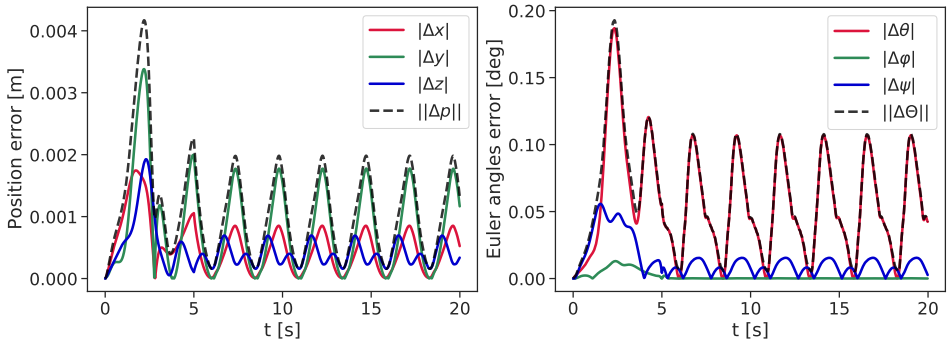


Figure 4.6: Resulting pose tracking errors. Euclidean norm of error is also shown.

Finally, in Figure 4.7 the solve time of the SQP RTI solver is presented. Specifically, both the timing of the preparation stage t_p and the feedback stage t_{fb} are given. First we see that the solver is able to run with a sampling time of 20 ms with sufficient margins, which corresponds to a frequency of 50 Hz. Secondly, this helps to illustrate the benefits of the RTI scheme, as discussed in Section 2.2.4. For the preparation stage the mean computation time was 4.77 ms during the test, while for the feedback stage it was 0.34 ms, which is over a magnitude lower. This shows how separating the SQP algorithm into these two steps drastically reduces feedback delay, as discussed in Section 2.2.4.

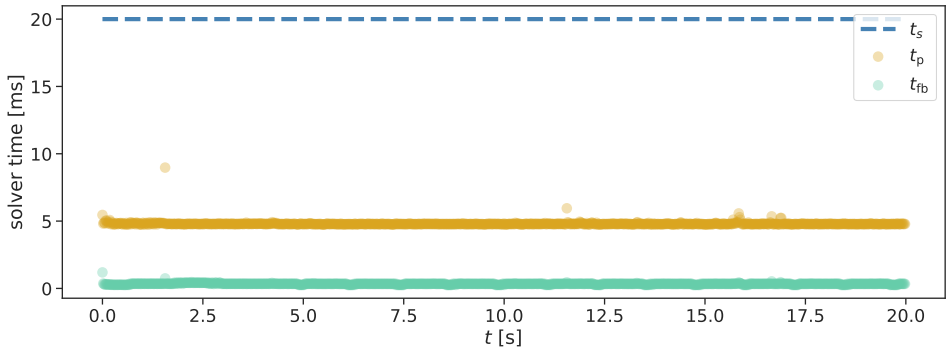


Figure 4.7: Solve time for preparation stage and feedback stage of SQP RTI solver.

4.1.2 Pendulum trajectory with UR10e

A similar test was then performed on a UR10e robot with the lab setup seen in Figure 4.8. The same trajectory and initial configuration of the robot was chosen, and the same tuning of

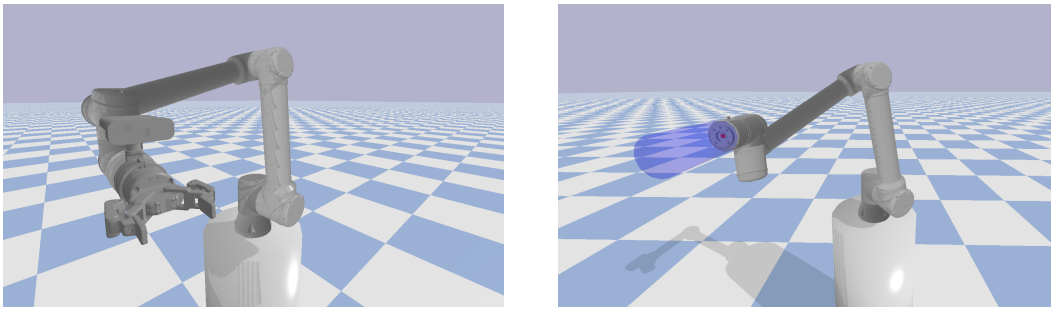
the NMPC was used as in Table 4.1. There are however a few significant differences between the simulation and the lab test, which will be discussed in the following, before the results are presented.

Figure 4.8: Lab setup with UR10e and Polaris Vicra optical tracker.

The first discrepancy is that several extensions were added to the end effector of the robot during the testing phase, specifically a SCHUNK AXIA FT-80 force/torque sensor, Azure Kinect camera, Robotiq 2F-85 gripper, as well as 3D-printed parts for assembling these together (see Figure 4.9). Testing the controller without this extra added mass and inertia taken into account led to oscillatory behavior. This was expected, as the controller is based on having an accurate URDF file for the robot, which evidently was no longer the case when additional parts were added.

Figure 4.9: UR10e with Robotiq 2F-85 gripper, Azure Kinect camera and SCHUNK AXIA FT-80 force/torque sensor attached.

This was solved by approximating the end effector attachment as a solid cylinder with



(a) UR10e with gripper. (b) Ur10e with cylinder approximation.

Figure 4.10: UR10e robot with gripper and cylinder approximation of gripper in PyBullet simulation.

constant mass density, such that the inertia is given by

$$I_{xx} = I_{yy} = \frac{1}{12}(3r^2 + l^2), \quad I_{zz} = \frac{1}{2}mr^2, \quad I_{xy} = I_{xz} = I_{yz} = 0, \tag{4.3}$$

where m is the mass, r is the radius and l is the length of the cylinder. Using the robot's end effector calibration procedure the mass was estimated to be $m = 2.48\,\text{kg}$. The radius of the cylinder was picked to be the radius of the tool flange of the robot, which is $r = 0.045\,\text{m}$ (*Universal Robots e-Series User Manual, UR10e*). The cylinder length that was used is the distance from the flange to the start of the gripper, found from the CAD of the end effector extension to be $l = 0.176\,\text{m}$. The inertia of the cylinder approximation of the extension was then calculated to be

$$I_{xx} = I_{yy} = 7.66 \times 10^{-3}\,\text{kgm}^2, \quad I_{zz} = 2.51 \times 10^{-3}\,\text{kgm}^2. \tag{4.4}$$

A visualization of the the actual attachment and the cylinder approximation is given in Figure 4.10a and Figure 4.10b respectively. It should be noted that this is a very crude approximation, since the assumption that the mass center is in $[\,0\;0\;\frac{l}{2}\,]^\top$ of the end effector link frame is evidently false. The mass distribution is far from uniform along the body, and especially the camera complicates the inertia matrix substantially. An area of further work is therefore to improve this approximation, possibly by use of CAD software directly. Note however that approximating the entire end effector link as a single rigid body will never be completely accurate, as closing and opening the gripper naturally changes the inertia over time. An even better approximation would therefore be to model the joints in the gripper individually. Nevertheless, the solid cylinder approximation is sufficient for testing the NMPC and comparing to the simulation case.

Another quite significant difference with the simulation in Section 4.1.1 is that UR robots do not provide direct torque control interfaces, only position and velocity control. Therefore,

in order to test the developed methods on the robot certain simplifications had to be made for control allocation. Specifically the joint velocity control interface was used instead, such that for a given solution $\boldsymbol{q}^*, \dot{\boldsymbol{q}}^*, \boldsymbol{u}^*$, the torque control input was not applied directly, but rather $\dot{\boldsymbol{q}}_1^*$ was applied as a joint velocity control input. This control allocation simplification will be further discussed in Chapter 5.

An additional quadratic cost term in joint velocity $\|\dot{\boldsymbol{q}}\|^2_{\boldsymbol{R}_v}$ was added, with $\boldsymbol{R}_v = 1$. It was observed that this helped with reducing oscillatory behavior in the system. Finally, note that since the UR10e robot has active gravity compensation, the dynamics were configured with an all-zero gravity vector, as opposed to the simulation case. The constant offsets in the control input for counteracting gravity observed for the simulations e.g. in Figure 4.3 are therefore not present in the tests on the real robot.

In Figure 4.11 the "virtual" torque control input is presented, and in Figure 4.12 and Figure 4.13 the resulting joint space and task space trajectories are shown. Note how the initial conditions in task space is a little different to the simulation, even though the initial conditions are the same in joint space. This is because of the additional transform added from the end effector attachment. The joint and task space trajectories are otherwise similar, yet the computed torque is substantially noisier and more oscillatory, in addition to the previously discussed missing offset to counteract gravity. This is likely a result of modeling errors leading to the NMPC producing a more oscillatory control input.

