Master's thesis

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

Martin Albertsen Brandt

Trajectory Tracking for Fixed-Base and Floating-Base Robot Manipulators

A Gaussian Process-Based Model Predictive Control Approach

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Esten Ingar Grøtli, Phillip Maree
May 2021

NTNU
Norwegian University of
Science and Technology

Martin Albertsen Brandt

Trajectory Tracking for Fixed-Base and Floating-Base Robot Manipulators

A Gaussian Process-Based Model Predictive Control Approach

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Esten Ingar Grøtli, Phillip Maree
May 2021

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

NTNU
Norwegian University of
Science and Technology

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Technology in Cybernetics and Robotics at the Norwegian University of Science and Technology. It was conducted at the Department of Engineering Cybernetics, in collaboration with SINTEF Digital. The thesis was supervised by Professor Jan Tommy Gravdahl, with Senior Research Scientist Esten Ingar Grøtli and Research Scientist Phillip Maree as co-supervisors.

The thesis builds on the work done during my summer internship in the Robotics and Control group in SINTEF Digital, as well as the project report Brandt 2020, where Model Predictive Control (MPC) for robot manipulator trajectory tracking was investigated. In the report, it was found that one of the main practical limitations of the method was dependence on an accurate dynamical model. In this work, this limitation is addressed with a learning-based MPC, which is primarily based on Carron et al. 2019 and Hewing, Kabzan, et al. 2019.

Multiple tools and libraries were used in the implementation. Firstly, it was written in Python and used the NumPy library presented in Harris et al. 2020 extensively for computations. GPflow by Matthews et al. 2017 was used for training, which is built on the machine learning framework TensorFlow in Abadi et al. 2016. The libraries CasADi from Andersson et al. 2019, acados by Verschueren et al. 2019, and urdf2casadi by Johannessen et al. 2019 were used in the MPC implementation. The PyBullet physics simulator from Coumans and Bai 2016–2021 was used for simulations. Finally, access to a UR10e robot was provided, which was interfaced with using the ur_rtde library given in Lindvig 2021.

Acknowledgements

I would like to thank my supervisors Jan Tommy Gravdahl, Esten Ingar Grøtli, and Phillip Maree for all the valuable guidance they have given me these last two semesters. I am very grateful to be given the freedom to explore this topic freely while being given such great feedback and suggestions along the way. Furthermore, I would like to thank Research Manager Sture Holmstrøm for allowing me to stay at the office during the past two semesters. It has been very motivating to work on this project in such a rewarding environment with great colleagues. I would also like to thank my office mates at campus for great discussions. Finally, I would like to thank my family, friends, and girlfriend for their support.

Martin Albertsen Brandt
Trondheim, May 31, 2021

Abstract

Model Predictive Control (MPC) provides a useful framework for trajectory tracking for robot manipulator arms. However, the performance is highly dependent on an accurate model of the system dynamics, which, especially for floating-base robot manipulators, are not necessarily easy to obtain. In this work, uncertainty in the dynamical model is handled by adding a learned Gaussian process (GP) model to a feedback linearization-based prior dynamics model, which aims to model the error between the prior and true dynamics. This augmented model is added in a stochastic MPC formulation, where both joint space and task space trajectory tracking costs are considered. Furthermore, sparse GP methods, primarily the Sparse Variational Gaussian Process (SVGP) method, are applied. This is used in conjunction with a sequential quadratic programming (SQP) Real-time iteration (RTI) solver to achieve real-time feasible computation time for both fixed-base and floating-base manipulators systems.

Specifically, results from simulations and lab tests for a 6 degrees of freedom (DOF) fixed-base robot manipulator are presented, as well as simulation results for a free-floating space manipulator system. The GP-based MPC was compared to a Nonlinear Model Predictive Control (NMPC) approach using only the prior dynamics, as well as the feedback linearization-based approach without the added GP disturbance model. It was found that significant improvements in prediction and trajectory tracking accuracy could be achieved with the GP-based MPC approach. However, the high prediction accuracy of the GP was only local around the training trajectory. Yet the results showed that GP-based MPC, and more generally NMPC approaches, can follow challenging joint and pose trajectories with low tracking error and real-time feasible computation time, for both fixed-base and floating-base robot manipulator systems.

Sammendrag

Modellprediktiv regulering (MPC) er et nyttig rammeverk for banefølging med robotmanipulatorer. Ytelsen er derimot avhengig av en nøyaktig model av dynamikken til systemet, som spesielt for robotmanipulatorer med flytende base ikke nødvendigvis er lett tilgjengelig. I dette arbeidet håndteres usikkerheten i den dynamiske modellen ved å kombinere en Gaussisk prosess (GP) med en tilbakekoblingslinearisert model av dynamikken. Denne modellen anvendes i en stokastisk MPC regulator, hvor kostnadsfunksjoner i både leddrommet og oppgaverommet blir tatt i betraktning. Sparsomme GP metoder blir anvendt, i tillegg til en sekvensiell kvadratisk programmering (SQP) sanntidsiterasjon (RTI) løser, for å oppnå sanntid beregningstid for systemer med både statisk og bevegelig base.

Resultater fra simuleringer og labeksperimenter for en robotmanipulator med statisk base og 6 frihetsgrader blir presentert. Videre blir resultater fra simuleringer av en frittflytende robotmanipulator i verdensrommet også presentert. MPC regulatoren med GP-dynamikk ble sammenlignet med en ulineær MPC basert på den opprinnelige modellen og en tilbakekoblingslinearisert MPC uten GP-dynamikken. Det ble funnet at forbedringer i prediksjonsnøyaktighet og banefølgingsfeil kunne bli oppnådd med metoden med GP-modellen. Prediksjonsnøyaktigheten var derimot bare lokal til banen som ble brukt for å trene GPen. Resultatene viste allikevel at GP-basert MPC, og generelt metoder basert på ulineær MPC, kan følge kompliserte baner for leddvinkler eller posisjon og orientering med høy nøyaktighet i sanntid, for både robotmanipulatorer med statisk base og bevegelig base.

Contents

List of Tables

List of Figures

Acronyms

ADCS	Attitude Determination and Control System
CV	Cross-validation
DH	Denavit-Hartenberg
DOF	Degrees of freedom
ELBO	Evidence lower bound
ERK	Explicit Runge-Kutta
ERK4	Explicit Runge-Kutta of 4th order
FITC	Fully Independent Training Conditional
GP	Gaussian process
GPR	Gaussian process regression
i.i.d.	Independent and identically distributed
IRK	Implicit Runge-Kutta
KKT	Karush-Kuhn-Tucker
KL	Kullback-Leibler
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
NLP	Nonlinear program
NMPC	Nonlinear Model Predictive Control

OCP	Optimal Control Problem
QLB	Quaternion Linear Blending
QP	Quadratic program
RMSE	Root mean square error
ROS	Robot Operating System
RTDE	Real-time Data Exchange
RTI	Real-time iteration
SE	Squared Exponential
SGD	Stochastic gradient descent
SLERP	Spherical linear interpolation
SQP	Sequential quadratic programming
SVGP	Sparse Variational Gaussian Process
URDF	Unified Robot Description Format
VFE	Variational Free Energy

Notation

$\mathbb{E}[x]$	Expected value of random variable x
$\nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$	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 $\boldsymbol{x}$
$\boldsymbol{I}_m$	$m \times m$ identity matrix
$\boldsymbol{A}_{i,\cdot}$	i-th row vector of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{\cdot,j}$	j-th column vector of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{ij}$	Element of matrix $\boldsymbol{A}$ on row i and column j
$\boldsymbol{A}^\dagger$	Pseudoinverse of matrix $\boldsymbol{A}$
$\det(\boldsymbol{A})$	Determinant of matrix $\boldsymbol{A}$
$\mathrm{diag}(\boldsymbol{x})$	Diagonal matrix where diagonal is given by elements of vector $\boldsymbol{x}$
$\mathrm{diag}(\boldsymbol{A})$	Vector of diagonal elements of matrix $\boldsymbol{A}$
$\mathrm{blkdiag}(\dots)$	Block diagonal matrix
$\mathrm{tr}(\boldsymbol{A})$	Trace of matrix $\boldsymbol{A}$
$\mathbb{N}$	The set of natural numbers $\{1, 2, \dots\}$
$\mathbb{N}_a$	The set of all natural numbers up to $a \in \mathbb{N}$, i.e., $\{1, 2, \dots, a\}$
$\mathbb{N}^0$	The set $\mathbb{N} \cup \{0\} = \{0, 1, 2, \dots\}$
$\|\boldsymbol{x}\|$	Euclidean norm of vector $\boldsymbol{x}$

$\|\boldsymbol{x}\|_{\boldsymbol{Q}}^2$	Squared Euclidean norm of vector $\boldsymbol{x}$ weighted by matrix $\boldsymbol{Q} \succ 0$, i.e., $\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x}$
$\boldsymbol{A} \succ 0$	Positive definite matrix
$\Pr(X)$	Probability of event X
$\mathrm{sgn}(x)$	Sign function of scalar x, where $\mathrm{sgn}(x) = \|x\|/x$, for $x \neq 0$ and $\mathrm{sgn}(x) = 0$ for $x = 0$
$[\boldsymbol{x}]_\times$	Skew-symmetric matrix of vector $\boldsymbol{x}$

1 Introduction

1.1 Motivation

The emergence of autonomous robots, specifically the increasing use of cost-effective and collaborative robot manipulator arms, motivates the need for fast and accurate trajectory tracking methods. Especially the rise of autonomous floating-base systems, such as space manipulator systems and AUVs, have challenging requirements for autonomy. This motivates the development of trajectory tracking methods that are able to plan the robot motion in the presence of uncertainty and system constraints.

Optimal control provides a useful framework for motion planning and control for robotic systems. Finding the optimal control policy over a future time horizon while adhering to constraints is evidently useful for many robotics applications, especially when solving this problem in a receding horizon manner with Model Predictive Control (MPC). However, predictive control approaches are dependent on having an accurate model of the system dynamics, which, especially for complex and nonlinear systems, is not necessarily easy to obtain. Furthermore, data from tests on the system are often available. This motivates using learning-based approaches to learn the system dynamics, such as model-free methods built on deep learning, which have risen in popularity in recent years for robotics applications. On the other hand, simplified models based on first principles can still provide a decent estimate of the true dynamics. This calls into question the necessity of completely discarding these rudimentary models in learning-based methods, for instance, in regards to sample efficiency. This motivates learning a disturbance model which models the residual between the first principles model and the true dynamics, rather than the entire dynamics directly.

In this work, a Gaussian process (GP) is used to model the disturbance dynamics, based on the approach described in Hewing, Kabzan, et al. 2019 and Carron et al. 2019. Gaussian process regression (GPR) is a Bayesian inference method, meaning it is based

on formulating a prior and updating this prior belief based on observed data, giving the posterior. An advantage of learning such a disturbance model using Bayesian inference is that the resulting model allows for quantifying the uncertainty in the model. This allows for seamless integration with stochastic MPC, where the stochastic nature of the state can be explicitly considered by propagating the state uncertainty over the prediction horizon. Furthermore, this framework of including a Bayesian model in a stochastic optimization problem allows formulating constraints in terms of the probability of constraint violation, which enables tuning how cautious the controller is in the presence of uncertainty.

However, stochastic MPC methods typically have high computational demands. This motivates investigating how to solve the optimization problem in a real-time feasible fashion. How to incorporate approximate GP methods is also of interest to improve scalability and computation time.

1.2 Objectives

The main objective of this work is to explore GP-based predictive control for trajectory tracking applications of robot manipulator arms. This includes investigating performance both for fixed-base manipulator systems and floating-base systems. Furthermore, the MPC approach using a GP disturbance model, from here on referred to as GP-MPC, will be compared to the more straightforward approach of Nonlinear Model Predictive Control (NMPC) using the nonlinear prior dynamics model directly. The work aims to investigate trajectory tracking with these methods with both task space and joint space trajectories. The aforementioned objectives can be concretized in the following research questions:

- How may a GP disturbance model be included in a real-time trajectory tracking NMPC controller for robot manipulators with a joint space or task space cost function?

- How does GP-MPC compare to deterministic NMPC using only the prior model for fixed-base robot manipulators, both in simulation tests and in lab tests?

- How may GP-MPC be applied to free-floating manipulators, specifically space manipulator systems, and how does it compare to deterministic NMPC?

1.3 Contribution

The GP-MPC and NMPC methods were implemented and tested to investigate the previously stated research questions. The main contributions are a task space cost formulation, in addition to the joint space formulation, only the latter of which is covered in Carron et al. 2019. Furthermore, the controllers were tested on a free-floating space manipulator system and a fixed-base 6 degrees of freedom (DOF) UR10e robot manipulator arm in simulation. To the best of the author's knowledge, GP-MPC has not been considered with task space trajectory tracking costs for robot manipulators nor the application of space manipulators. Results are presented from lab experiments of the proposed methods on a UR10e robot as well.

The trajectory tracking MPC methods were implemented for real-time control using the Real-time iteration (RTI) scheme for sequential quadratic programming (SQP). Finally, the work also considers multiple sparse GP approximation methods for efficient computation, primarily the Sparse Variational Gaussian Process (SVGP) method introduced in Hensman et al. 2013, which builds upon the Variational Free Energy (VFE) method in Titsias 2009. To the best of the author's knowledge, the SVGP method has not been applied to GP-based MPC methods in the literature.

1.4 Outline

The outline of the remainder of this text is as follows: in Chapter 2 the necessary background theory on kinematics and dynamics for robot manipulators is given. Chapter 3 gives an overview of optimal control theory and goes in further detail on SQP and the RTI scheme. In Chapter 4 GPR is introduced, as well as sparse GP methods. Finally, GP-based MPC is formulated, which combines stochastic MPC with a model learned using GPR. Chapter 5 presents the specific trajectory tracking MPC problems considered in this work. In addition, it covers modeling and the control strategies for the space manipulator system, as well as the software tools used to implement the discussed methods. Results from simulations and lab experiments are presented in Chapter 6, consisting of an illustrative toy example with a planar arm, simulations and lab results with a UR10e robot, and simulation results with a free-floating space manipulator arm. The results are discussed, and directions for further work are suggested in Chapter 7. Chapter 8 contains some

concluding remarks. Finally, some supplementary material is given in the appendices. These include additional formulas for converting between different rotation formulations in Appendix A and for manipulating Gaussian distributions in Appendix B. Modeling of the planar arm is presented in Appendix C for completeness, and supplementary results from UR10e lab tests and space manipulator simulations are given in Appendix D.

2 Kinematics and dynamics of robot manipulator arms[†]

The foundation of control design, motion planning, and simulation for robot manipulator arms starts with how the motion of robots are described. This is done by formulating the kinematics (motion as a purely geometric concept) and the dynamics (motion as a consequence of applied forces and torques). This chapter will start by covering the necessary formalisms for how to represent the position and orientation of rigid bodies in space. Then the kinematics of robot manipulators will be formulated using the Denavit-Hartenberg (DH) convention, and the Jacobian will be introduced to formulate the differential kinematics. Finally, the dynamics of the system will be considered briefly.

2.1 Rigid body rotation representations

When expressing the state of a rigid body in space, it is uniquely defined by its position and orientation, collectively referred to as its pose, relative to some reference frame. Its position can trivially be described by the vector $\boldsymbol{p} \in \mathbb{R}^3$. How a rigid body's orientation is represented is not quite as trivial. This section will, therefore, cover the different rotation representations used throughout this work. The reader is referred to Siciliano et al. 2010, Fossen 2011 and Solà 2017 for further details on rigid body rotations.

2.1.1 Rotation matrices

When describing the rotation of a rigid body in some reference frame, a possible starting point is to look at how the axes of the reference frame are transformed under a rotation.

[†] This chapter is adapted from Brandt 2020.

Given the unit vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ and the rotated unit vectors $\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}'$, the corresponding rotation of an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^3$ can be defined 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 termed the rotation matrix and $SO(3)$ is the special orthogonal group in three dimensions, containing all rotations around the origin. When $SO(3)$ is parametrized using rotation matrices, its properties can be defined by $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}_3$ and $\det(\boldsymbol{R}) = 1$, where $\det(\boldsymbol{R})$ is the determinant of $\boldsymbol{R}$. Note that a sequence of rotations is given by multiplication, such that $\boldsymbol{R}_0^2 = \boldsymbol{R}_0^1 \boldsymbol{R}_1^2$ 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.

2.1.2 Angle-axis representation

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

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

where $[\boldsymbol{u}]_\times$ denotes the skew-symmetric matrix of $\boldsymbol{u}$

$$[\boldsymbol{u}]_\times = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}. \tag{2.3}$$

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, rotation vector, and their relation to the rotation matrix are visualized in Figure 2.1.

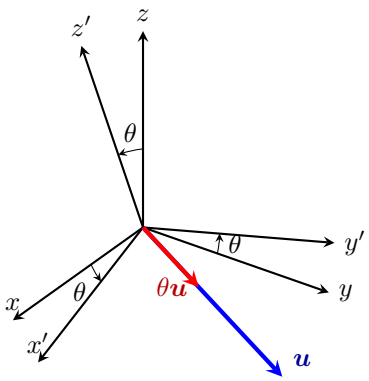


Figure 2.1: Relation between the rotation matrix defined by x', y' and z', the angle-axis representation given by $\boldsymbol{u}$ and θ and the rotation vector $\theta\boldsymbol{u}$.

2.1.3 Euler angles

Another possible minimal representation is to consider three consecutive rotations, each around one of the coordinate axes. The Euler angles $\boldsymbol{\Theta} = [\,\varphi\ \theta\ \psi\,]^\top$ can then be defined, 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 2011 and Siciliano et al. 2010.

2.1.4 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.

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

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \boldsymbol{\varepsilon} \end{bmatrix}, \tag{2.4}$$

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

The multiplication of two quaternions $\boldsymbol{q}_1 = [\,\eta_1 \;\; \boldsymbol{\varepsilon}_1^\top\,]^\top$ and $\boldsymbol{q}_2 = [\,\eta_2 \;\; \boldsymbol{\varepsilon}_2^\top\,]^\top$ is defined by

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \begin{bmatrix} \eta_1\eta_2 - \boldsymbol{\varepsilon}_1^\top \boldsymbol{\varepsilon}_2 \\ \eta_2\boldsymbol{\varepsilon}_1 + \eta_1\boldsymbol{\varepsilon}_2 + [\boldsymbol{\varepsilon}_1]_\times \boldsymbol{\varepsilon}_2 \end{bmatrix}. \tag{2.5}$$

The inverse of a quaternion is defined as

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|^2}, \tag{2.6}$$

where $\|\boldsymbol{q}\| = \sqrt{\eta^2 + \boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}}$ is the quaternion norm and

$$\boldsymbol{q}^* = \begin{bmatrix} \eta \\ -\boldsymbol{\varepsilon} \end{bmatrix} \tag{2.7}$$

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

$$\boldsymbol{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \boldsymbol{u}\sin\frac{\theta}{2} \end{bmatrix}, \tag{2.8}$$

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 quaternion $\boldsymbol{q}$ is given by

$$\boldsymbol{x}' = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \otimes \boldsymbol{q}^*. \tag{2.9}$$

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

$$\boldsymbol{R} = (\eta - \boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon})\boldsymbol{I}_3 + 2\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top + 2\eta[\boldsymbol{\varepsilon}]_\times. \tag{2.10}$$

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 $-\boldsymbol{q}$ represent the same rotation. While the representation is not unique, it is however singularity-free.

A composition of several rotations, similarly to rotation matrices, is given by the product of these unit quaternions, i.e., $\boldsymbol{q}_{AC} = \boldsymbol{q}_{AB} \otimes \boldsymbol{q}_{BC}$. The unit quaternion error can then be defined as

$$\delta\boldsymbol{q} = \boldsymbol{q}_1^* \otimes \boldsymbol{q}_2, \tag{2.11}$$

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

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

$$e^{\boldsymbol{q}} = e^{\eta} \begin{bmatrix} \cos\|\boldsymbol{\varepsilon}\| \\ \frac{\boldsymbol{\varepsilon}}{\|\boldsymbol{\varepsilon}\|}\sin\|\boldsymbol{\varepsilon}\| \end{bmatrix}, \tag{2.12}$$

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

$$\log\boldsymbol{q} = \begin{bmatrix} 0 \\ \theta\boldsymbol{u} \end{bmatrix}, \tag{2.13}$$

where $\boldsymbol{u}$ and θ are defined as in Section 2.1.2. 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.

Finally, using the preceding definitions the unit quaternion power can be defined as

$$\boldsymbol{q}^s = \exp(s\log\boldsymbol{q}) = \begin{bmatrix} \cos s\theta \\ \boldsymbol{u}\sin s\theta \end{bmatrix}, \tag{2.14}$$

which creates the basis for interpolation between quaternions. This will be discussed further in Section 5.2.2.

2.2 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. A robot manipulator arm with n DOF, i.e., n joints, is depicted in Figure 2.2. One end of the chain is connected to a fixed 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. Let $\boldsymbol{k}_i$ denote the unit norm joint axis of the i-th joint, and $\boldsymbol{p}_i$ be the translation to the i-th joint. The states of the system can be defines as $\boldsymbol{x} \in \mathbb{R}^{2n}$, consisting of the joint angles $\boldsymbol{q} \in \mathbb{R}^n$ and the joint angular velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, i.e.,

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}. \tag{2.15}$$

The space spanned by $\boldsymbol{q}$ is denoted as the joint space.

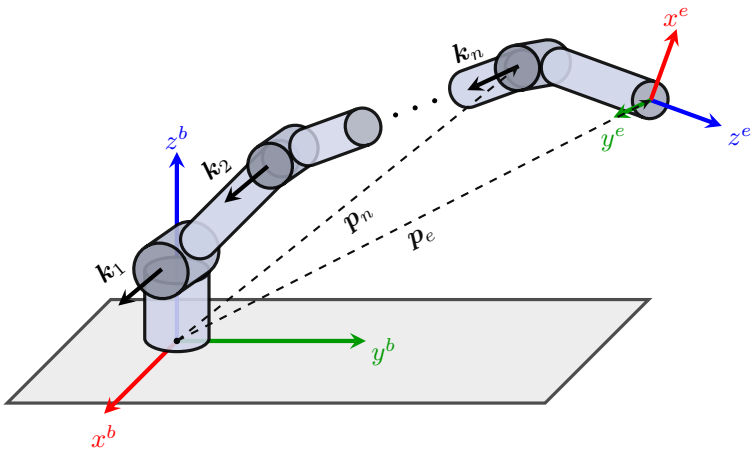


Figure 2.2: Robot manipulator arm with n DOF. The base frame and end effector frame are indicated, as well as the joint axes $\boldsymbol{k}_i$ and the position vectors $\boldsymbol{p}_i$.

However, when interacting with the environment, the tasks for the robot are usually expressed in terms of the desired end effector pose $\boldsymbol{r}^e(t)$, not a desired joint configuration. The end effector position is denoted by $\boldsymbol{p}_e$ and the end effector orientation is denoted

as the unit quaternion $\boldsymbol{Q}_e$. The end effector pose space, spanned by the end effector pose $\boldsymbol{h}_e = [\, \boldsymbol{p}_e^\top \;\; \boldsymbol{Q}_e^\top \,]^\top$, is therefore known as the task space or operational 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 termed the forward kinematics. In the following, the forward kinematics equations for robot manipulator arms will be formulated using homogeneous transformation matrices.

2.2.1 Homogeneous transformation matrices

Motivated by the goal of finding a mapping from the joint space to the task space, it is noted 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.

As discussed in Siciliano et al. 2010, one first extends the representation of a point $\boldsymbol{p} \in \mathbb{R}^3$ by appending a unit element, i.e.,

$$\widetilde{\boldsymbol{p}} = \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix}, \tag{2.16}$$

which is termed homogeneous coordinates. The homogeneous transformation matrix can then be defined as

$$\boldsymbol{T}_1^0 = \begin{bmatrix} \boldsymbol{R}_1^0 & \boldsymbol{r}_1^0 \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in SE(3), \tag{2.17}$$

which transforms a homogeneous coordinate vector $\widetilde{\boldsymbol{p}}$ from frame 1 to frame 0 defined by the translation vector $\boldsymbol{r}_1^0 \in \mathbb{R}^3$ and the rotation vector $\boldsymbol{R}_1^0 \in SO(3)$. $SE(3)$ is termed the special Euclidean group, and consists of all rigid body transformations. The transformation is then written as

$$\widetilde{\boldsymbol{p}}^0 = \boldsymbol{T}_1^0 \, \widetilde{\boldsymbol{p}}^1. \tag{2.18}$$

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

$$\widetilde{\boldsymbol{p}}^0 = \boldsymbol{T}_1^0 \, \boldsymbol{T}_2^1 \, \cdots \, \boldsymbol{T}_n^{n-1} \, \widetilde{\boldsymbol{p}}^n. \tag{2.19}$$

It is then straightforward to formulate the forward kinematics of the end effector using a series of homogeneous transformations. Firstly, a fixed coordinate frame is defined in every link. The sequence of homogeneous transformations from the end effector frame, from every link to its parent, to the base frame, can then be considered. The homogeneous transformation matrix describing the pose of the end effector with respect to the base frame is then given as:

$$\boldsymbol{T}_e^b(\boldsymbol{q}) = \boldsymbol{T}_0^b\, \boldsymbol{T}_1^0(q_1)\, \boldsymbol{T}_2^1(q_2)\, \cdots\, \boldsymbol{T}_n^{n-1}(q_n)\, \boldsymbol{T}_e^n, \tag{2.20}$$

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

2.2.2 The Denavit-Hartenberg convention

It has been seen how the pose of the end effector is related to the base frame with homogeneous transformations. However, how these homogeneous transformation matrices are formulated, and therefore also the end effector pose, using the joint angles, are yet to be seen. This task comes down to how to define the link frames and the transformations between them. The DH convention formulates a systematic way of expressing these transformations for any robot manipulator.

The transform from frame $i+1$ to frame i, for $i \in \mathbb{N}_{n-1} = \{1, 2, \ldots, n-1\}$, following the DH convention, is defined as

$$\begin{aligned} \boldsymbol{T}_{i+1}^i &= \begin{bmatrix} \boldsymbol{R}_z(\theta_i) & d_i \boldsymbol{e}_z \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_x(\alpha_i) & a_i \boldsymbol{e}_x \\ \boldsymbol{0}^T & 1 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{R}_z(\theta_i)\, \boldsymbol{R}_x(\alpha_i) & a_i \boldsymbol{R}_z(\theta_i)\, \boldsymbol{e}_x + d_i \boldsymbol{e}_z \\ \boldsymbol{0}^T & 1 \end{bmatrix}, \end{aligned} \tag{2.21}$$

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, the forward kinematics of the end effector can then be defined as a 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.