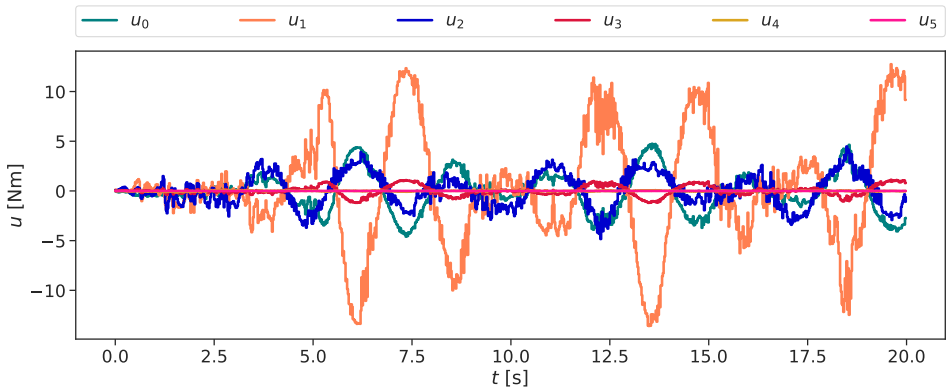


Figure 4.11: Computed control input $\boldsymbol{u}^*$ from NMPC solver for trajectory tracking test on UR10e.

From the tracking error in Figure 4.14 we see that the error is noisier and of about the same magnitude as in Figure 4.6, with errors peaking at about 5.5 mm and 0.25 deg for position and Euler angles respectively.

Finally, we observe a significant increase in computation time in Figure 4.15, with mean computation times of 8.99 ms and 0.459 ms for the preparation phase and feedback phase respectively. Yet this is still far within the total sample time of $t_s = 20$ ms. It is of interest that when comparing to Figure 4.7, there is a larger variance in computation time, and

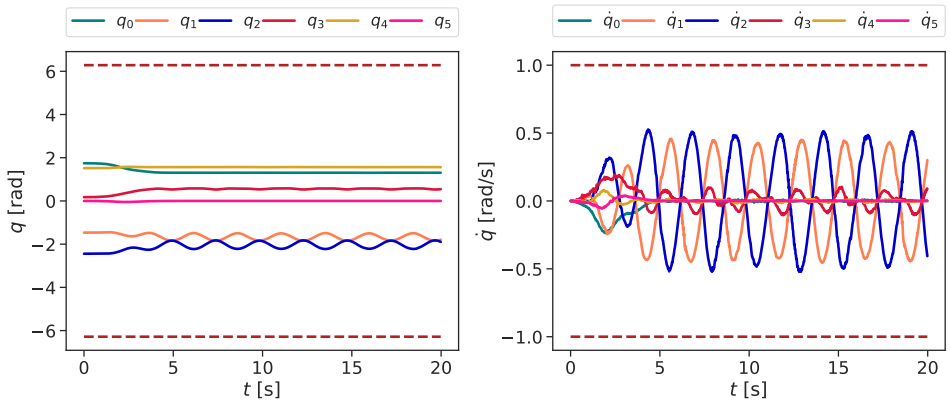


Figure 4.12: Resulting joint angle $\boldsymbol{q}$ and joint velocity $\dot{\boldsymbol{q}}$ trajectory for trajectory tracking test on UR10e.

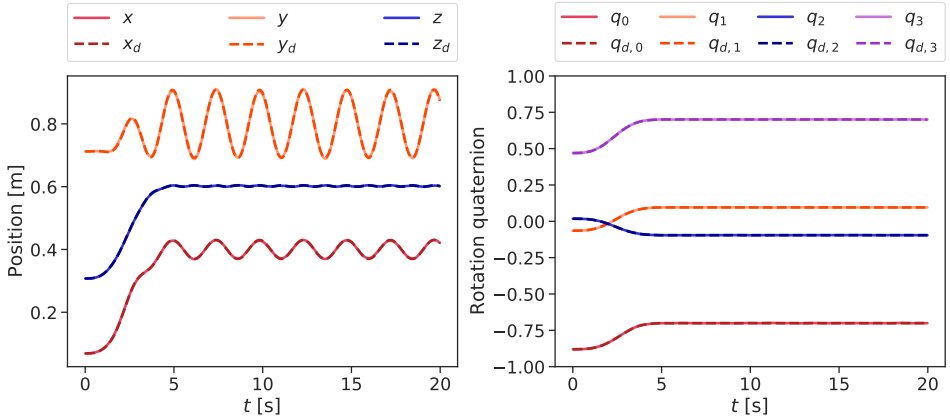


Figure 4.13: Desired end effector pose trajectory and resulting pose from UR10e trajectory tracking test.

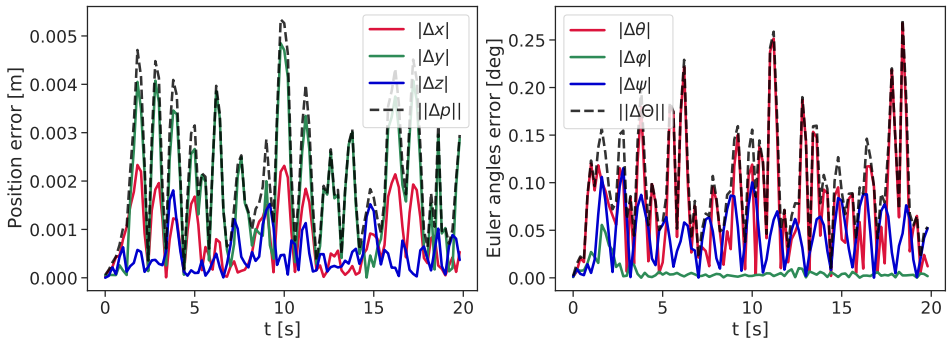


Figure 4.14: Resulting pose tracking errors. Euclidean norm of error is also plotted.

some structure emerges. Both this longer computation time and larger empirical variance in computation time is likely a result of taking the controller from simulation to a real environment, which brings noise, disturbances and modeling errors.

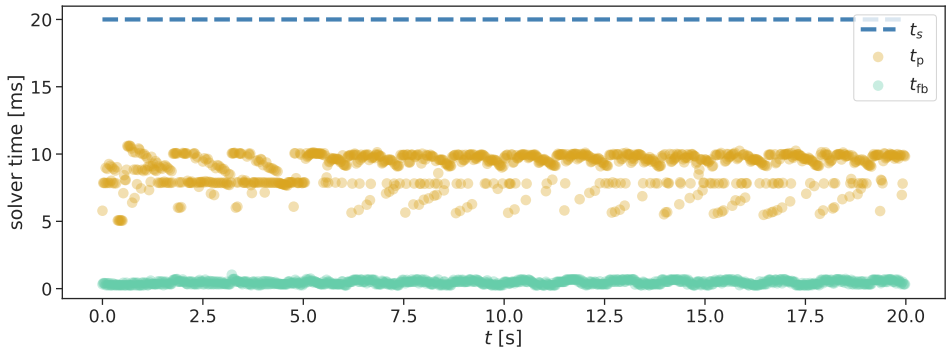


Figure 4.15: Solve time for preparation stage and feedback stage of SQP RTI solver.

4.2 Grasping moving object

In the following, we will consider having the robot perform a pick and place type operation of a moving object. As this work is motivated by human-robot collaboration and the emerging need for robot manipulators to perform tasks in dynamic environments, it seems like a fitting benchmark to test the system on the more realistic example of grasping a moving object. Specifically, the task will be to approach a moving optical pointer (held by a human), which will be tracked by the Polaris Vicra optical tracker system seen in Figure 4.8, grasp it with the gripper and move back to the initial configuration. An example of the gripper grasping the pointer can be seen in Figure 4.16.

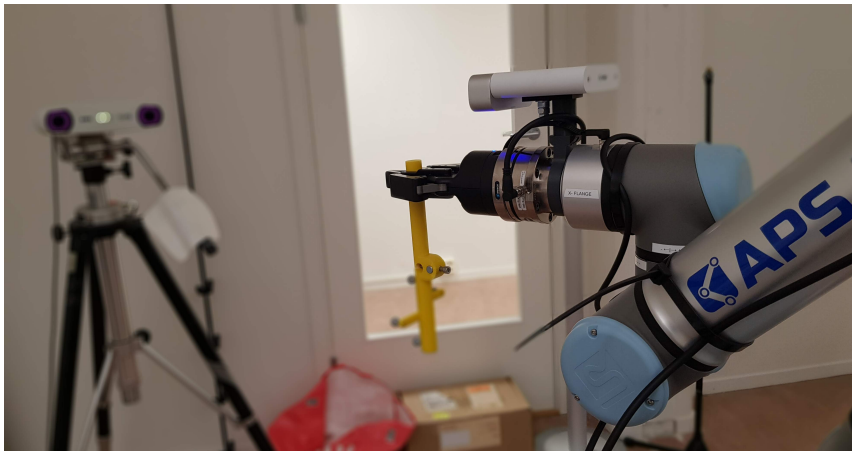


Figure 4.16: UR10e with gripper grasping pointer that is being tracked by Polaris Vicra optical tracker.

First, in order to get the pointer pose in the base frame, such that the robot is able to approach and grasp it, certain computations have to be made. The transform from the pointer to the tracker $\boldsymbol{T}_p^t$ is measured at every time step, and the transform from the tracker to the robot base frame $\boldsymbol{T}_t^b$ is known from calibrating the tracker, such that the pointer pose is given in base frame by $\boldsymbol{T}_p^b = \boldsymbol{T}_t^b \, \boldsymbol{T}_p^t$.

The experiment was then implemented as a simple timed sequence of actions. First the robot will move from the initial configuration and approach the pointer using the blending function in Eq. (3.13) over a time horizon of $T_1 = 7.5\,\text{s}$. Then the robot will wait and simply track the position of the pointer for a duration of $T_2 = 1.0\,\text{s}$, before using $T_3 = 2.25\,\text{s}$ to close the gripper. Finally, the robot will move back to the initial configuration while grasping the pointer over a duration of $T_4 = 7.5\,\text{s}$. The other parameters for the experiment are presented in Table 4.2.

Table 4.2: Configuration of NMPC parameters for UR10e grasping test of moving pointer.

Parameter	Value
N	15
T_f [s]	0.6
$\boldsymbol{\Theta}_d$ [rad]	$[-\frac{\pi}{2}, 0.0, 0.0]$
$\boldsymbol{q}_0$ [deg]	$[94, -64, -156, 39, 82, 0]$
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	2.5
$\boldsymbol{Q}_p$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_d$	$10^1 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_q$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{R}_a$	$10^{-2} \cdot \boldsymbol{I}_6$
$\boldsymbol{R}_u$	$10^{-4} \cdot \boldsymbol{I}_6$

The relevant results of this test can be seen in the position trajectory Figure 4.17 and the pose tracking error Figure 4.18. In the former, the different phases of the sequence are also visualized, and we can see how the robot is able to approach the pointer and grasp it, while it is moving, mostly in x and y direction. We observe a constant offset between the gripper and the pointer when the gripper grasps it, which is likely a consequence of the tracker having some small error from $\boldsymbol{T}_t^b$ not being completely accurate. We can also see that while the object is moving the tracking error is much larger since the controller is responding to unforeseen changes in the trajectory. The system is especially prone to this because it is assumed that the object is static over the MPC horizon, which was not the case for the predefined trajectory in Section 4.1. A possible extension to this system is therefore to use

tracking methods such as Kalman Filter or particle filter based approaches, to predict the future motion of the target and input this over the MPC horizon. Yet since the error peaks at about 13 mm and 0.25 deg it shows that the static target assumption works acceptably even when the object is moving.

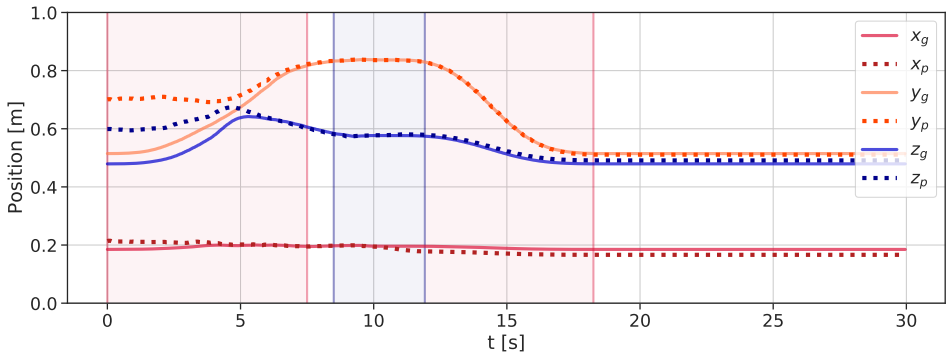


Figure 4.17: Position trajectory of gripper $\boldsymbol{p}_g$ and position trajectory of pointer object $\boldsymbol{p}_p$. The two red time intervals indicate the blending phases for approaching the pointer and moving back to the initial configuration. The blue interval corresponds to the time interval when the gripper is closing.

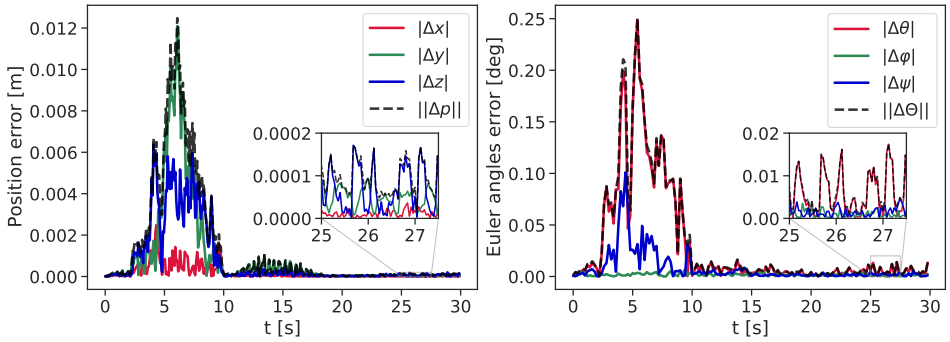


Figure 4.18: Pose tracking error from grasping moving object.

Note that for this experiment a longer sample time of 40 ms has been chosen. This is because the tracker runs at 60 Hz, or for this experiment a mean duration of 17.3 ms per time step, to sample the current pose of the pointer, and the call is blocking. The total time for the solver was 8.14 ms, so this blocking call to sample the current track increases computation time significantly. Finding ways to overcome this limitation of the tracker blocking while sampling, possibly by multithreading or using an alternative tracking system, is a possible area of further development.

Several other improvements could be made to the system. First, the pose information from the tracker could be used such that the robot approaches with orientation normal to the pointer, instead of having the desired orientation given and constant. This could help avoid colliding with the object and is evidently necessary for actual industrial use cases.

Furthermore, the implementation based on a predefined sequence of steps is not very robust or modular, and only serves as a small demo of the possible uses of the NMPC controller. Especially the trajectory blending for approaching and moving away with the pointer could be improved to not use a predefined amount of time, but rather make decisions dynamically, e.g. based on the distance to the pointer.

4.3 Singularity avoidance results

The singularity avoidance NMPC extension discussed in Section 3.3 was tested in simulation by having the robot follow a trajectory spanning a large part of the task space with certain points close to a singularity. Specifically, a trefoil knot position trajectory was considered:

$$\boldsymbol{p}_d(t) = \boldsymbol{p}_0 + a \begin{bmatrix} \sin(\omega t) + 2\sin(2\omega t) \\ \cos(\omega t) - 2\cos(2\omega t) \\ -b\sin(3\omega t) \end{bmatrix}, \tag{4.5}$$

with $\omega = 0.25\,\text{rad/s}$, $a = 0.10$ and $b = 2.5$, while keeping a constant quaternion reference $\boldsymbol{\Theta}_d$. The blended pose trajectory is shown in Figure 4.19. It was observed that this choice of trajectory produced configurations with low manipulability.

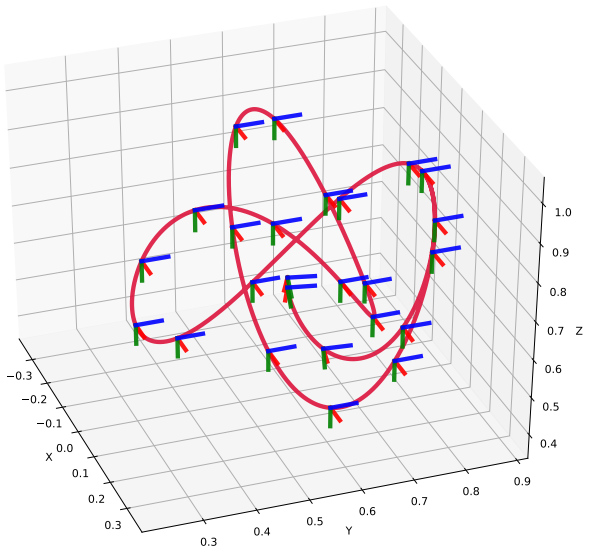


Figure 4.19: Blended trefoil knot trajectory for testing singularity avoidance. The red, green and blue axes represent the XYZ orientation of the end effector for those points of the trajectory.

The singularity avoidance cost in Eq. (3.20) was used, and implemented using QR decomposition, based on the brief discussion of the different alternatives in Section 3.3. The

parameters for the simulations are given in Table 4.3.

Table 4.3: Configuration of NMPC parameters for UR10e singularity avoidance test in PyBullet simulation.

Parameter	Value
N	20
T_f [s]	0.4
$\boldsymbol{\Theta}_d$ [rad]	$[-\frac{\pi}{2}, 0.0, 0.0]$
$\boldsymbol{q}_0$ [rad]	$[-2.0, -1.7, 2.1, 2.6, -1.6, 0.0]$
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	3.0
$\boldsymbol{Q}_p$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_q$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_m$	10^1
$\boldsymbol{R}_u$	$10^{-4} \cdot \boldsymbol{I}_6$
$\boldsymbol{R}_a$	$10^{-2} \cdot \boldsymbol{I}_6$

The resulting manipulability index values while tracking the trajectory with and without singularity avoidance is shown in Figure 4.20. We clearly observe that the NMPC is able to improve the manipulability index of the manipulator, especially in the local minima, which are also the most important as they are the critical points that are closest to the singularity. And it should be noted that the mean solve time only goes from 5.31 ms without singularity avoidance to 5.64 ms with, so little additional computational effort is needed. However, the improved manipulability comes at the cost of increased tracking errors, as seen in Figure 4.21. Especially we observe two error peaks that correspond to the two prominent local minima in Figure 4.20 where the robot is closest to a singular configuration. For this tuning the errors are also significantly larger in general, which shows a limitation of the NMPC approach, that trying to weight several different conflicting cost terms might result in overall lower performance and makes tuning quite difficult and localized to a single scenario. This is especially prominent for a 6 DOF robot, and lower errors might be observed when testing with a redundant robot manipulator with 7 or more DOF.