2.3 Differential kinematics

In this section the Jacobian will be introduced, which provides the mapping from joint velocities to end effector velocity, termed the differential kinematics. This is analogous to how the kinematics gives the mapping from joint angles to end effector pose. As discussed in Siciliano et al. 2010, the end effector velocity $\boldsymbol{\nu}_e \in \mathbb{R}^6$ is related to the joint angular velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ by the linear map given by the geometric Jacobian $\boldsymbol{J}(\boldsymbol{q})$:

$$\boldsymbol{\nu}_e = \begin{bmatrix} \boldsymbol{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{2.22}$$

where $\boldsymbol{v}_e \in \mathbb{R}^3$ and $\boldsymbol{\omega}_e \in \mathbb{R}^3$ are the linear and angular velocity of the end effector, $\boldsymbol{J} = [\, \boldsymbol{J}_T^\top \;\; \boldsymbol{J}_R^\top \,]^\top \in \mathbb{R}^{6\times n}$, and $\boldsymbol{J}_T \in \mathbb{R}^{3\times n}$ is the Jacobian, and $\boldsymbol{J}_R \in \mathbb{R}^{3\times n}$ are the translational and rotational components of the Jacobian respectively. Note that the general case of three spatial dimensions is assumed here.

It is shown in Siciliano et al. 2010 that the geometric Jacobian for a general fixed-base robot manipulator arm is

$$\boldsymbol{J} = \begin{bmatrix} [\boldsymbol{k}_1]_\times(\boldsymbol{p}_e - \boldsymbol{p}_1) & \cdots & [\boldsymbol{k}_n]_\times(\boldsymbol{p}_e - \boldsymbol{p}_n) \\ \boldsymbol{k}_1 & \cdots & \boldsymbol{k}_n \end{bmatrix}. \tag{2.23}$$

It is assumed that all joints are revolute, i.e., rotational, as opposed to prismatic joints, which are translational.

The inverse problem, i.e., finding the required joint velocities needed to reach a desired end effector velocity, is also of interest, which is termed inverse differential kinematics. This can be achieved by inversion of the Jacobian, or more generally by the pseudoinverse of the Jacobian:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^\dagger(\boldsymbol{q})\boldsymbol{\nu}_e, \tag{2.24}$$

where $\boldsymbol{J}^\dagger(\boldsymbol{q}) = \boldsymbol{J}^\top(\boldsymbol{q})(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^\top(\boldsymbol{q}))^{-1}$ is the pseudoinverse. In practice a damped

pseudoinverse is often used to avoid singular configurations:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^\top(\boldsymbol{q}) \left(\boldsymbol{J}(\boldsymbol{q}) \boldsymbol{J}^\top(\boldsymbol{q}) + \lambda^2 \boldsymbol{I}_6 \right)^{-1} \boldsymbol{\nu}_e, \tag{2.25}$$

with λ being a damping parameter.

2.4 Manipulator dynamics

For control and simulation purposes, it is useful to study not only the kinematics of the manipulator but also the dynamics. As introduced in Section 2.2, 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 $\boldsymbol{\tau}$. The general manipulator equations of motion can then be derived as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v \dot{\boldsymbol{q}} + \boldsymbol{F}_s \, \mathbf{sgn}(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{J}^\top(\boldsymbol{q}) \boldsymbol{h}_e. \tag{2.26}$$

Here $\boldsymbol{M}$ is the square, symmetric and positive-definite mass matrix, $\boldsymbol{C}$ is the Coriolis and centripetal force matrix, $\boldsymbol{g}$ is the gravity term, and $\boldsymbol{F}_v \in \mathbb{R}^{n \times n}$ and $\boldsymbol{F}_s \in \mathbb{R}^{n \times n}$ are diagonal matrices of viscous and static (Coulomb) friction coefficients respectively. The rightmost term in Eq. (2.26) gives the joint torques from contact forces and torques on the end effector, where $\boldsymbol{h}_e$ denotes the forces and torques from the end effector on the environment. Finally $\mathbf{sgn}(\cdot)$ denotes the vector of element-wise sign functions.

The equations of motion can be derived, e.g., by Lagrange's equation or using a Newton-Euler approach such as the recursive Newton-Euler algorithm. The specific formulation of $\boldsymbol{M}(\boldsymbol{q})$ and $\boldsymbol{C}(\boldsymbol{q})$ are given in Siciliano et al. 2010. The formulation in Eq. (2.26) allows for the computation of required torques for a given desired motion and is referred to as the inverse dynamics.

From here on it is assumed that $\boldsymbol{F}_v = \boldsymbol{F}_s = \boldsymbol{0}$ and $\boldsymbol{h}_e = \boldsymbol{0}$ in the models used for control and planning, i.e., static and viscous friction terms are neglected and the robot does not exert any forces or torques on its environment. The resulting simplified inverse dynamics model is then

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}. \tag{2.27}$$

By rewriting the equations of motion in terms of the state vector $\boldsymbol{x} = [\, \boldsymbol{q}^\top \;\; \dot{\boldsymbol{q}}^\top \,]^\top =$

$[\,\boldsymbol{x}_1^\top \;\; \boldsymbol{x}_2^\top\,]^\top$ the forward dynamics

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\tau}) = \begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{M}(\boldsymbol{x}_1)^{-1}(\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{x}_1, \boldsymbol{x}_2) - \boldsymbol{g}(\boldsymbol{x}_1)) \end{bmatrix} \tag{2.28}$$

are obtained. They can be derived by explicitly calculating the inverse of the inertia matrix or using a recursive forward dynamics algorithm such as the articulated body algorithm or the composite rigid body algorithm, as detailed in Featherstone 2008.

3 Optimal control[†]

Optimal control theory considers 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 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 optimal control policy. The branch of numerical optimal control is concerned with how to apply numerical methods to approximately solve such problems. This is of practical interest, as it has the potential to solve complex optimal control problem (OCP)s in real-time. This chapter will give a brief introduction to optimal control, NMPC and direct numerical optimal control methods. Additionally, the SQP method for solving nonlinear program (NLP)s will be introduced, and the RTI scheme used for solving NLPs in real-time in this work will be discussed.

3.1 Optimal control and model predictive control

This section will start by introducing the general infinite-horizon OCP for continuous-time systems. It will then examine more computationally tractable formulations by discretizing the problem and using the finite horizon approximation. Finally, the concept of closing the loop with MPC will be discussed. The section is based on Rawlings et al. 2019, Grüne and Pannek 2011 and Johansen 2011.

[†] This chapter is adapted from Brandt 2020.

Firstly, the continuous-time infinite horizon OCP is given as:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \int_0^\infty \ell(\boldsymbol{x},\boldsymbol{u})\,dt \\ \text{s.t.} \quad & \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}), \\ & \boldsymbol{x}(t) \in \mathcal{X}, \quad \forall t, \\ & \boldsymbol{u}(t) \in \mathcal{U}, \quad \forall t, \\ & \boldsymbol{x}(0) = \boldsymbol{x}_0, \end{aligned} \tag{3.1}$$

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. $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and $\boldsymbol{u} \in \mathbb{R}^{n_u}$ denote the time-varying state and control input, respectively. 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 may also be considered.

It is desirable to solve Eq. (3.1) analytically and apply the optimal control input $\boldsymbol{u}^\star(t)$ to the dynamical system. However, finding the exact solution is rarely computationally tractable for a nonlinear OCP. Discretization is therefore necessary, and here a distinction is made between indirect and direct methods. Indirect methods are concerned with how to solve the continuous OCP in Eq. (3.1) and then discretize the solution, while direct methods first discretize the problem and then solve it using numerical optimization. In the following, direct methods will be explored further, as they provide methods that are computationally efficient and easily applicable to a wide range of problems.

The discretized system dynamics $\boldsymbol{x}_{i+1} = \boldsymbol{f}_d(\boldsymbol{x}_i,\boldsymbol{u}_i)$ will now be considered, with a constant time step, also referred to as sample time, of t_s. How the discretized dynamics function $\boldsymbol{f}_d$ is obtained will be discussed in further detail in Section 3.2. Furthermore, the cost function will now be a sum over the time steps instead of an integral. The resulting

optimization problem, termed the discrete-time infinite horizon OCP, can be written as

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \sum_{i=0}^{\infty} \ell_d(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{3.2a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{i+1} = \boldsymbol{f}_d(\boldsymbol{x}_i, \boldsymbol{u}_i), \quad i = 0, \dots, N-1, \tag{3.2b}$$

$$\boldsymbol{x}_i \in \mathcal{X}, \quad i = 0, \dots, N, \tag{3.2c}$$

$$\boldsymbol{u}_i \in \mathcal{U}, \quad i = 0, \dots, N-1, \tag{3.2d}$$

$$\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0, \tag{3.2e}$$

where ℓ_d is the discretized stage cost and $\bar{\boldsymbol{x}}_0$ is the initial condition. The optimization variables are the vector of stacked state variables $\boldsymbol{x} = [\, \boldsymbol{x}_0^\top \ \cdots \ \boldsymbol{x}_N^\top \,]^\top$ and the vector of stacked input variables $\boldsymbol{u} = [\, \boldsymbol{u}_0^\top \ \cdots \ \boldsymbol{u}_{N-1}^\top \,]^\top$. This problem is, however, still computationally intractable for most combinations of stage costs and dynamics, as a consequence of the infinite horizon length.

Yet the simple case with linear time-invariant dynamics

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i, \tag{3.3}$$

quadratic stage cost

$$\ell_d(\boldsymbol{x}_i, \boldsymbol{u}_i) = \|\boldsymbol{x}_i\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2, \tag{3.4}$$

and no state and input constraints, i.e., $\mathcal{X} = \mathbb{R}^{n_x}$ and $\mathcal{U} = \mathbb{R}^{n_u}$, will briefly be discussed. Here $\boldsymbol{Q} \succ 0$ and $\boldsymbol{R} \succ 0$ are the state weight matrix and input weight matrix, respectively. Under these assumptions an analytic solution exists, termed the Linear Quadratic Regulator (LQR), given by the control law

$$\boldsymbol{u}_i = -\boldsymbol{K}\boldsymbol{x}_i, \tag{3.5}$$

where the LQR gain $\boldsymbol{K}$ is given by

$$\boldsymbol{K} = \left(\boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{B} + \boldsymbol{R}\right)^{-1} \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{A} \tag{3.6}$$

and $\boldsymbol{P}$ is given by the positive definite solution of the discrete algebraic Ricatti equation:

$$\boldsymbol{P} = \boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{P} \boldsymbol{A} - \boldsymbol{A}^T \boldsymbol{P} \boldsymbol{B} \left(\boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{B} + \boldsymbol{R}\right)^{-1} \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{A}. \tag{3.7}$$

For more involved discrete-time OCPs, the infinite-horizon solution is often intractable, as it is infinite-dimensional. In the following the study will, therefore, be restricted to the discrete-time finite horizon OCP, with a finite horizon length T, consisting of N steps of length t_s. Considering general nonlinear costs and dynamics, the following NLP is obtained:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \sum_{i=0}^{N-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_N) \tag{3.8a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{i+1} = \boldsymbol{f}_d(\boldsymbol{x}_i, \boldsymbol{u}_i), \quad i = 0, \dots, N-1, \tag{3.8b}$$

$$\boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq 0, \quad i = 0, \dots, N-1, \tag{3.8c}$$

$$\boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = 0, \quad i = 0, \dots, N-1, \tag{3.8d}$$

$$\boldsymbol{g}_N(\boldsymbol{x}_N) \leq 0, \tag{3.8e}$$

$$\boldsymbol{h}_N(\boldsymbol{x}_N) = 0, \tag{3.8f}$$

$$\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0. \tag{3.8g}$$

Here the final stage cost $\ell_f(\boldsymbol{x}_N)$ is introduced, in order to approximate the remaining cost from N to infinity. Furthermore, the state and input constraints are formulated more explicitly as a set of nonlinear inequality and equality constraints with $\boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i)$ and $\boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i)$.

After formulating the discrete-time finite-horizon OCP in Eq. (3.8), the NMPC algorithm can be formulated, as given in Algorithm 1. By solving the OCP Eq. (3.8) at every time step with the current sampled state and applying the first control input from the solution, one effectively closes the loop.

This concept is illustrated in Figure 3.1. MPC introduces robustness against modeling errors and noise when compared to applying the optimal control sequence $\boldsymbol{u}^\star$ directly in open loop. Nevertheless, this comes at the cost of high computational demands, as the optimization problem must be solved at every time step. Theoretical details on NMPC, such as stability, feasibility, and robustness, will not be discussed here. See Grüne and Pannek 2011 and Johansen 2011 for further details on NMPC theory.

Algorithm 1 NMPC

1: **For** every time step t_k:
2: Sample the current state $\boldsymbol{x}(t_k)$ of the system.
3: Solve Eq. (3.8) using $\bar{\boldsymbol{x}}_0 = \boldsymbol{x}(t_k)$, and denote the solution as $\boldsymbol{x}^\star, \boldsymbol{u}^\star$.
4: Let $\boldsymbol{u}_0^\star$ be the control input at the current time step.

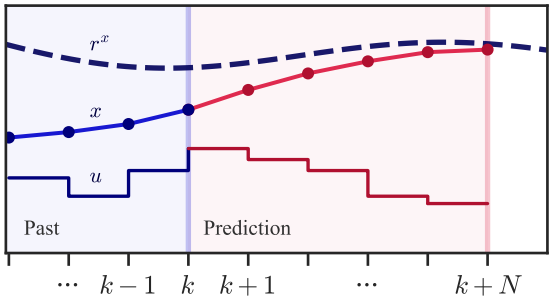


Figure 3.1: Visualization of MPC. The state x is controlled using the control input u in order to track the reference trajectory r^x, shown both for the previous time steps in closed loop and for the current prediction horizon.

3.2 Direct 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 and Rawlings et al. 2019. Starting with direct single shooting, the basic idea is to eliminate the state variable $\boldsymbol{x}$ by recursively substituting the discretized system dynamics $\boldsymbol{f}_d(\boldsymbol{x}_i, \boldsymbol{u}_i)$ for $\boldsymbol{x}_{i+1}$. One then obtains 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

$$\begin{aligned}
\boldsymbol{y}_{k+1} &= \boldsymbol{y}_k + \frac{1}{6}\left(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\right), \quad \text{where} \\
\boldsymbol{k}_1 &= h\boldsymbol{f}\left(t_k, \boldsymbol{y}_k\right), \\
\boldsymbol{k}_2 &= h\boldsymbol{f}\left(t_k + \frac{h}{2}, \boldsymbol{y}_k + \frac{\boldsymbol{k}_1}{2}\right), \\
\boldsymbol{k}_3 &= h\boldsymbol{f}\left(t_k + \frac{h}{2}, \boldsymbol{y}_k + \frac{\boldsymbol{k}_2}{2}\right), \\
\boldsymbol{k}_4 &= h\boldsymbol{f}\left(t_k + h, \boldsymbol{y}_k + \boldsymbol{k}_3\right),
\end{aligned} \tag{3.9}$$

for $k \in \mathbb{N}^0 = \{0, 1, 2, \dots\}$, which is a method that will be frequently used in later chapters.

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, the direct shooting method is applied, and an additional constraint is added to ensure continuity between every interval. This means additional variables have to be added for the artificial initial condition at the start of every interval. However, the resulting NLP is sparser and often converges faster, as stated in Rawlings et al. 2019.

Finally, with direct collocation, the general idea is to approximate the state and input as piecewise polynomials, where the state trajectory is parametrized by a number of knot points between every time step. While shooting methods typically assume piecewise constant control inputs between every time step, this approach allows to approximate the control input as a polynomial between every step. This leads to an even sparser NLP than multiple shooting, but evidently with even more optimization variables.

3.3 Sequential quadratic programming

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

$$\min_{\boldsymbol{z}} \boldsymbol{f}(\boldsymbol{z}) \quad \text{s.t.} \quad \boldsymbol{g}(\boldsymbol{z}) \leq 0,\ \boldsymbol{h}(\boldsymbol{z}) = 0 \tag{3.10}$$

at every time step. In the following, the SQP method is presented, based on Nocedal and Wright 2006 and Diehl et al. 2009. The reader should, however, be aware that many other numerical methods exist, of which interior point methods are also frequently used for numerical optimal control, which are discussed in detail in the aforementioned references.

In order to formulate the principle behind SQP, the first-order optimality conditions for Eq. (3.10) are stated, known as the Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} \nabla_{\boldsymbol{z}} \mathcal{L}\left(\boldsymbol{z}^{\star}, \boldsymbol{\lambda}^{\star}, \boldsymbol{\mu}^{\star}\right) &= 0, \\ \boldsymbol{g}\left(\boldsymbol{z}^{\star}\right) &\leq 0, \\ \boldsymbol{h}\left(\boldsymbol{z}^{\star}\right) &= 0, \\ \boldsymbol{\mu}^{\star} &\geq 0, \\ g_i\left(\boldsymbol{z}^{\star}\right) \mu_i^{\star} &= 0, \quad i = 1, \ldots, n_g. \end{aligned} \tag{3.11}$$

Here $\boldsymbol{\mu}$ 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 the optimal solution is denoted by $(\boldsymbol{z}^{\star}, \boldsymbol{\mu}^{\star}, \boldsymbol{\lambda}^{\star})$. Furthermore, the Lagrangian function $\mathcal{L}$ is defined as

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \boldsymbol{f}(\boldsymbol{z}) + \boldsymbol{\lambda}^{\top} \boldsymbol{h}\left(\boldsymbol{z}\right) + \boldsymbol{\mu}^{\top} \boldsymbol{g}\left(\boldsymbol{z}\right). \tag{3.12}$$

The SQP method is based on solving the KKT system Eq. (3.11) using Newton's method for nonlinear systems, i.e., iteratively solving the linearized system of equations. It can be shown that this is equivalent to iteratively solving the following quadratic program (QP):

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \nabla_{\boldsymbol{z}} \boldsymbol{f}^{\top}(\boldsymbol{z}_k)(\boldsymbol{z} - \boldsymbol{z}_k) + \frac{1}{2}(\boldsymbol{z} - \boldsymbol{z}_k)^{\top} \nabla_{\boldsymbol{z}}^2 \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)(\boldsymbol{z} - \boldsymbol{z}_k) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{z}_k) + \nabla_{\boldsymbol{z}} \boldsymbol{h}^{\top}(\boldsymbol{z}_k)(\boldsymbol{z} - \boldsymbol{z}_k) = 0, \\ & \boldsymbol{g}(\boldsymbol{z}_k) + \nabla_{\boldsymbol{z}} \boldsymbol{g}^{\top}(\boldsymbol{z}_k)(\boldsymbol{z} - \boldsymbol{z}_k) \leq 0, \end{aligned} \tag{3.13}$$

where $\boldsymbol{z}_k$ is the current iterate of the approximation of the solution $\boldsymbol{z}^{\star}$. One such iteration is termed a Newton step, as it corresponds to a single step of Newton's method.

The Hessian $\nabla_{\boldsymbol{z}}^2 \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)$ typically requires high computational effort to compute exactly, and it must be positive semidefinite for the problem Eq. (3.13) to be convex, which is not guaranteed for $\boldsymbol{z}$ far away from the optimum. The Gauss-Newton method

is one possible solution to these limitations. Under the special case of a sum of squares objective function, i.e.,

$$\boldsymbol{f}(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2, \tag{3.14}$$

the Hessian is approximated by

$$\nabla^2_{\boldsymbol{z}}\mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \approx \boldsymbol{H} = \nabla\boldsymbol{r}(\boldsymbol{x})\nabla\boldsymbol{r}(\boldsymbol{x})^\top, \tag{3.15}$$

which provides a more tractable expression. The reader is referred to Nocedal and Wright 2006 for further details on SQP and Gauss-Newton. However, one aspect of SQP implementations which is of vital importance to NMPC specifically are warm starts, which will be considered in further detail in the following section.

3.4 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 close to the solution at the current time step. When doing this shifting procedure to warm start the solver, it can, therefore, be assumed 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. This 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. Note that the Gauss-Newton Hessian approximation given in Eq. (3.15) is usually applied 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 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 $\boldsymbol{u}_0^\star$ can be applied to the system. This feedback delay results in the solver using outdated information of the system state and can reduce stability and performance of the closed

loop system. Gros et al. 2016 refers to this issue as the real-time dilemma.

The consequences of the real-time dilemma are mitigated with the RTI scheme 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. Then in the feedback phase, the QP is fully formed and solved. Since all the computations 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 the preparation step. The basic loop of the preparation and feedback phase is visualized in Figure 3.2, which illustrates how the RTI scheme minimizes the feedback delay. Since, in practice, the preparation phase typically takes longer than the feedback phase, the feedback delay can be drastically reduced, as mentioned in Diehl et al. 2009. The SQP RTI is shown in Algorithm 2.

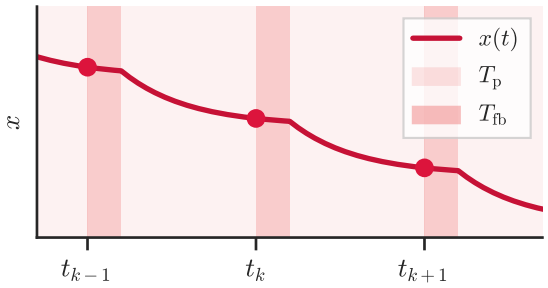


Figure 3.2: RTI loop, with T_p and T_fb denoting the preparation phase and feedback phase respectively.

Algorithm 2 SQP RTI

Preparation phase:

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

2: Evaluate the terms in the QP Eq. (3.13), 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 $\Delta\boldsymbol{x}^\star_k, \Delta\boldsymbol{u}^\star_k$.

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

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

4 Gaussian process regression and Gaussian process-based MPC

The following chapter will introduce GPR, a Bayesian nonlinear regression method, and explore how to include a GP model in a stochastic MPC problem. Firstly, Bayesian linear regression will be briefly discussed as an introduction to Bayesian inference. This framework will then be applied to GPs, resulting in GP regression. Subsequently, sparse GP approximation methods will be explored, motivated by the need for computationally efficient GP methods. Finally, GP models will be applied in an optimal control context to formulate GP-based MPC, abbreviated to GP-MPC in the following.

4.1 Bayesian linear regression

The probabilistic regression problem can be formulated as follows: given a training set $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ of a matrix of vector inputs $\boldsymbol{X} = [\, \boldsymbol{x}_1 \;\; \cdots \;\; \boldsymbol{x}_M \,]^\top \in \mathbb{R}^{M \times n}$, with M realizations of $\boldsymbol{x} \in \mathbb{R}^n$, and a vector of noisy scalar outputs $\boldsymbol{y} = [\, y_1 \;\; \cdots \;\; y_M \,]^\top \in \mathbb{R}^M$, which are assumed to be realizations of the underlying function $f(\boldsymbol{x})$:

$$y = f(\boldsymbol{x}) + \varepsilon, \tag{4.1}$$

one wants to predict the distribution of f at new test points $\boldsymbol{x}^*$, denoted f^*. In the following it is assumed that the noise ε is independent and identically distributed (i.i.d.) zero mean Gaussian distributed, i.e.,

$$\varepsilon \sim \mathcal{N}\left(0, \sigma_n^2\right), \tag{4.2}$$

where σ_n^2 is the noise variance.

In the following, the Bayesian linear regression approach to this problem is introduced, which will then be generalized with GPR. The section is based on Tipping 2004 and Rasmussen and Williams 2006, and the reader is referred to these for further details. In Bayesian linear regression it is assumed that the underlying function f is linear in the parameters $\boldsymbol{w}$:

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}. \tag{4.3}$$

The fundamental principle behind the Bayesian framework is that the parameters $\boldsymbol{w}$ are assumed to be random variables as well as $\boldsymbol{x}$ and $\boldsymbol{y}$. One therefore needs to specify a prior over the parameters, which for linear Bayesian regression is a zero mean multivariate Gaussian distribution:

$$\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}), \tag{4.4}$$

with $\boldsymbol{\Sigma}$ being the covariance matrix.

In order to make predictions about the posterior distribution $p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{X})$, Bayes' theorem

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}) p(\boldsymbol{w})}{p(\boldsymbol{y} \mid \boldsymbol{X})}, \tag{4.5}$$

can be applied, where the likelihood $p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X})$ is given by

$$p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{X}\boldsymbol{w}, \sigma_n^2 \boldsymbol{I}\right), \tag{4.6}$$

with $\boldsymbol{I}$ denoting the identity matrix of appropriate size. The normalization constant $p(\boldsymbol{y} \mid \boldsymbol{X})$ is given by

$$p(\boldsymbol{y} \mid \boldsymbol{X}) = \int p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w}) \mathrm{d}\boldsymbol{w} \tag{4.7}$$

and can effectively be ignored as it is independent of $\boldsymbol{w}$. The posterior can then be evaluated as

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(\sigma_n^{-2} \boldsymbol{A}^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \boldsymbol{A}^{-1}), \tag{4.8}$$

where $\boldsymbol{A} = \sigma_n^{-2} \boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{\Sigma}^{-1}$, as detailed in Rasmussen and Williams 2006. It is seen that using the Bayesian framework by assuming a prior over the weights and using Bayes' theorem allows to infer the distribution of $\boldsymbol{w}$ instead of only learning a single estimated value for $\boldsymbol{w}$. This procedure of inferring the posterior distribution given a prior model and

data can be seen as updating the prior belief using the collected dataset and is at the core of the Bayesian framework.