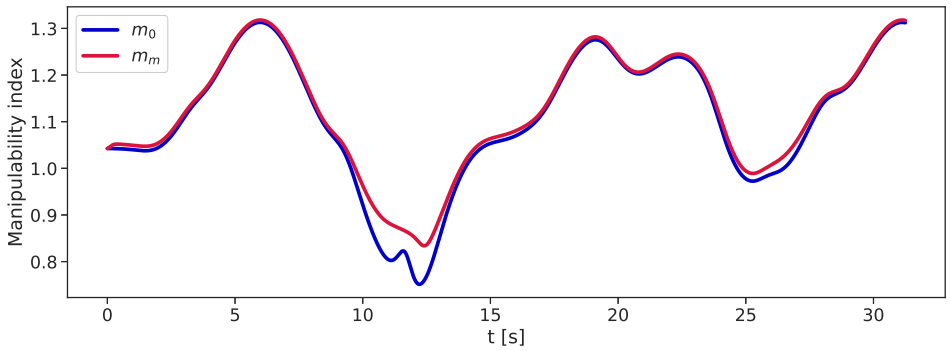


Figure 4.20: Manipulability index m_m and m_0 for NMPC respectively with and without singularity avoidance cost.

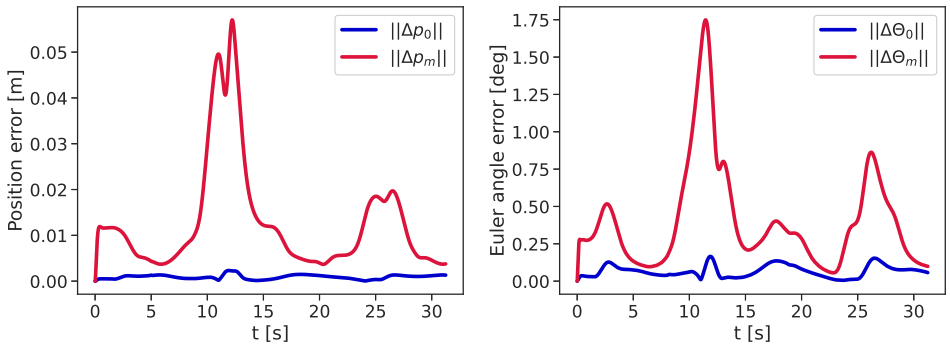


Figure 4.21: Resulting pose tracking error where the m and 0 subscripts correspond to with and without singularity avoidance respectively.

4.4 Collision avoidance results

4.4.1 Static obstacle in simulation

The trajectory tracking NMPC with obstacle avoidance and self-collision avoidance constraints was first tested in simulation. A single static sphere obstacle was considered for simplicity. The UR10e robot was approximated by $n_R = 16$ spheres of varying radii. The simulation setup with both the set of robot spheres and the obstacle is shown in Figure 4.22. The robot was given a circular position trajectory to follow in the xy-plane, while keeping a constant z value and orientation. In Figure 4.23 the desired trajectory, as well as the actual trajectory of the robot is shown.

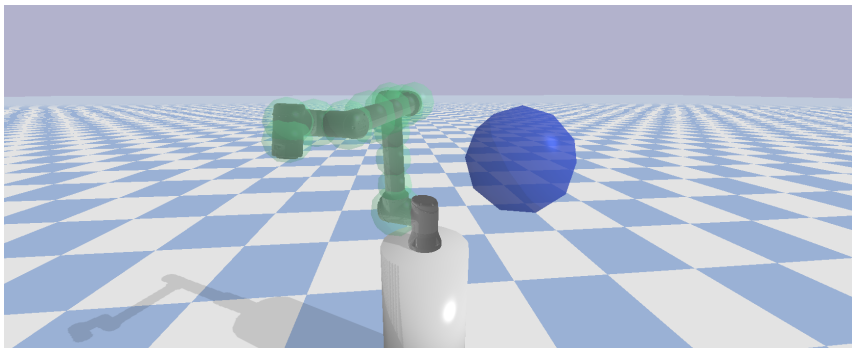


Figure 4.22: PyBullet simulation setup for testing collision avoidance.

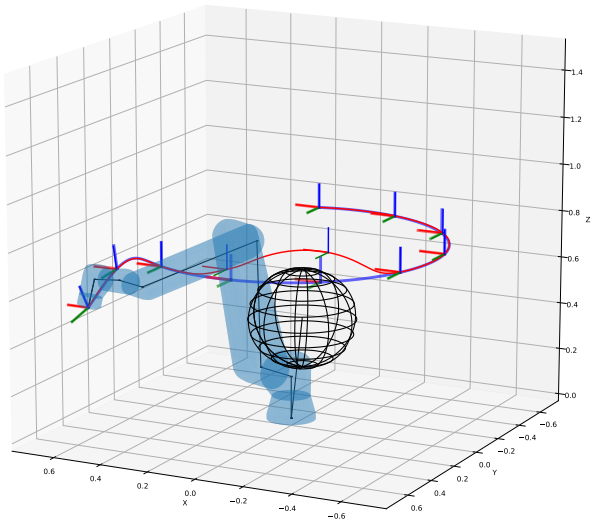


Figure 4.23: Geometry of the considered collision avoidance problem. Desired pose trajectory is shown in blue and actual pose trajectory is shown in red.

It was observed that the added nonlinear constraints increased computation time significantly, such that a time horizon of $T_f = 0.6\,\text{s}$ over $N = 20$ steps was now used, which

corresponds to 30 ms time steps. Other parameters for the experiment, such as the position $\boldsymbol{p}_O$ and radius R_O of the obstacle, and the radius r_d and height z_d of the planar circular trajectory is given in Table 4.4. Note that for the collision avoidance experiments slack variables were used on the joint velocity, joint acceleration, and collision avoidance constraints, with corresponding linear cost weights $\boldsymbol{z}_{\dot{\boldsymbol{q}}} = \boldsymbol{z}_{\ddot{\boldsymbol{q}}} = 10, z_{\text{ca}} = 10^3$ and quadratic cost weights $\boldsymbol{W}_{\dot{\boldsymbol{q}}} = \boldsymbol{W}_{\ddot{\boldsymbol{q}}} = \boldsymbol{W}_{\text{ca}} = 10^3$.

Table 4.4: Configuration of NMPC parameters for UR10e collision avoidance test in simulation.

Parameter	Value
N	20
T_f [s]	0.6
$\boldsymbol{\Theta}_d$ [rad]	$[\pi, 0.0, 0.0]$
$\boldsymbol{q}_0$ [rad]	$[1.2, -1.8, -1.5, -1.2, 1.5, 0.0]$
$\dot{q}_{\text{max}}$ [rad/s]	1.0
$\ddot{q}_{\text{max}}$ [rad/s^2]	5.0
$\boldsymbol{p}_O$ [m]	$[-0.3, 0.5, 1.6]$
r_O [m]	0.2
z_d [m]	0.75
r_d [m]	0.6
$d_{O,\text{min}}$ [m]	0.05
$d_{R,\text{min}}$ [m]	0.01
β_{min} [m]	1.0
$\boldsymbol{Q}_p$	$5 \cdot 10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_q$	$10^4 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_O$	10^{-2}
$\boldsymbol{R}_u$	$5 \cdot 10^{-4} \cdot \boldsymbol{I}_6$
$\boldsymbol{R}_a$	$10 \cdot \boldsymbol{I}_6$

In Figure 4.24 the control input and corresponding position trajectory are shown, and the resulting pose errors are given in Figure 4.25. As expected, a large error can be observed when the robot avoids the obstacle, before it continues along the desired trajectory with comparable tracking error to the previous simulation results.

Finally, in Figure 4.26 the distances between the obstacle and all the points on the robot body are shown. It is clearly seen that all the distances stay outside the specified minimum distance to avoid collisions.

The total mean computation time for the collision avoidance test in simulation was

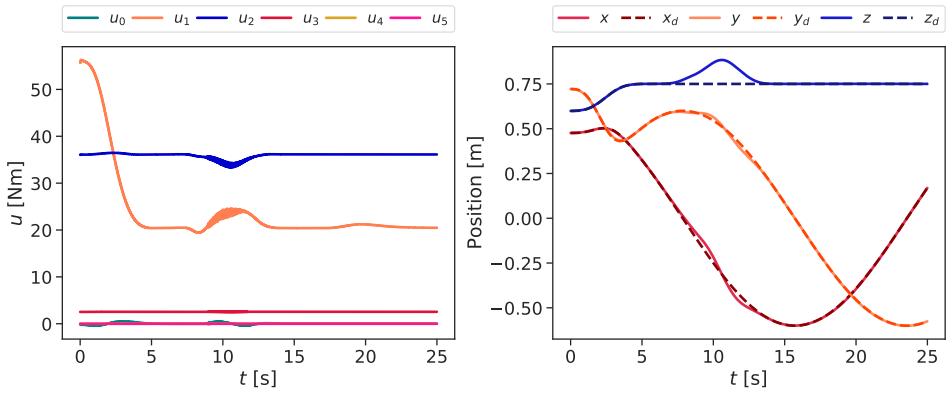


Figure 4.24: NMPC control input and pose trajectory for collision avoidance test in simulation.

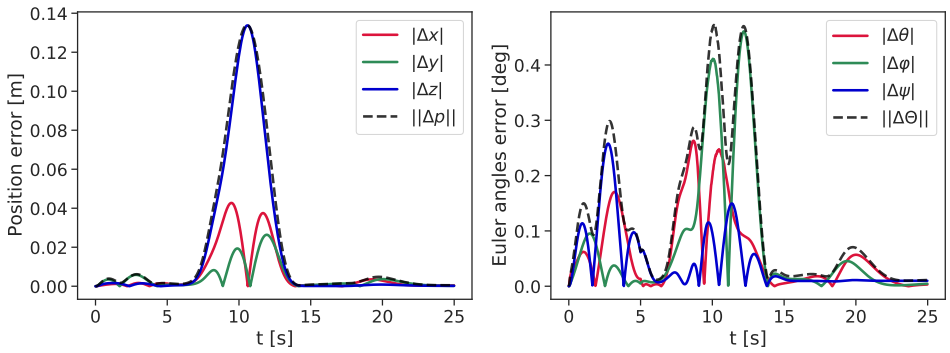


Figure 4.25: Pose tracking error for collision avoidance test in simulation.

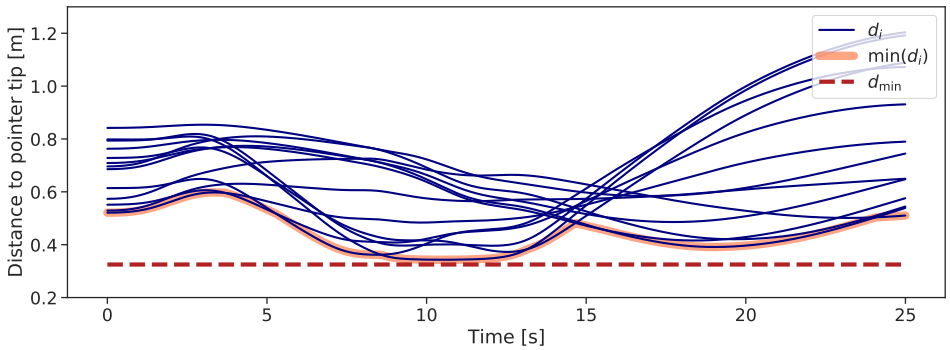


Figure 4.26: Distance between robot spheres and obstacle for collision avoidance test in simulation.

19.2 ms, which is a very notable increase when comparing to the test in Section 4.1.1 with no obstacle avoidance and self-collision avoidance.

Apart from the increased computation time, which is still real-time feasible, the solver produced the desired behavior and avoided the obstacle. During testing of the collision avoidance NMPC, it was however observed that the solver was prone to crashing from infeasibility in extreme cases. Specifically, the combination of limits on the joint velocity with $\dot{q}_{\max}$ and the highly nonlinear collision avoidance constraints lead to the internal QP solver reporting the problem to be infeasible when close to the minimum obstacle distance given by $\delta_{O,i,j}$. Adding slack to the collision avoidance constraints partly helped to mitigate this infeasibility problem, but the solver was highly sensitive to the tuning of the slack weights and the other weight matrices in general, and having slack on these constraints are not really wanted in the first place.

In a sense this displays some of the limits of this approach to the collision avoidance problem. The use of SQP RTI to solve NMPC problems with highly nonlinear constraints and conflicting nonlinear costs push the solver to its limits and numerical issues can arise, as observed here. An area of potential further research is therefore to formulate the problem to be more numerically stable and less dependent on having very precisely tuned parameters for a specific situation.

4.4.2 Moving obstacle with UR10e

One of the major benefits of the NMPC approach to collision avoidance is that moving obstacles, such as humans or other robots, can easily be considered by updating the corresponding sphere positions at every time step. This was tested in a realistic scenario by having the UR10e robot follow a simple trajectory while avoiding the pointer used previously in Section 4.2 that was tracked by the optical tracking system. The same setup as in Figure 4.8 was used, with a human holding the pointer. A trefoil knot position trajectory as in Eq. (4.5) was again considered, with $\omega = 0.5\,\text{rad/s}$, $a = 0.025$ and $b = 2.5$, while keeping a constant quaternion reference $\mathbf{\Theta}_d$. The blended trajectory is shown in Figure 4.27. The parameters for the test are given in Table 4.5.

The control input and end effector position trajectory from the test are shown in Figure 4.28, and the tracking error is shown in Figure 4.29. A large position error is observed as the robot avoids the pointer, which is being moved towards the robot to disrupt its nominal motion. The robot avoids the pointer successfully, as can be seen in Figure 4.30.

Earlier for the simulation test, we observed some oscillations when the robot is moving around the obstacle in Figure 4.24, and these oscillations are only more present when testing in the lab with a moving obstacle, as can be seen in Figure 4.28. The oscillations are likely a result of the conflicting trajectory tracking and collision avoidance cost terms in the cost function. When close to the obstacle these two cost terms will generate a border where the

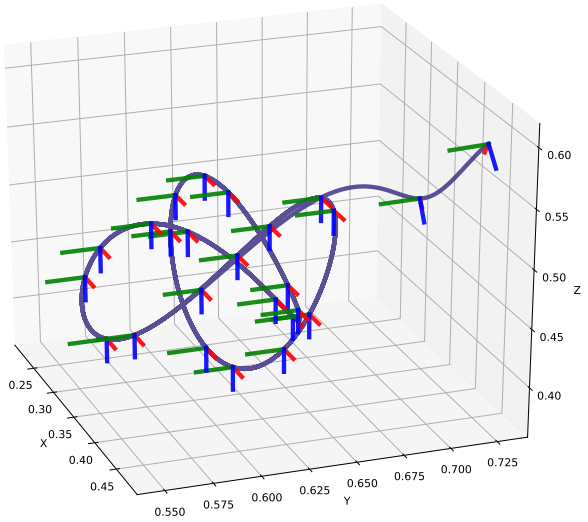


Figure 4.27: Blended trefoil knot trajectory.

Table 4.5: Configuration of NMPC parameters for UR10e collision avoidance test in lab.

Parameter	Value
N	15
T_f [s]	0.65
$\boldsymbol{\Theta}_d$ [rad]	$[\pi, 0.0, 0.0]$
$\boldsymbol{q}_0$ [rad]	$[1.2, -1.8, -1.5, -1.2, 1.5, 0.0]$
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	5.0
r_O [m]	0.2
$d_{O,\min}$ [m]	0.05
$d_{R,\min}$ [m]	0.01
$\beta_{\min}$ [m]	1.0
$\boldsymbol{Q}_p$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_q$	$10^4 \cdot \boldsymbol{I}_3$
$\boldsymbol{Q}_O$	10^{-3}
$\boldsymbol{R}_u$	$5 \cdot 10^{-4} \cdot \boldsymbol{I}_6$
$\boldsymbol{R}_a$	$2 \cdot \boldsymbol{I}_6$

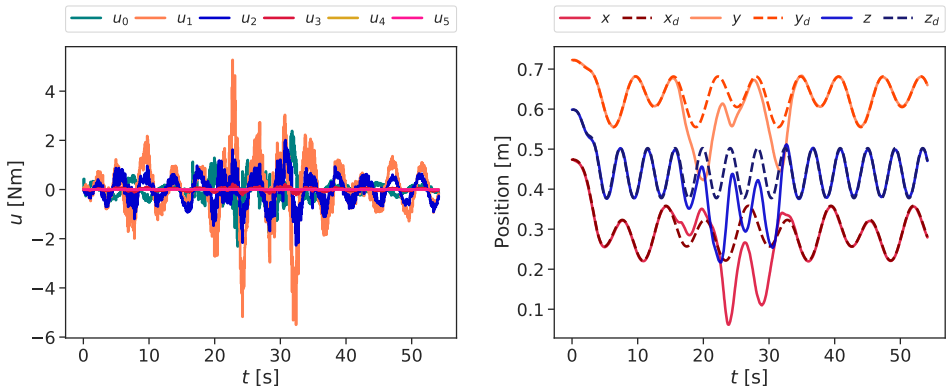


Figure 4.28: NMPC control input and pose trajectory for collision avoidance test with pointer.

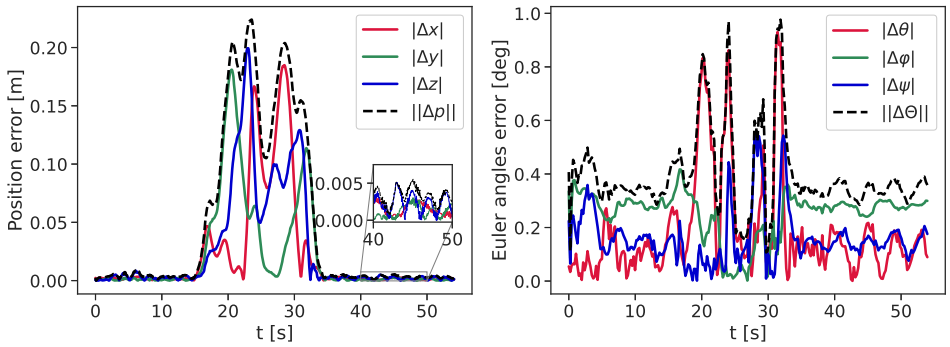


Figure 4.29: Pose tracking error for collision avoidance test with pointer.