Finally, the predictive distribution can be obtained by marginalizing the model over the weights to obtain the predictive distribution:

$$p(f^* \mid \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{y}) = \int p(f^* \mid \boldsymbol{x}^*, \boldsymbol{w}) p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \mathrm{d}\boldsymbol{w}, \tag{4.9}$$

which for the linear model has the closed form solution

$$p(f^* \mid \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}\left(\sigma_n^{-2} (\boldsymbol{x}^*)^\top \boldsymbol{A}^{-1} \boldsymbol{X}^\top \boldsymbol{y}, (\boldsymbol{x}^*)^\top \boldsymbol{A}^{-1} \boldsymbol{x}^*\right). \tag{4.10}$$

By assuming a linear model, the expressiveness of the resulting predictions is limited. A remedy for this limitation is to first project the inputs into a high dimensional feature space given by a set of basis functions $\phi : \mathbb{R}^n \mapsto \mathbb{R}^m$, and then perform linear regression in this new space. The new generalized model can be written as

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{w}. \tag{4.11}$$

Let $\boldsymbol{\Phi} = \left[\, \phi_1(\boldsymbol{x}) \;\; \cdots \;\; \phi_m(\boldsymbol{x}) \,\right]^\top \in \mathbb{R}^{M \times m}$ be the matrix of inputs in the new feature space, analogous to $\boldsymbol{X}$. Making the same assumptions as before, one arrives at the same result as Eq. (4.10) with $\boldsymbol{\Phi}$ substituted for $\boldsymbol{X}$:

$$p(f^* \mid \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}\left(\sigma_n^{-2} \phi(\boldsymbol{x}^*)^\top \boldsymbol{A}^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y},\; \phi(\boldsymbol{x}^*)^\top \boldsymbol{A}^{-1} \phi(\boldsymbol{x}^*)\right), \tag{4.12}$$

with $\boldsymbol{A} = \sigma_n^{-2} \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \boldsymbol{\Sigma}^{-1}$. One now needs to invert the $m \times m$ matrix $\boldsymbol{A}$, which for high dimensional feature spaces may be challenging. This is solved by using the so-called kernel trick. Firstly, it is outlined in Rasmussen and Williams 2006 that Eq. (4.12) can be rewritten entirely in terms of what is termed the kernel or covariance function $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top \boldsymbol{\Sigma} \phi(\boldsymbol{x'})$, such that all the terms in feature space can be formulated using k. Furthermore, notice that k is simply an inner product by writing $k(\boldsymbol{x}, \boldsymbol{x}') = \widetilde{\phi}(\boldsymbol{x})^\top \widetilde{\phi}(\boldsymbol{x'})$, with $\widetilde{\phi}(\boldsymbol{x}) = \boldsymbol{\Sigma}^{\frac{1}{2}} \phi(\boldsymbol{x})$, such that

$$\begin{aligned} p(f^* \mid \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}\big(& k(\boldsymbol{x}^*, \boldsymbol{x})(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y}, \\ & k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(\boldsymbol{x}^*, \boldsymbol{x})(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} k(\boldsymbol{x}, \boldsymbol{x}^*)\big)\big), \end{aligned} \tag{4.13}$$

where the notation $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \in \mathbb{R}^{M\times M}$ is introduced. This allows to rewrite the computationally inefficient prediction in Eq. (4.12) in terms of inner products in input space, which in most cases is significantly more efficient. Furthermore, this shifts the focus from the feature space to the kernel function itself and even allows to use infinite-dimensional kernels, which will play a central role in GPR.

4.2 Gaussian process regression

The kernelized Bayesian linear regression in the previous section provides the basis for GPR. For GPR, a GP will be used as a prior model, which will be combined with data to infer the posterior distribution, similarly to the previously discussed Bayesian linear regression. However, before arriving at this point, GPs will first be introduced as a concept to motivate its use in Bayesian regression. The following section is based on Rasmussen and Williams 2006.

4.2.1 Gaussian processes

A GP is a set of random variables for which any finite subset has a Gaussian distribution. It is uniquely defined by its mean function m and covariance function k and is denoted by $\mathcal{GP}(m, k)$. GPs can be considered a generalization of the multivariate Gaussian distribution, which is also uniquely defined by its mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$:

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n \det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{4.14}$$

Whereas the multivariate Gaussian distribution is a distribution over an n-dimensional vector $\boldsymbol{x}$, the GP is a distribution over a function f. The mean function m and covariance function k can therefore be thought of as an infinite-dimensional mean vector and covariance matrix.

To demonstrate this generalization from multivariate Gaussian distribution to GP, a bivariate Gaussian distribution is first considered, shown in Figure 4.1. For higher-dimensional Gaussian distributions, one can instead plot the indices of the input dimensions in relation to a single possible realization of the Gaussian variable, as shown in Figure 4.2.

By increasing the number of dimensions, it is observed how the Gaussian distribution

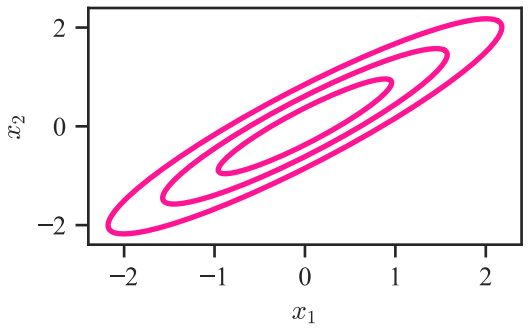


Figure 4.1: 1-sigma, 2-sigma and 3-sigma covariance ellipses for a bivariate Gaussian distribution.

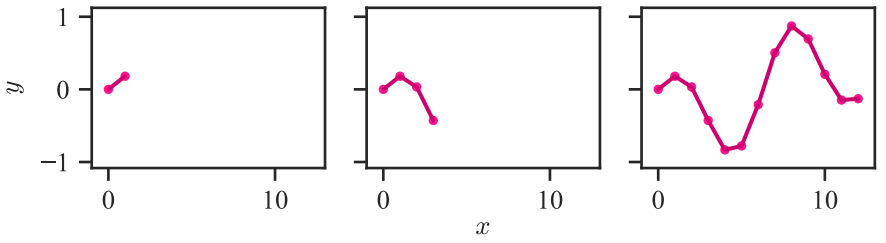


Figure 4.2: Realization of multivariate Gaussian for 2, 4 and 13 dimensions.

may describe a certain shape of functions, described by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Furthermore, taking this concept into the Bayesian context and considering this Gaussian distribution as a prior, one may fix some dimensions of $\boldsymbol{x}$ and calculate the conditional Gaussian on those dimensions, given by Eq. (B.2). Figure 4.3 illustrates this concept, where the conditional mean and covariance then defines our posterior belief given the known dimensions, which in this case are x_0, x_4 and x_7.

Finally, generalizing from a finite-dimensional vector space to function space, one gets a GP. This is shown in Figure 4.4, as a GP conditioned on the same data as before, but now with continuous mean and covariance functions. The variance of the finite-dimensional Gaussian in Figure 4.3 and the infinite-dimensional GP in Figure 4.4 are visualized as 2-sigma limits. A 2-sigma probability refers to the probability given by the area within 2 standard deviations around the mean. This provides a convenient way of quantifying probabilities for Gaussian variables and will be used frequently in this work. The concept

is visualized in Figure 4.5.

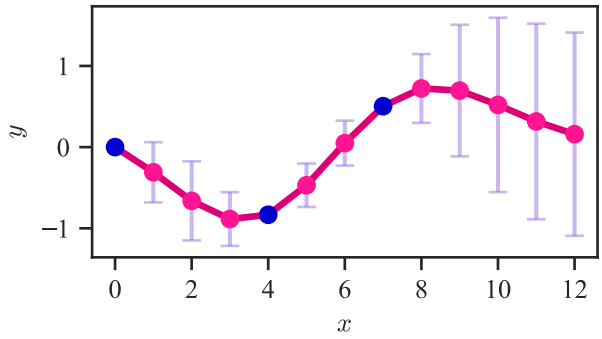


Figure 4.3: Mean and 2-sigma limits of multivariate Gaussian conditioned on x_0, x_4 and x_7.

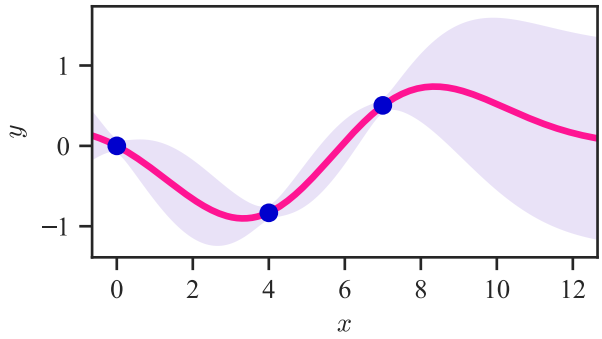


Figure 4.4: Mean and 2-sigma limits of GP conditioned on x_0, x_4 and x_7.

In order to formalize this concept, a GP is used as a prior for Bayesian inference over a real-valued function f:

$$f \sim \mathcal{GP}(m, k). \tag{4.15}$$

Furthermore, a zero mean function and a Squared Exponential (SE) covariance function

$$m(\boldsymbol{x}) = 0, \quad k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{L}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right), \tag{4.16}$$

is assumed, where $\boldsymbol{L} = \text{diag}(\ell_1^2, \ldots, \ell_n^2)$ is a diagonal matrix of the squared length-scales

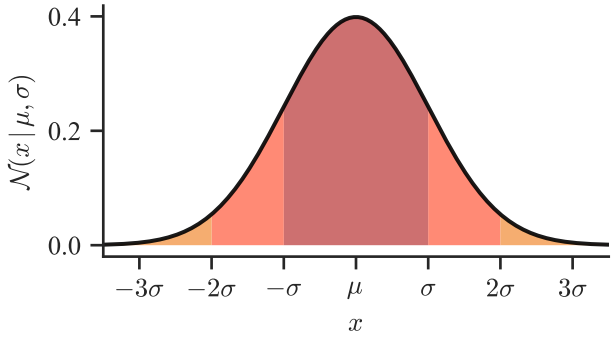


Figure 4.5: 1-sigma, 2-sigma and 3-sigma bounds for a Gaussian distributed variable x with mean μ and variance σ^2.

and σ_f^2 is the signal variance. Other mean and covariance functions may be used, of which the Matérn, exponential and rational quadratic covariance are among the mentioned in Rasmussen and Williams 2006. A zero mean and SE kernel is, however, often used in practice to model continuous functions, and is used in this work. The SE kernel matrix is visualized in Figure 4.6.

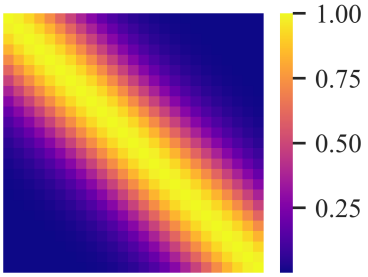


Figure 4.6: Colormap of a SE kernel matrix, for a linearly spaced dataset.

Like for Bayesian linear regression, Bayes' theorem can be used to obtain the joint posterior between the training values $\boldsymbol{f} \in \mathbb{R}^M$ and the test values $\boldsymbol{f}^* \in \mathbb{R}^{M^*}$:

$$p(\boldsymbol{f}, \boldsymbol{f}^* \mid \boldsymbol{y}) = \frac{p(\boldsymbol{f}, \boldsymbol{f}^*)p(\boldsymbol{y} \mid \boldsymbol{f})}{p(\boldsymbol{y})}. \tag{4.17}$$

Zero mean Gaussian noise is again assumed as for Eq. (4.2), such that the likelihood is

given by $p(\boldsymbol{y} \mid \boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}, \sigma_n^2 \boldsymbol{I})$ and the joint GP prior is

$$p(\boldsymbol{f}, \boldsymbol{f}^*) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_{x,x} & \boldsymbol{K}_{x,x^*} \\ (\boldsymbol{K}_{x,x^*})^\top & \boldsymbol{K}_{x^*,x^*} \end{bmatrix}\right), \tag{4.18}$$

analogous to the joint Gaussian formula in Eq. (B.1). Here the shorthand notations $\boldsymbol{K}_{x,x} = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) \in \mathbb{R}^{M \times M}$, $\boldsymbol{K}_{x,x^*} = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}^*) \in \mathbb{R}^{M \times M^*}$ and $\boldsymbol{K}_{x^*,x^*} = \boldsymbol{K}(\boldsymbol{X}^*, \boldsymbol{X}^*) \in \mathbb{R}^{M^* \times M^*}$ are introduced. $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}')$ is the covariance matrix, with entries $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}'_j)$ given by the covariance function k.

Marginalizing out the training values, like in Eq. (4.9), the predictive distribution

$$\begin{aligned} p(\boldsymbol{f}^* \mid \boldsymbol{y}) = \mathcal{N}\big(&\boldsymbol{K}_{x,x^*}^\top (\boldsymbol{K}_{x,x} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y}, \\ &\boldsymbol{K}_{x^*,x^*} - \boldsymbol{K}_{x,x^*}^\top (\boldsymbol{K}_{x,x} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{K}_{x,x^*}\big) \end{aligned} \tag{4.19}$$

is obtained, which is identical to the kernelized Bayesian linear regression in Eq. (4.13). This shows how GPR can be retrieved from kernelized Bayesian linear regression, with infinite-dimensional kernel functions.

4.2.2 Learning the hyperparameters

How the mean and covariance functions are parameterized, and the values of those parameters, termed hyperparameters, specify the shape of the GP. In order to get a GP that generalizes well to the data, it is therefore important to find appropriate values for these parameters. In the case of the SE kernel these are the squared length-scale matrix $\boldsymbol{L}$, the signal variance σ_f^2 and the noise variance σ_n^2. In the following, it is briefly explored how these hyperparameters shape the resulting predictive distribution for a one-dimensional example, meaning that $\boldsymbol{L} = \ell$. In Figure 4.7 it is seen how varying the length-scale ℓ changes the "wiggliness" of the function, while in Figure 4.8 the signal variance σ_f^2 changes the vertical span of the function. Finally in Figure 4.9 the effect of varying the noise variance σ_n^2 is shown. The kernel hyperparameters are usually determined by

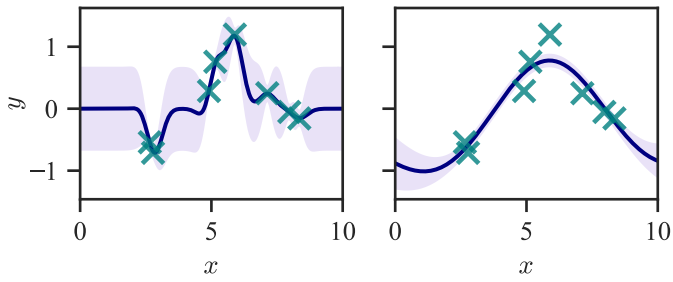


Figure 4.7: GP mean and 2-sigma variance bounds for $\ell = 0.3$ (left) and $\ell = 3$ (right), with $\sigma_n^2 = 0.001$ and $\sigma_f^2 = 0.05$ fixed. The training dataset is also shown.

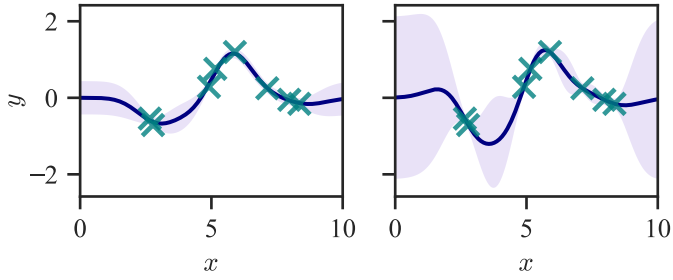


Figure 4.8: GP mean and 2-sigma variance bounds for $\sigma_f^2 = 0.02$ (left) and $\sigma_f^2 = 0.5$ (right), with $\sigma_n^2 = 0.001$ and $\ell = 0.8$ fixed. The training dataset is also shown.

maximizing the log marginal likelihood

$$\begin{aligned}\log p(\boldsymbol{y} \mid \boldsymbol{X}) = &-\frac{1}{2}\boldsymbol{y}^\top \left(\boldsymbol{K}_{x,x} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} \\ &-\frac{1}{2}\log\det\left(\boldsymbol{K}_{x,x} + \sigma_n^2 \boldsymbol{I}\right) - \frac{n}{2}\log 2\pi,\end{aligned} \tag{4.20}$$

such that the optimal hyperparameters $\boldsymbol{\theta} = (\boldsymbol{L}, \sigma_f, \sigma_n)$ are given by

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{y} \mid \boldsymbol{X}). \tag{4.21}$$

To illustrate the process of learning the hyperparameters by maximizing the log marginal likelihood, a simple example is considered, with a dataset $\mathcal{D}$ of size $M = 8$ shown in Figure 4.10 together with the underlying function f. The initial GP prior before training

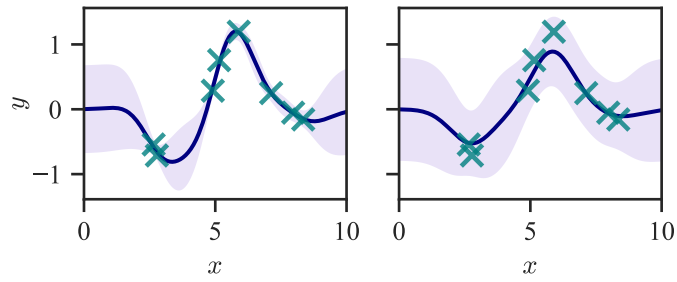


Figure 4.9: GP mean and 2-sigma variance bounds for $\sigma_n^2 = 0.001$ (left) and $\sigma_n^2 = 0.02$ (right), with $\sigma_f^2 = 0.05$ and $\ell = 0.8$ fixed. The training dataset is also shown.

is shown in Figure 4.11a. A zero mean function and SE covariance function is again assumed. After training the GP one then gets the resulting posterior shown in Figure 4.11b. 100 samples, denoted by f_s, are also shown for the prior and posterior, showing how the mean and covariance shapes the GP realizations.

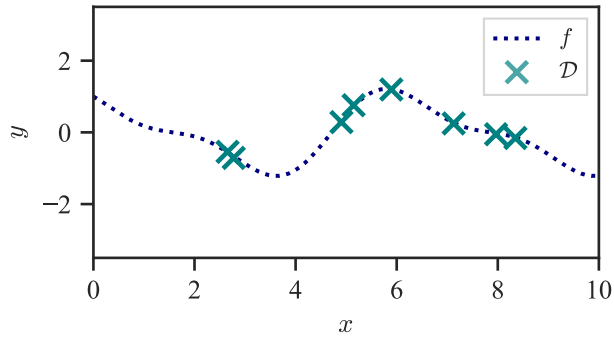
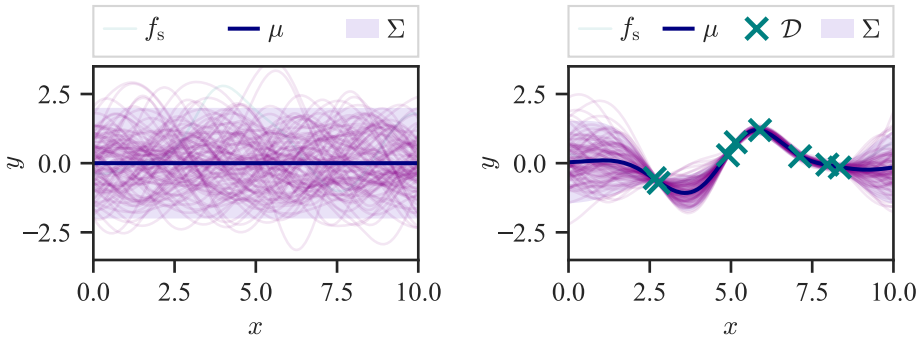


Figure 4.10: True function f and sampled dataset $\mathcal{D}$.

The computation of the inverse covariance matrix is typically done using the Cholesky decomposition $\boldsymbol{K} = \boldsymbol{L}\boldsymbol{L}^\top$, where $\boldsymbol{L}$ is lower triangular. The resulting inverse algorithm has computation complexity $\mathcal{O}(M^3)$, meaning learning, as well as prediction, also has complexity $\mathcal{O}(M^3)$. This poses major limitations for the scalability of GPR. In the next section, it will be seen how to overcome these limitations with sparse GP approximation methods.



(a) (b)

Figure 4.11: Prior GP distribution with zero mean and unit variance (left), and posterior GP predictive distribution with mean μ and variance Σ (right), visualized with 2-sigma bounds. Samples f_s from the prior and posterior distributions are shown.

4.3 Sparse GP methods

The main limitation of exact GPR is that the computational complexity is $\mathcal{O}(M^3)$, from the inversion of the covariance matrix..Furthermore, even after the inverse covariance matrix is calculated, making new predictions still scales with $\mathcal{O}(M^2)$. Sparse GP methods are a possible way to overcome this limitation on scalability, such that GPR can be used on large and high-dimensional datasets.

The underlying principle behind sparse GP methods is to introduce $\widetilde{M}$ inducing variables $\widetilde{\boldsymbol{f}}$, with corresponding input values $\widetilde{\boldsymbol{x}}$, where $\widetilde{M} \ll M$. $\widetilde{\boldsymbol{x}}$ could either be a subset of the training inputs or pseudo-points, and should be found such that they summarize the complete dataset as best as possible. Therefore, for sparse GP methods, both the already discussed hyperparameters $\boldsymbol{\theta} = (\boldsymbol{L}, \sigma_f, \sigma_n)$ and the inducing inputs $\widetilde{\boldsymbol{x}}$ need to be learned for sparse GP methods.

In the following, two different approaches to sparse GPs will be discussed, namely approximating the joint distribution with Fully Independent Training Conditional (FITC) from Snelson and Ghahramani 2005 and approximating the posterior with VFE from Titsias 2009. The SVGP method in Hensman et al. 2013 will also be discussed, which builds upon the principle behind the VFE method, and further improves on its scalability.

In addition to these references the section is also based on Quiñonero-Candela and Rasmussen 2005 and Liu et al. 2020.

4.3.1 Fully Independent Training Conditional

Many sparse GP methods, including the FITC method introduced in Snelson and Ghahramani 2005, adds a conditional distribution q in order to approximate the joint distribution between the training points $\boldsymbol{f}$ and the test points $\boldsymbol{f}^*$, given previously in Eq. (4.18). The main assumption behind this approximate distribution is that $\boldsymbol{f}$ and $\boldsymbol{f}^*$ are assumed conditionally independent given $\widetilde{\boldsymbol{x}}$, such that

$$\begin{aligned} p\left(\boldsymbol{f}^*, \boldsymbol{f}\right) &= \int p\left(\boldsymbol{f}^*, \boldsymbol{f} \mid \widetilde{\boldsymbol{x}}\right) p(\widetilde{\boldsymbol{x}}) \mathrm{d}\widetilde{\boldsymbol{x}} \\ &\simeq q\left(\boldsymbol{f}^*, \boldsymbol{f}\right) = \int q\left(\boldsymbol{f}^* \mid \widetilde{\boldsymbol{x}}\right) q(\boldsymbol{f} \mid \widetilde{\boldsymbol{x}}) p(\widetilde{\boldsymbol{x}}) \mathrm{d}\widetilde{\boldsymbol{x}}. \end{aligned} \tag{4.22}$$

How $q\left(\boldsymbol{f}^* \mid \widetilde{\boldsymbol{x}}\right)$ and $q(\boldsymbol{f} \mid \widetilde{\boldsymbol{x}})$ are formed using additional assumptions are what sets these different sparse GP methods apart.

The FITC method uses the following approximation for the training conditional:

$$q_{\mathrm{FITC}}(\boldsymbol{f} \mid \widetilde{\boldsymbol{x}}) = \mathcal{N}\left(\boldsymbol{K}_{x,\tilde{x}} \boldsymbol{K}_{\tilde{x},\tilde{x}}^{-1} \widetilde{\boldsymbol{x}}, \operatorname{diag}(\boldsymbol{K}_{x,x} - \boldsymbol{Q}_{x,x})\right), \tag{4.23}$$

where the following shorthand notation is introduced: $\boldsymbol{Q}_{a,b} = \boldsymbol{K}_{a,\tilde{x}} \boldsymbol{K}_{\tilde{x},\tilde{x}}^{-1} \boldsymbol{K}_{b,\tilde{x}}^{\top}$, and $\boldsymbol{K}_{a,b} = \boldsymbol{K}(a,b)$ with $\boldsymbol{K}$ as defined after Eq. (4.18).

This results in the following predictive distribution for a single test point f^*:

$$\begin{aligned} q_{\mathrm{FITC}}\left(f^* \mid \boldsymbol{y}\right) &= \mathcal{N}\Big(\boldsymbol{Q}_{x^*,x}\left(\boldsymbol{Q}_{x,x} + \boldsymbol{\Lambda}\right)^{-1} \boldsymbol{y}, \\ &\qquad \boldsymbol{K}_{x^*,x^*} - \boldsymbol{Q}_{x^*,x}\left(\boldsymbol{Q}_{x,x} + \boldsymbol{\Lambda}\right)^{-1} \boldsymbol{Q}_{x,x^*}\Big) \\ &= \mathcal{N}\left(\boldsymbol{K}_{x^*,\tilde{x}} \boldsymbol{\Gamma}^{-1} \boldsymbol{K}_{\tilde{x},x} \boldsymbol{\Lambda}^{-1} \boldsymbol{y}, \right. \\ &\qquad \left. \boldsymbol{K}_{x^*,x^*} - \boldsymbol{Q}_{x^*,x^*} + \boldsymbol{K}_{x^*,\tilde{x}} \boldsymbol{\Gamma}^{-1} \boldsymbol{K}_{\tilde{x},x^*}\right), \end{aligned} \tag{4.24}$$

where $\boldsymbol{\Gamma} = \boldsymbol{K}_{\tilde{x},\tilde{x}} + \boldsymbol{K}_{\tilde{x},x} \boldsymbol{\Lambda}^{-1} \boldsymbol{K}_{x,\tilde{x}}$ and $\boldsymbol{\Lambda} = \operatorname{diag}(\boldsymbol{K}_{x,x} - \boldsymbol{Q}_{x,x} + \sigma_n^2 \boldsymbol{I})$. The latter formulation in Eq. (4.24) avoids the inversion of the large matrix $\boldsymbol{Q}_{x,x} + \boldsymbol{\Lambda}$. The resulting computational complexity is then reduced from $\mathcal{O}(M^3)$ to $\mathcal{O}(M\widetilde{M}^2)$.