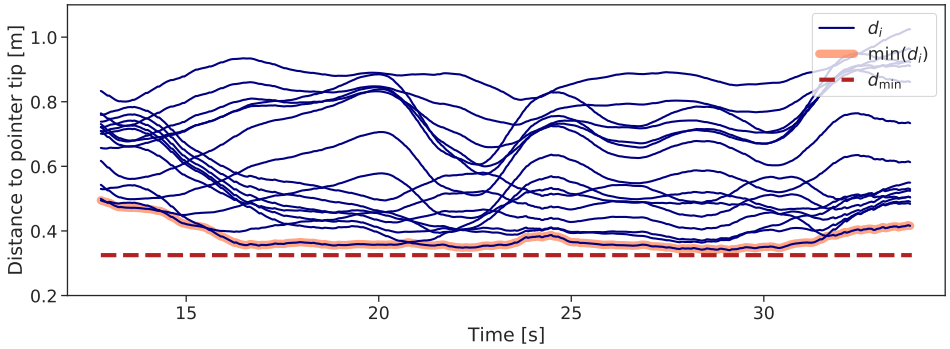


Figure 4.30: Distance between robot spheres and pointer.

cost terms are in balance. And because of the large sensitivity of the collision avoidance cost and constraints close to the obstacle, oscillations will occur, where the robot moves too close to the obstacle and the NMPC pushes the robot away, before the tracking error cost dominates the cost function and pushes it closer to the obstacle again and so on. Modeling errors, a moving obstacle and high computation time only amplify this effect further. Especially the issue with the optical tracker introducing a long time delay as discussed in Section 4.2 is therefore problematic, which lead to a mean computation time of 32.0 ms.

Chapter 5

Discussion and further work

In this chapter, the results from Chapter 4 will be discussed and possible areas of further work will be explored. Many specific issues and possible improvements have already been mentioned throughout Chapter 4, so in the following the emphasis will be on the overall performance and larger-scale problems that were observed throughout this work, as well as a larger outlook on further work.

Firstly, the performance of the trajectory tracking controller was found to be satisfactory, both in simulation in Section 4.1.1 and on the real UR10e robot in Section 4.1.2. The low tracking errors and real-time feasible computation time, even for quite fast and dynamic trajectories, show that the NMPC approach to the problem works well. Especially the simplification in the SQP RTI solver of only doing a single Newton step at every iteration seems to be reasonable. Based on the overall results, the developed NMPC framework for robot manipulator trajectory tracking is very promising, especially because of the inherent capability of handling dynamic trajectories and changing environments. However, several problems and areas of further improvement have been established during testing of the NMPC and will be discussed in detail in the following sections.

5.1 Benefits and limitations of NMPC based on full manipulator dynamics

The end effector that was added to the UR10e in Section 4.1.2 demonstrated an important limitation of the NMPC approach using the full equations of motion in Eq. (2.24) as dynamics constraints. As discussed in Section 4.2, running the controller without accurate knowledge of the added mass and inertia reduced performance substantially. This shows how the controller is specific to a single robot model, and even slightly changing the dynamic parameters can make the closed-loop system oscillatory. Firstly, this means that expert domain knowledge is needed for modeling the robot, and changing the robot will then require modifying the URDF

model accordingly, which is not trivial. This also shows how the controller is not very robust to modeling errors. Yet the use of the full dynamics in NMPC of robots allows for more dynamic maneuvers, which especially for dynamically stable floating base systems is very useful. Yet for the application of controlling a static base robot manipulator it is probably not needed in most applications, especially because industrial robot manipulators usually have a quite advanced internal controller for following the input signal.

The fact that robot manipulators usually have advanced internal controllers, and that full dynamics are usually not necessary then motivates using simple velocity control dynamics $\dot{\boldsymbol{q}} = \boldsymbol{u}$ instead, as discussed in Section 3.1. However, without going into more detail here, when comparing the simple dynamics approach to the full dynamics approach for more advanced tasks such as collision avoidance, the full dynamics approach had a better performance overall during testing. So for extensions of the system with increased requirements of agility, such as manipulator arms mounted on wheeled or legged robotics, the full dynamics formulation might be more suitable. A quite open area of further research is therefore to extend the NMPC to floating base mobile robotic systems and test this in simulation and in a real-world setting. It should be emphasized that the real-time trajectory tracking NMPC framework developed in this work is quite general, so using the same methods for mobile manipulator systems or even completely different robotic systems is entirely possible.

Furthermore, it might not even be necessary to run the NMPC in closed-loop directly, as that may interfere with the inner control loop, especially if the speed of the outer NMPC control loop approaches that of the inner loop. Yet from the observed results this was not considered a significant issue, because of the combination of the low feedback delay of the RTI scheme and the fast RTDE communication interface with the robot controller giving a low overall delay, as well as the sampling time of the NMPC being significantly larger than the sampling time of the internal robot controller in the UR10e.

Another problematic aspect of using an NMPC formulation based on the full robot dynamics is that the control variables are the joint torques, while commercial robot manipulator manufacturers typically only provide position and velocity control interfaces to their robots. This is for instance the case for UR robots like the UR10e. As previously discussed, the joint velocities at the following time step, i.e. $\dot{\boldsymbol{q}}_1^*$, was therefore used with velocity control instead. What then should happen is that the internal robot controller will apply a torque that accelerates the joints from the current velocity to the target velocity, given some maximum acceleration. This torque will be close to the "virtual" torque of the solver $\boldsymbol{u}_0^*$, but it will not be exactly the same. How the internal controller in the robot works is of course not generally known, so this is just an assumption one has to make when doing control allocation from torque control to velocity control for robot manipulators. Yet assuming a short sampling time and an accurate model of the robot dynamics, it is reasonable. This was also experimentally observed in simulation, as no noticeable differences were observed when comparing torque

control and velocity control for the same test parameters. Yet it does mean that we lose some of the rigidity of the torque and acceleration constraints in the optimization problem, as these variables are not exactly the same as the ones that will actually be used on the robot.

This discussion motivates several branches of further work. Firstly, one could explore other control allocation strategies from torque to velocity, or experiment further with the simple integrator dynamics. Secondly, it would be of interest to test the torque controller on a robot manipulator arm with an actual torque control interface, such as the Franka Emika Panda robot.

The discussion of the limitations of using URDF models for obtaining the system dynamics also motivates trying to learn the dynamics directly based on data. System identification of the robot model, e.g. by using Gaussian processes, is therefore an interesting area of further research. Particularly combining prior knowledge of the system with learning-based approaches to identify the dynamical model to be used in an NMPC seems promising for robot manipulator motion planning and control, and could build on a lot of the methods in this work.

5.2 SQP RTI and numerical issues

One of the main challenges during testing has been that adding highly nonlinear constraints and costs that are conflicting with each other creates an optimization problem that is not always well-behaved. Specifically, numerical instability and oscillatory behavior in the system has been repeatedly observed, most notably in Section 4.4. It clearly shows a general limitation of this approach, so further work on how to formulate a better-posed problem is definitely relevant.

One idea is to consider constraints and costs on $\dot{\boldsymbol{u}}$ in order to deal with the oscillations. This was briefly tested but was found to be difficult to implement in practice. This is because one would typically want to approximate $\dot{\boldsymbol{u}}$ with finite differences, which introduces coupling terms between optimization variables in different time steps of the NMPC problem, which is not possible in acados. An alternative is to let $\dot{\boldsymbol{u}}$ be the control input directly in the optimization problem, but this poses other problems when you want to input this new control input into your physical system, such as time delays.

Yet the main cause of the oscillations and numerical issues are likely the conflicting cost terms, so finding ways to better deal with these directly could also be explored. Multiple task-priority control methods for redundant robotic systems, that consider how to achieve multiple tasks with different priorities, are possibly better suited for dealing with such problems. So work on combining these methods and NMPC is relevant, possibly by solving nested optimization problems like in Lunni et al. 2017.

Furthermore, the SQP RTI solver in acados does not consider globalization, in order

to guarantee a reasonably consistent solve time, so getting stuck in local minima is also a concern. The sub-optimality of using the SQP RTI solver is also important to be aware of, especially for a trajectory tracking controller, where the cost function is changing at every time step. It could mean that the solver is always trying to catch up and never really converging. In theory, this is a large concern, as it could in worst-case lead to constraints not being satisfied and therefore posing a safety risk in a real-world scenario. However, from warm starts and the fast convergence rate of SQP, this was not seen as a big issue in practice.

5.3 Stability

A recurrent issue that was observed for the results in this work was a lack of asymptotic stability, both in simulation and lab experiments. Instead of converging asymptotically to the desired trajectory, the error converged to a cycle in the vicinity of it, as can for instance be seen in Figure 4.6. Several hypotheses for the cause of these cycles were investigated and will be summarized and discussed in this section.

Initially, it was hypothesized that the observed cycles around zero tracking error were caused by the RTI scheme causing the optimizer to not fully converge. However, configuring the solver to use normal SQP, i.e. running consecutive Newton steps until convergence produced the same result. This shows how the RTI scheme works well and that the assumption made with a single Newton step being sufficient seems to be reasonable. Yet that means the RTI scheme causing the solver to never converge is not the cause.

It was also believed that the nature of the trajectory tracking problem caused the solver to always lag behind, as the desired trajectory is changing at every time step. However, by studying setpoint regulation, i.e. using a constant desired pose trajectory instead, a constant offset was observed, analogous to the cycles that were observed for trajectory tracking.