4.3.2 Variational Free Energy

The basis of the VFE method from Titsias 2009, as well as the SVGP method discussed in the next section, is to approximate the posterior $p(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y})$, instead of the prior $p\left(\boldsymbol{f}^*, \boldsymbol{f}\right)$, as was done with FITC in Eq. (4.22). A variational distribution $q(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y})$ is then introduced in order to approximate $p(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y})$, and is found by minimizing the Kullback-Leibler (KL)-divergence between the distributions. The KL-divergence can be regarded as a (non-symmetric) distance between two distributions p and q and is defined as

$$\mathrm{KL}\left(p(z) \parallel q(z)\right) = \sum_z p(z) \log \frac{p(z)}{q(z)}. \tag{4.25}$$

The KL-divergence between $p(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y})$ and $q(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y})$ is

$$\mathrm{KL}\left(p(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y}) \parallel q(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y})\right) = \log p(\boldsymbol{y}) - F_q, \tag{4.26}$$

where

$$F_q = \left\langle \log \frac{p\left(\boldsymbol{y}, \boldsymbol{f}, \widetilde{\boldsymbol{f}}\right)}{q\left(\boldsymbol{f}, \widetilde{\boldsymbol{f}} \mid \boldsymbol{y}\right)} \right\rangle_{q(\boldsymbol{f}, \tilde{\boldsymbol{f}} \mid \boldsymbol{y})} \tag{4.27}$$

is termed the evidence lower bound (ELBO) and the shorthand notation $\langle p \rangle_q$ denotes the expectation of p over the distribution q. It can be shown that minimizing this is equivalent to maximizing the ELBO, discussed in detail in Titsias 2009.

Finally the predictive distribution is

$$\begin{aligned} p(f^*) = \mathcal{N}\Big(& f^* \mid \boldsymbol{K}_{x^*,\tilde{x}} \boldsymbol{K}_{\tilde{x},\tilde{x}}^{-1} \boldsymbol{m}, \\ & \boldsymbol{K}_{x^*,x^*} - \boldsymbol{K}_{x^*,\tilde{x}} \boldsymbol{K}_{\tilde{x},\tilde{x}}^{-1} \boldsymbol{K}_{x^*,\tilde{x}}^\top + \boldsymbol{K}_{x^*,\tilde{x}} \boldsymbol{K}_{\tilde{x},\tilde{x}}^{-1} \boldsymbol{S} \boldsymbol{K}_{\tilde{x},\tilde{x}}^{-1} \boldsymbol{K}_{x^*,\tilde{x}}^\top \Big), \end{aligned} \tag{4.28}$$

where

$$\begin{aligned} \boldsymbol{m} &= \sigma^{-2} \boldsymbol{K}_{\tilde{x},\tilde{x}} \boldsymbol{S} \boldsymbol{K}_{\tilde{x},x} \boldsymbol{y}, \\ \boldsymbol{S} &= \left(\boldsymbol{K}_{\tilde{x},\tilde{x}} + \sigma^{-2} \boldsymbol{K}_{\tilde{x},x} \boldsymbol{K}_{x,\tilde{x}}\right)^{-1}, \end{aligned} \tag{4.29}$$

and $\boldsymbol{m} \in \mathbb{R}^{\widetilde{M}}, \boldsymbol{S} \in \mathbb{R}^{\widetilde{M} \times \widetilde{M}}$. Evaluating the mean and covariance of the predictive distribution again results in $\mathcal{O}(M\widetilde{M}^2)$ computational complexity.

4.3.3 Sparse Variational Gaussian Process

The SVGP method from Hensman et al. 2013 uses stochastic variational inference to improve the scalability of VFE. They obtain the relaxed bound of the ELBO

$$F_q = \langle \log p(\boldsymbol{y} \mid \boldsymbol{f}) \rangle_{p(\boldsymbol{f}|\tilde{\boldsymbol{f}}) q(\tilde{\boldsymbol{f}}|\boldsymbol{y})} - \mathrm{KL}\left(q\left(\widetilde{\boldsymbol{f}} \mid \boldsymbol{y}\right) \parallel p\left(\widetilde{\boldsymbol{f}}\right)\right), \tag{4.30}$$

which is minimized. Furthermore, this cost is well suited for stochastic gradient descent (SGD), such that training can be done using mini-batches, which can help accelerate learning. Prediction is done as in Eq. (4.28), yet it is important to note that a big difference with VFE is that $\boldsymbol{m}$ and $\boldsymbol{S}$ are now additional variational parameters in the optimization problem.

To illustrate sparse GPR a simple example is considered with a one-dimensional nonlinear function f, shown in Figure 4.12 together with the sampled dataset $\mathcal{D}$. The GP prior and the $\widetilde{M} = 40$ randomly chosen initial inducing inputs are shown in Figure 4.13a, with initial hyperparameter values $\sigma_n^2 = 0.75, \sigma_f^2 = 1.0, \ell = 1.0$.

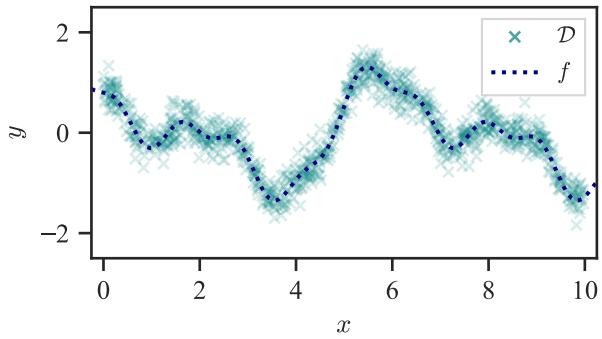


Figure 4.12: True function f and sampled dataset $\mathcal{D}$.

After training using the SVGP method, the predictive distribution shown in Figure 4.13b is obtained. By optimizing the inducing inputs it is observed that they spread out to cover the entire space more evenly.

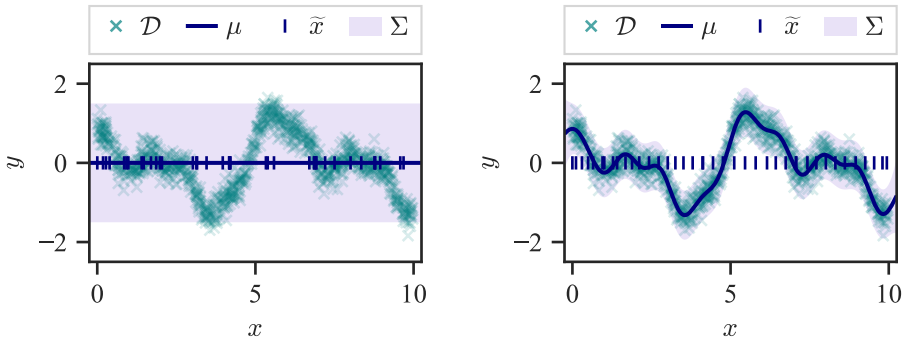


Figure 4.13: Prior (left) and posterior (right) GP distributions, with initial random inducing point locations and final locations after training with SVGP. The variance is indicated with 3-sigma limits.

4.4 GP-based MPC

In the following section, the GP-MPC method will be introduced, based on Hewing, Liniger, et al. 2018 and Hewing, Kabzan, et al. 2019. GP-MPC considers combining an a priori dynamics model with a disturbance model given by a GP in a nonlinear MPC controller. The GP is trained on previous trajectory tracking trials and added in the dynamics in a stochastic MPC problem with chance constraints. How the state distribution is propagated, as well as the cost and constraints, will be considered in detail in order to formulate a tractable OCP.

Before going into further detail on GP-MPC, it is worth mentioning that many other approaches exist for learning-based trajectory tracking control of robots. Reinforcement learning is a model-free paradigm that has been widely used in recent years for robotic manipulation tasks, e.g., in Gu et al. 2017. The approach is based on learning the control policy directly from having the robot repeat a task in its environment. Iterative learning control is a similar approach that also has seen frequent use for robotic manipulation, which is more rooted in traditional control approaches, compared to reinforcement learning, which is more rooted in optimization and machine learning. The main idea is, however, similar in that an action is repeated and the control law is adjusted iteratively based on the performance.

Partly what makes including GP dynamics in an MPC controller interesting is that uncertainty can be considered directly in the controller. Information about the uncertainty of neural networks can, however, also be deduced, for instance, in Loquercio et al. 2020. There are naturally also other Bayesian options, such as Bayesian linear regression, discussed earlier in Section 4.1. These approaches are applied to MPC in McKinnon and Schoellig 2019. However, the combination of a prior model with a learned disturbance model, as well as how uncertainty in the model can be handled directly in the controller in order to achieve cautiousness, makes GP-MPC an interesting approach.

4.4.1 GP disturbance model

In the following the discrete-time dynamical system given by the model

$$\boldsymbol{x}_{i+1} = \boldsymbol{f}\left(\boldsymbol{x}_i, \boldsymbol{u}_i\right) + \boldsymbol{B}_d\left(\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{w}_i\right), \tag{4.31}$$

is considered, with system state $\boldsymbol{x} \in \mathbb{R}^{n_x}$, control input $\boldsymbol{u} \in \mathbb{R}^{n_u}$ and $i \in \mathbb{N}^0$ denoting the time step. The model consisting of a nominal process model $\boldsymbol{f}$ and a disturbance model $\boldsymbol{g}$, the latter of which describes unknown discrepancies between the nominal model and the true model. The disturbance acts on the system through the process disturbance matrix $\boldsymbol{B}_d \in \mathbb{R}^{n_x \times n_y}$. The system is also affected by i.i.d. zero mean Gaussian process noise $\boldsymbol{w}(k) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_n)$, where $\boldsymbol{\Sigma}_n = \text{diag}(\sigma_{n,1}^2, \ldots, \sigma_{n,n_x}^2)$. Furthermore, the system is subject to general state and input constraints

$$\boldsymbol{x}_i \in \mathcal{X}, \quad \boldsymbol{u}_i \in \mathcal{U}, \tag{4.32}$$

where $\mathcal{X}$ is the state constraint set and $\mathcal{U}$ is the control input constraint set.

The basis behind GP-MPC is that a GP prior is assumed on the disturbance function $\boldsymbol{g}$, such that the disturbance dynamics can be inferred using GP regression. Specifically, the input points are the state and control input measurements $\boldsymbol{z}_i = [\boldsymbol{x}_i^\top \ \boldsymbol{u}_i^\top]^\top \in \mathbb{R}^{n_z}$, with $n_z = n_x + n_u$, and the outputs are the disturbances

$$\boldsymbol{y}_i = \boldsymbol{g}\left(\boldsymbol{x}_i, \boldsymbol{u}_i\right) + \boldsymbol{w}_i = \boldsymbol{B}_d^\dagger\left(\boldsymbol{x}_{i+1} - \boldsymbol{f}\left(\boldsymbol{x}_i, \boldsymbol{u}_i\right)\right). \tag{4.33}$$

The dataset is then given by $\mathcal{D} = \{\boldsymbol{Z}, \boldsymbol{Y}\}$, with M inputs $\boldsymbol{Z} = [\boldsymbol{z}_0^\top \cdots \boldsymbol{z}_M^\top]^\top \in \mathbb{R}^{M \times n_z}$

and outputs $\boldsymbol{Y} = [\boldsymbol{y}_0^\top \cdots \boldsymbol{y}_M^\top]^\top \in \mathbb{R}^{M \times n_y}$. An independent GP prior is assumed on every output dimension of $\boldsymbol{g}$, such that every column of $\boldsymbol{Y}$ is normally distributed with

$$\boldsymbol{Y}_{\cdot,j} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{z,z}^j + \sigma_{n,j}^2 \boldsymbol{I}), \tag{4.34}$$

where $j \in \mathbb{N}_{n_y}$ denotes the j-th dimension of the output, and $\boldsymbol{K}_{z,z}^j = \boldsymbol{K}_j(\boldsymbol{z}, \boldsymbol{z})$ is the covariance matrix, whose elements are given by the covariance function $k_j(\boldsymbol{x}, \boldsymbol{x}')$. A zero mean function and squared exponential covariance function is assumed, like in Eq. (4.16).

Eq. (4.19) is then applied to a single test point $\boldsymbol{z}^*$, in order to obtain the predictive distribution of every output dimension:

$$\begin{aligned} \mu_j^d(\boldsymbol{z}^*) &= (\boldsymbol{K}_{z,z^*}^j)^\top \left(\boldsymbol{K}_{z,z}^j + \sigma_{n,j}^2 \boldsymbol{I}\right)^{-1} \boldsymbol{Y}_{\cdot,j}, \\ \Sigma_j^d(\boldsymbol{z}^*) &= \boldsymbol{K}_{z^*,z^*}^j - (\boldsymbol{K}_{z,z^*}^j)^\top \left(\boldsymbol{K}_{z,z}^j + \sigma_{n,j}^2 \boldsymbol{I}\right)^{-1} \boldsymbol{K}_{z,z^*}^j. \end{aligned} \tag{4.35}$$

The resulting GP approximation for all dimensions of $\boldsymbol{g}$ is then given by

$$\boldsymbol{d}(\boldsymbol{z}^*) \sim \mathcal{N}\left(\boldsymbol{\mu}^d(\boldsymbol{z}^*), \boldsymbol{\Sigma}^d(\boldsymbol{z}^*)\right), \tag{4.36}$$

with $\boldsymbol{\mu}_d = [\mu_1^d \cdots \mu_n^d]^\top$ and $\boldsymbol{\Sigma}_d = \operatorname{diag}(\Sigma_1^d, \ldots, \Sigma_n^d)$.