A constant offset could point to modeling errors being the issue. And while there are certainly modeling errors for the real-world tests, for the simulation tests the exact same URDF was used to generate the dynamics constraints in the NMPC and to simulate the robot dynamics. Even when simulating the closed-loop system with ERK4, such that even the numerical integration scheme in the controller and in the simulation was the same, the same cycles were observed. Furthermore, it could also have been that inaccuracies in the ERK4 integrator in the NMPC were causing these issues. Yet trying different numerical integration schemes, both explicit and implicit, and with fewer or more steps, yielded the same resulting errors. Also modifying the solver tolerances had no noticeable effect.

So after ruling out most of the logical error causes, it would then seem that the resulting trajectories are indeed what the optimization solver actually finds to be optimal. And considering the NMPC solver tries to weight tracking error against torque and joint acceleration minimization it would then make sense that the manipulator will not track the

trajectory exactly. This however does not explain why it does not converge exactly for the setpoint regulation case, so further work should be done on investigating asymptotic stability of the developed NMPC. Yet it should be emphasized that this is a theoretical issue, and in reality the observed errors are so low that this can largely be ignored.

This discussion also motivates further experimentation with terminal sets and terminal costs, as this could improve stability. However, testing with a separate terminal cost on tracking error and having terminal constraints on tracking error had no noticeable effect.

5.4 Other

A few additional areas of discussion and further work are also worth mentioning here. Firstly, the computation times presented in this work have definitely shown that NMPC of 6 DOF robot manipulators with the full dynamics is real-time feasible. We have seen that this comes at the cost of sacrificing optimality and constraint satisfaction guarantees by using a real-time SQP solver. Yet further computation time decrease would of course be desirable, and parallelization is a topic which has not been considered for this work that has potential for significant speed improvements. To maximize the performance and capabilities of the system this should definitely be investigated further. It is worth noting that acados supports multithreading via the OpenMP implementation, so the already developed framework for NMPC could possibly be extended to use multithreading without having to port any code.

Another interesting area of further work is to extend the NMPC problem to consider uncertainties with stochastic NMPC. This ties nicely in with the previously discussed concept of using learning-based approaches for learning the dynamics, many of which are probabilistic. Then knowledge of the uncertainties of the model can be taken into account when finding the optimal control policy. If real-time feasibility can be maintained while doing online dynamics learning and stochastic NMPC is however questionable.

Furthermore, considering uncertainties is also very useful for collision avoidance. As previously discussed, it would be interesting to use some tracking strategy, e.g. a Kalman filter based tracker, to predict the future motion of moving obstacles and use this in the NMPC problem. Since most tracking strategies are probabilistic by nature and provide information about the uncertainties involved, it could potentially be included in the optimization problem.

Chapter 6

Conclusion

This report has investigated the use of NMPC for real-time task space trajectory tracking for robot manipulator arms. In order to answer the previously posed research questions in Chapter 1, a general framework for trajectory tracking NMPC was developed for robot manipulators, using an SQP RTI solver for solving the NMPC problem. The framework was tested on a UR10e robot, both in simulation and in a lab environment. The results presented in Chapter 4 showed satisfactory tracking accuracy, and the solver was able to run at 50 Hz. Furthermore, singularity avoidance, obstacle avoidance and self-collision avoidance was included in the NMPC problem and tested. The results showed that there was a significant increase in computation time for collision avoidance, yet the controller was still real-time feasible. This was demonstrated practically by having the robot grasp a moving object, as well as avoid a moving obstacle.

The approach presented in this work based on using the full nonlinear robot dynamics in the NMPC problem shows promise for performing agile maneuvers, but also has the limitation of needing an accurate model of the robot. Moreover, this work has illustrated how the RTI scheme is an effective strategy for employing NMPC for robotic systems with demanding real-time requirements. Yet the NMPC approach with highly nonlinear constraints also comes at the risk of numerical issues and oscillations. Despite these issues the developed trajectory tracking NMPC has shown to be very promising for autonomous and collaborative robotics applications.

Appendix A

Conversions between rotation representations

This appendix expands on the rotation representations discussed in Section 2.1.1, by providing addition conversion formulas used in this work, found in Diebel 2006 and Egeland and Gravdahl 2002.

A.1 Euler angles - unit quaternion conversion

The transformation from ZYX Euler angles $\boldsymbol{\Theta}$ to a unit quaternion $\boldsymbol{q}$ is given by

$$\boldsymbol{q}(\boldsymbol{\Theta}) = \begin{bmatrix} c_{\varphi/2}c_{\theta/2}c_{\psi/2} + s_{\varphi/2}s_{\theta/2}s_{\psi/2} \\ s_{\varphi/2}c_{\theta/2}c_{\psi/2} - c_{\varphi/2}s_{\theta/2}s_{\psi/2} \\ c_{\varphi/2}s_{\theta/2}c_{\psi/2} + s_{\varphi/2}c_{\theta/2}s_{\psi/2} \\ c_{\varphi/2}c_{\theta/2}s_{\psi/2} - s_{\varphi/2}s_{\theta/2}c_{\psi/2} \end{bmatrix}, \tag{A.1}$$

where we note $s_\alpha = \sin(\alpha), c_\alpha = \cos(\alpha)$. The inverse transformation is given by

$$\boldsymbol{\Theta}(\boldsymbol{q}) = \begin{bmatrix} \operatorname{atan2}(2\eta\varepsilon_1 + 2\varepsilon_2\varepsilon_3, \eta^2 - \varepsilon_1^2 - \varepsilon_2^2 + \varepsilon_3^2) \\ -\arcsin(2\varepsilon_1\varepsilon_3 - 2\eta\varepsilon_2) \\ \operatorname{atan2}(2\eta\varepsilon_3 + 2\varepsilon_1\varepsilon_2, \eta^2 + \varepsilon_1^2 - \varepsilon_2^2 - \varepsilon_3^2) \end{bmatrix}. \tag{A.2}$$

A.2 Rotation vector - unit quaternion conversion

From Eq. (2.7) we see that we map a rotation vector to a unit quaternion by

$$\boldsymbol{q} = \begin{bmatrix} \cos(\frac{\|\boldsymbol{v}\|}{2}) \\ \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}\sin(\frac{\|\boldsymbol{v}\|}{2}) \end{bmatrix}, \tag{A.3}$$

and the inverse map from a unit quaternion to a rotation vector is given by

$$\boldsymbol{v} = \frac{2\arccos(\eta)}{\sqrt{1-\eta^2}}\boldsymbol{\varepsilon}. \tag{A.4}$$

A.3 Rotation vector - rotation matrix conversion

Finally, the map from rotation vector to rotation matrix is given by

$$\begin{bmatrix} \boldsymbol{r}_1(\boldsymbol{v}) & \boldsymbol{r}_2(\boldsymbol{v}) & \boldsymbol{r}_3(\boldsymbol{v}) \end{bmatrix}, \tag{A.5}$$

where

$$\boldsymbol{r}_1(\boldsymbol{v}) = \frac{1}{\|\boldsymbol{v}\|^2}\begin{bmatrix} \left(v_1^2 - v_2^2 - v_3^2\right) s_{\frac{\|\boldsymbol{v}\|}{2}}^2 + \|\boldsymbol{v}\|^2 c_{\frac{\|\boldsymbol{v}\|}{2}}^2 \\ 2s_{\frac{\|\boldsymbol{v}\|}{2}}\left(v_1 v_2 s_{\frac{\|\boldsymbol{v}\|}{2}} - |\,\boldsymbol{v}\| v_3 c_{\frac{\|\boldsymbol{v}\|}{2}}\right) \\ 2s_{\frac{\|\boldsymbol{v}\|}{2}}\left(v_1 v_3 s_{\frac{\|\boldsymbol{v}\|}{2}} + |\,\boldsymbol{v}\| v_2 c_{\frac{\|\boldsymbol{v}\|}{2}}\right) \end{bmatrix}, \tag{A.6}$$

$$\boldsymbol{r}_2(\boldsymbol{v}) = \frac{1}{\|\boldsymbol{v}\|^2}\begin{bmatrix} 2s_{\frac{\|\boldsymbol{v}\|}{2}}\left(v_1 v_2 s_{\frac{\|\boldsymbol{v}\|}{2}} + |\,\boldsymbol{v}\| v_3 c_{\frac{\|\boldsymbol{v}\|}{2}}\right) \\ \left(v_2^2 - v_3^2 - v_1^2\right) s_{\frac{\|\boldsymbol{v}\|}{2}}^2 + \|\boldsymbol{v}\|^2 c_{\frac{\|\boldsymbol{v}\|}{2}}^2 \\ 2s_{\frac{\|\boldsymbol{v}\|}{2}}\left(v_2 v_3 s_{\frac{\|\boldsymbol{v}\|}{2}} - |\,\boldsymbol{v}\| v_1 c_{\frac{\|\boldsymbol{v}\|}{2}}\right) \end{bmatrix}, \tag{A.7}$$

$$\boldsymbol{r}_3(\boldsymbol{v}) = \frac{1}{\|\boldsymbol{v}\|^2}\begin{bmatrix} 2s_{\frac{\|\boldsymbol{v}\|}{2}}\left(v_1 v_3 s_{\frac{\|\boldsymbol{v}\|}{2}} - |\,\boldsymbol{v}\| v_2 c_{\frac{\|\boldsymbol{v}\|}{2}}\right) \\ 2s\frac{\|\boldsymbol{v}\|}{2}\left(v_2 v_3 s\frac{\|\boldsymbol{v}\|}{2} + |\,\boldsymbol{v}\| v_1 c\frac{\|\boldsymbol{v}\|}{2}\right) \\ \left(v_3^2 - v_1^2 - v_2^2\right) s_{\frac{\|\boldsymbol{v}\|}{2}}^2 + \|\boldsymbol{v}\|^2 c_{\frac{\|\boldsymbol{v}\|}{2}}^2 \end{bmatrix}. \tag{A.8}$$

The map from rotation matrix to rotation vector is given by

$$\boldsymbol{v} = \frac{1}{2}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}. \tag{A.9}$$

A.4 Rotation matrix - unit quaternion conversion

The transformation from unit quaternion to rotation matrix is given by

$$\boldsymbol{R}(\boldsymbol{q}) = \begin{bmatrix} \eta^2 + \varepsilon_1^2 - \varepsilon_2^2 - \varepsilon_3^2 & 2\varepsilon_1\varepsilon_2 + 2\eta\varepsilon_3 & 2\varepsilon_1\varepsilon_3 - 2\eta\varepsilon_2 \\ 2\varepsilon_1\varepsilon_2 - 2\eta\varepsilon_3 & \eta^2 - \varepsilon_1^2 + \varepsilon_2^2 - \varepsilon_3^2 & 2\varepsilon_2\varepsilon_3 + 2\eta\varepsilon_1 \\ 2\varepsilon_1\varepsilon_3 + 2\eta\varepsilon_2 & 2\varepsilon_2\varepsilon_3 - 2\eta\varepsilon_1 & \eta^2 - \varepsilon_1^2 - \varepsilon_2^2 + \varepsilon_3^2 \end{bmatrix}. \tag{A.10}$$

The inverse mapping is not quite as trivial, and in general four possible mappings exist. One possible mapping is

$$\boldsymbol{q}(\boldsymbol{R}) = \frac{1}{2} \begin{bmatrix} (1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{23} - r_{32}) / (1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{31} - r_{13}) / (1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{12} - r_{21}) / (1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \end{bmatrix}, \tag{A.11}$$

yet there exist cases where this transform is not defined, and all four mappings must be considered in general. The reader is referred to Diebel 2006 for further details.

Bibliography

Andersson, Joel A. E., Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl (2019). "CasADi – A software framework for nonlinear optimization and optimal control". *Mathematical Programming Computation* 11.1, pp. 1–36.

Arbo, Mathias Hauan, Esten Ingar Grøtli, and Jan Tommy Gravdahl (2017). "On model predictive path following and trajectory tracking for industrial robots". *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 100–105.

Brekke, Edmund (2020). "Fundamentals of Sensor Fusion". Unpublished.

Cascio, Joe, Mark Karpenko, Qi Gong, Pooya Sekhavat, and Isaac Michael Ross (2009). "Smooth proximity computation for collision-free optimal control of multiple robotic manipulators". *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2452–2457.

Castillo-Lopez, Manuel, Seyed Amin Sajadi-Alamdari, Jose Luis Sanchez-Lopez, Miguel A. Olivares-Mendez, and Holger Voos (2018). "Model Predictive Control for Aerial Collision Avoidance in Dynamic Environments". *26th Mediterranean Conference on Control and Automation (MED)*, pp. 1–6.

Coumans, Erwin and Yunfei Bai (2016–2020). *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `https://pybullet.org`.

Diebel, James (2006). *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Tech. rep. Stanford University.

Diehl, Moritz, Hans Joachim Ferreau, and Niels Haverbeke (2009). "Efficient numerical methods for nonlinear MPC and moving horizon estimation". In: *Nonlinear model predictive control*. Berlin, Heidelberg: Springer, pp. 391–417.

E-Series From Universal Robots. Universal Robots. URL: `https://www.universal-robots.com/media/1802432/e-series-brochure.pdf` (visited on 11/12/2020).

Egeland, Olav and Jan Tommy Gravdahl (2002). *Modeling and simulation for automatic control*. Trondheim, Norway: Marine Cybernetics.

Featherstone, Roy (2008). *Rigid Body Dynamics Algorithms*. New York, USA: Springer Science & Business Media.

Fossen, Thor I. (1999). *Guidance and Control of Ocean Vehicles*. Chichester, England: John Wiley & Sons.

Frison, Gianluca and Moritz Diehl (2020). "HPIPM: a high-performance quadratic programming framework for model predictive control". *arXiv preprint arXiv:2003.02547*.

Gerdts, Matthias, René Henrion, Dietmar Hömberg, and Chantal Landry (2012). "Path planning and collision avoidance for robots". *Numerical Algebra, Control & Optimization* 2.3, pp. 437–463.

Gros, Sébastien, Mario Zanon, Rien Quirynen, Alberto Bemporad, and Moritz Diehl (2016). "From linear to nonlinear MPC: bridging the gap via the real-time iteration". *International Journal of Control* 93.1, pp. 62–80.

Homsi, Saed Al (2016). "Online generation of time- optimal trajectories for industrial robots in dynamic environments". PhD thesis. Grenoble, France: Université Grenoble Alpes.

Huynh, Du Q (2009). "Metrics for 3D rotations: Comparison and analysis". *Journal of Mathematical Imaging and Vision* 35.2, pp. 155–164.

Jackson, Brian, Kevin Tracy, and Zachary Manchester (2021). "Planning with Attitude". International Conference on Robotics and Automation (Submitted).

Johannessen, Lill Maria Gjerde, Mathias Hauan Arbo, and Jan Tommy Gravdahl (2019). "Robot Dynamics with URDF & CasADi". *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*. IEEE, pp. 185–190.

Johansen, Tor A. (2011). "Introduction to nonlinear model predictive control and moving horizon estimation". In: *Selected topics on constrained and nonlinear control*. Bratislava, Slovakia: STU and Trondheim, Norway: NTNU, pp. 187–240.

Krämer, Maximilian, Christoph Rösmann, Frank Hoffmann, and Torsten Bertram (2020). "Model predictive control of a collaborative manipulator considering dynamic obstacles". *Optimal Control Applications and Methods* 41.4, pp. 1211–1232.

Lars, Grüne and Pannek Jürgen (2011). *Nonlinear model predictive control theory and algorithms*. London, England: Springer.

Lindvig, Anders Prier (2020). *ur_rtde*. `https://gitlab.com/sdurobotics/ur_rtde`. Gitlab repository.

Lunni, Dario, Angel Santamaria-Navarro, Roberto Rossi, Paolo Rocco, Luca Bascetta, and Juan Andrade-Cetto (2017). "Nonlinear model predictive control for aerial manipulation". *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 87–93.

Max. Joint Torques (2015). Universal Robots. URL: `https://www.universal-robots.com/articles/ur/max-joint-torques/` (visited on 11/12/2020).

Myhre, Torstein Anderssen (2016). "Vision-Based Control of a Robot Interacting with Moving and Flexible Objects". PhD thesis. Trondheim, Norway: NTNU.

Nie, Yuanbo and Eric C. Kerrigan (2018). "How Should Rate Constraints be Implemented in Nonlinear Optimal Control Solvers?" *IFAC-PapersOnLine*. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018 51.20, pp. 362–367.

Nocedal, Jorge and Stephen Wright (2006). *Numerical optimization*. New York, USA: Springer.

ROS Industrial. URL: `https://rosindustrial.org/` (visited on 12/06/2020).

Rymansaib, Zuhayr, Pejman Iravani, and M. Necip Sahinkaya (2013). "Exponential trajectory generation for point to point motions". *2013 IEEE/ASME international conference on advanced intelligent mechatronics*. IEEE, pp. 906–911.

Siciliano, Bruno, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo (2010). *Robotics: Modelling, Planning and Control*. London, England: Springer.

Solà, Joan (2017). "Quaternion kinematics for the error-state Kalman filter". *arXiv:1711.02508*.

Tran, Ngo-Quoc-Huy, Ionela Prodan, Esten Grøtli, and Laurent Lefèvre (2018). "Potential-field constructions in an MPC framework: application for safe navigation in a variable coastal environment". *IFAC-PapersOnLine*. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018 51.20, pp. 307–312.

Universal Robots e-Series User Manual, UR10e. Version 5.0.2. Universal Robots.

Verschueren, Robin, Gianluca Frison, Dimitris Kouzoupis, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Jonathan Frey, Thivaharan Albin, Rien Quirynen, and Moritz Diehl (2019). "acados: a modular open-source framework for fast embedded optimal control". *arXiv preprint arXiv:1910.13753*.

Wächter, Andreas and Lorenz T. Biegler (2006). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". *Mathematical Programming* 106.1, pp. 25–57.

Yoshikawa, Tsuneo (1985). "Manipulability of Robotic Mechanisms". *The International Journal of Robotics Research* 4.2, pp. 3–9.

Zhao, Yu, Hsien-Chung Lin, and Masayoshi Tomizuka (2018). "Efficient trajectory optimization for robot motion planning". *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, pp. 260–265.

Zube, Angelika (2015). "Cartesian nonlinear model predictive control of redundant manipulators considering obstacles". *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, pp. 137–142.