The GP approximation of the true disturbance $\boldsymbol{g}$ is used in the discrete-time dynamical model in Eq. (4.31), such that

$$\boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{B}_d(\boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{w}_i) \tag{4.37}$$

describes the stochastic model.

4.4.2 Stochastic MPC problem

The model in Eq. (4.37) cannot be used directly in a deterministic OCP of the form in Eq. (3.8), as the disturbance $\boldsymbol{d}$ is modeled by a GP and is inherently stochastic. The state vector itself is, therefore, also a random variable with some distribution. In the following, robust MPC and stochastic MPC will be briefly outlined to put the present problem in a theoretical context. Then the specific stochastic MPC problem with GP dynamics will be formulated.

Robust MPC and stochastic MPC, as discussed in Rawlings et al. 2019 and Mesbah

2016, provide two different approaches for how to handle uncertainties in optimal control. In brief, the fundamental assumption of robust MPC is that the uncertainty in the system lies in some bounded set. One can then tighten the state constraints in a new MPC problem, such that the original state constraints are guaranteed to be satisfied in the worst-case realization of the bounded disturbance.

Stochastic MPC makes assumptions about the distribution, rather than the boundedness, of the disturbance. Then, instead of guaranteeing constraint satisfaction, one provides chance constraints of the form

$$\Pr\left(\boldsymbol{x}_i \in \mathcal{X}\right) \geq 1 - \epsilon, \tag{4.38}$$

stating that the probability of constraint violation at time step i should be less than $\epsilon \in (0, 1)$. Furthermore, considering the stochastic nature of the state, one typically defines a cost function of the form

$$J(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}\left[\sum_{i=0}^{N-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_N)\right], \tag{4.39}$$

i.e., as the expectation of the sum of the stage costs over the prediction horizon.

A stochastic MPC problem can then be formulated, using the stochastic model in Eq. (4.37):

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \mathbb{E}\left[\sum_{i=0}^{N-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_N)\right] \tag{4.40a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{B}_d(\boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{w}_i), \quad i = 0, \ldots, N-1, \tag{4.40b}$$

$$\Pr\left(\boldsymbol{x}_i \in \mathcal{X}\right) \geq 1 - \epsilon, \quad i = 0, \ldots, N, \tag{4.40c}$$

$$\boldsymbol{u}_i \in \mathcal{U}, \quad i = 0, \ldots, N-1, \tag{4.40d}$$

$$\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0, \tag{4.40e}$$

where ℓ is the stage cost, ℓ_f is the terminal cost, N is the prediction horizon, and $\bar{\boldsymbol{x}}_0$ is the state measurement at the current time step. Again it is used that $\boldsymbol{x} = [\, \boldsymbol{x}_0^\top \;\; \cdots \;\; \boldsymbol{x}_N^\top \,]^\top$ and $\boldsymbol{u} = [\, \boldsymbol{u}_0^\top \;\; \cdots \;\; \boldsymbol{u}_{N-1}^\top \,]^\top$ are the optimization variables. Chance constraints on the state variable are introduced, with maximum probability ϵ of constraint violation. The control input is considered deterministic, such that the input constraints are formulated as

regular deterministic constraints.

The optimization problem in Eq. (4.40) is untractable, since the time propagation of the state variables in Eq. (4.37) renders the state distribution non-Gaussian. In the following, several approximations will be made, in order to formulate the cost function, chance constraints, and most importantly, the uncertainty propagation, resulting in a computationally tractable MPC problem.

4.4.3 State distribution propagation

In order to get a tractable way of propagating the uncertainty over the prediction horizon, the state, control input, and disturbance are approximated as jointly Gaussian at every time step. Furthermore, a first-order Taylor approximation is applied. It is shown in Hewing, Liniger, et al. 2018 that this results in the following propagation equations for the state mean and variance:

$$\boldsymbol{\mu}_{i+1}^{x} = \boldsymbol{f}(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i) + \boldsymbol{B}_d \boldsymbol{\mu}^d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i), \tag{4.41}$$

$$\boldsymbol{\Sigma}_{i+1}^{x} = \underbrace{\begin{bmatrix} \nabla_x \boldsymbol{f}(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i) & \boldsymbol{B}_d \end{bmatrix}}_{\widetilde{\boldsymbol{A}}_i} \underbrace{\begin{bmatrix} \boldsymbol{\Sigma}_i^x & \boldsymbol{\Sigma}_i^{xd} \\ (\boldsymbol{\Sigma}_i^{xd})^\top & \boldsymbol{\Sigma}_i^d + \boldsymbol{\Sigma}_n \end{bmatrix}}_{\boldsymbol{\Sigma}_i} \underbrace{\begin{bmatrix} \nabla_x \boldsymbol{f}(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i)^\top \\ \boldsymbol{B}_d^\top \end{bmatrix}}_{\widetilde{\boldsymbol{A}}_i^\top}, \tag{4.42}$$

where

$$\boldsymbol{\Sigma}_i^{xd} = \boldsymbol{\Sigma}_i^x \nabla_x \boldsymbol{\mu}_d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i)^\top, \tag{4.43}$$

$$\boldsymbol{\Sigma}_i^{d} = \boldsymbol{\Sigma}_d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i) + \nabla_x \boldsymbol{\mu}_d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i) \boldsymbol{\Sigma}_i^x \nabla_x \boldsymbol{\mu}_d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i)^\top, \tag{4.44}$$

and $i \in \mathbb{N}^0$ is the time step.

The process of propagating the uncertainty over the MPC time horizon is visualized in Figure 4.14, for the joint states of a 2 DOF robot manipulator.

4.4.4 Cost function

There are several options for how to realize the stochastic cost function in Eq. (4.40a) by choosing the stage cost ℓ and terminal cost ℓ_f. Since trajectory tracking MPC is usually accomplished with a quadratic cost function, a logical choice would be to consider the

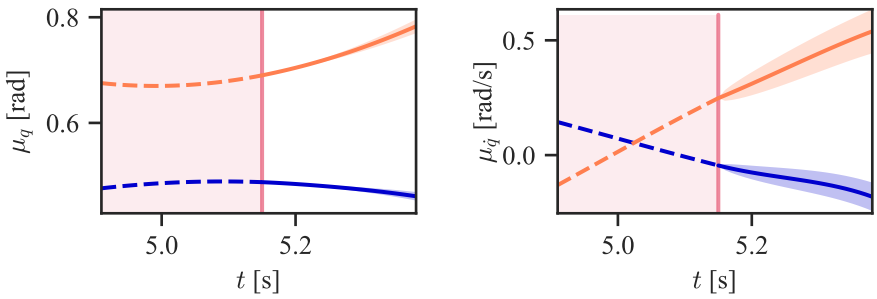


Figure 4.14: Time propagation of state distribution, here shown with 50-sigma variance bounds to exaggerate the effect. The dashed lines indicate the closed-loop solution leading up to the current time step.

expectation of a quadratic form, i.e., the cost function

$$\begin{aligned} J_e &= \mathbb{E}\left[\sum_{i=0}^{N-1} \left(\|\boldsymbol{x}_i - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2\right) + \|\boldsymbol{x}_N - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2\right] \\ &= \sum_{i=0}^{N-1} \left(\|\boldsymbol{\mu}_i^x - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \mathrm{tr}(\boldsymbol{Q}\boldsymbol{\Sigma}_i^x) + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2\right) + \|\boldsymbol{\mu}_N^x - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2 + \mathrm{tr}(\boldsymbol{P}\boldsymbol{\Sigma}_N^x), \end{aligned} \tag{4.45}$$

where $\boldsymbol{r}_i^x$ is the reference trajectory at the i-th time step, $\boldsymbol{Q}$ is the state weight matrix, $\boldsymbol{R}$ is the control input weight matrix and $\mathrm{tr}(\cdot)$ denotes the trace. The control input is assumed deterministic, which is why the additional trace term only appears for the state variable. The reader is referred to Cao et al. 2017 for the details on the derivation of the expectation of a quadratic form used in Eq. (4.45).

While Eq. (4.45) provides a simple and differentiable cost function, it is also possible to take a so-called certainty equivalence approach, as mentioned in Carron et al. 2019, where it is approximated that the expectation of the cost equals the cost evaluated in the state mean, such that one gets a quadratic cost evaluated in the mean for the stage cost and terminal cost:

$$J_c = \sum_{i=0}^{N-1} \left(\|\boldsymbol{\mu}_i^x - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2\right) + \|\boldsymbol{\mu}_N^x - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2. \tag{4.46}$$

While the certainty equivalence principle does not apply here, i.e., that the optimal solution

to the stochastic MPC problem is equivalent to the solution of the deterministic version evaluated in the mean, it is still a reasonable approximation to do in practice.

4.4.5 Chance constraints

By approximating the state distribution as Gaussian and restricting the state constraint set $\mathcal{X}$ to only linear constraints, one can formulate a tractable version of the chance constraints in Eq. (4.40c). As discussed in Hewing, Kabzan, et al. 2019, under the assumption of a Gaussian state distribution, the distribution is uniquely defined by $\boldsymbol{\mu}_i^x$ and $\boldsymbol{\Sigma}_i^x$, such that the state constraints can also be formulated using $\boldsymbol{\mu}_i^x$ and $\boldsymbol{\Sigma}_i^x$ alone. Not that the dependence on the i-th time step is omitted in the following section on chance constraints for notational clarity.

Given the above mentioned assumptions, the j-th linear chance constraint in $\mathcal{X}$ can be written as

$$\Pr\left(\boldsymbol{x} \in \mathcal{X}_j\right) > 1 - \epsilon_j, \quad \mathcal{X}_j = \{\boldsymbol{x} \mid \boldsymbol{a}_j^\top \boldsymbol{x} \leq b_j\}, \tag{4.47}$$

defined using $\boldsymbol{a}_j \in \mathbb{R}_x^n$ and $b_j \in \mathbb{R}$, and with probability of constraint violation ϵ_j, for $j \in \mathbb{N} = \{1, 2, \dots\}$. The tightened constraint on the Gaussian state mean is then given in Hewing, Kabzan, et al. 2019 as $\boldsymbol{\mu}^x \in \mathcal{Z}_j^l$, where the tightened state constraint set $\mathcal{Z}_j^l$ is given by

$$\mathcal{Z}_j^l = \left\{ \boldsymbol{z} \mid \boldsymbol{a}_j^\top \boldsymbol{z} \leq b_j - \Phi^{-1}(1-\epsilon_j)\sqrt{\boldsymbol{a}_j^\top \boldsymbol{\Sigma}^x \boldsymbol{a}_j} \right\} \tag{4.48}$$

and Φ^{-1} is the inverse standard normal cumulative distribution.

Specifically for state bounds, i.e., $x_j \leq b_j$, $j \in \mathbb{N}_{n_x}$, the simplification that $\boldsymbol{a}_j^\top \boldsymbol{\Sigma}^x \boldsymbol{a}_j = \boldsymbol{\Sigma}_{j,j}^x$ is obtained. These single state dimension bounds can then be stacked such that the joint state chance constraints $\boldsymbol{\mu}_i^x \in \mathcal{Z}^b$ is obtained, with the set

$$\mathcal{Z}^b = \left\{ \boldsymbol{z} \mid \boldsymbol{z} \leq \boldsymbol{b} - \Phi^{-1}(1-\epsilon)\sqrt{\text{diag}(\boldsymbol{\Sigma}^x)} \right\}, \tag{4.49}$$

with $\boldsymbol{b} \in \mathbb{R}^{n_x}$ and where the square root in Eq. (4.49) is element-wise. ϵ is the violation probability for the entire joint state chance constraint. As noted in Paulson et al. 2020, as long as

$$\sum_{j=1}^{n_x} \epsilon_j \leq \epsilon, \tag{4.50}$$

the joint chance states constraint will be satisfied. The simple choice, which is done in this

work, is then to let $\epsilon_i = \frac{\epsilon}{n_x} \; \forall \; j \in \mathbb{N}_{n_x}$, i.e., let the violation probabilities be static and split evenly over all the constraints. As stated in Paulson et al. 2020, this may, however, result in significant conservatism in the MPC controller.

Finally, to get two-sided bounds, i.e., $\|\boldsymbol{x}\| \leq \boldsymbol{b}$, one simply takes the union of the lower bound constraint set and upper bound constraint set, defined using Eq. (4.49), and half the violation probability in order to get an overall constraint violation probability of ϵ for the two-sided bounds:

$$\mathcal{Z} = \left\{ \boldsymbol{z} \;\middle|\; \|\boldsymbol{z}\| \leq \boldsymbol{b} - \Phi^{-1}\left(1 - \frac{\epsilon}{2}\right) \sqrt{\text{diag}(\boldsymbol{\Sigma}^x)} \right\}. \tag{4.51}$$

The chance constraints in Eq. (4.51) tighten the deterministic constraints based on how certain the prediction is and how certain it is specified that the chance constraints should be. This effectively creates an inherently cautious controller. When the system is in a region where the GP is uncertain, the controller will be more cautious, and when the GP produces a prediction with low variance, the controller can push the system closer to the bounds.

This concept is illustrated in Figure 4.15, where an open-loop solution of the MPC problem is shown. As the uncertainty of the GP prediction grows over the prediction horizon, the controller compensates by moving further away from the state bounds. This highlights a consequential advantage of using probabilistic regression methods in predictive control, namely that the uncertainty estimates can be actively used to create a cautious controller.

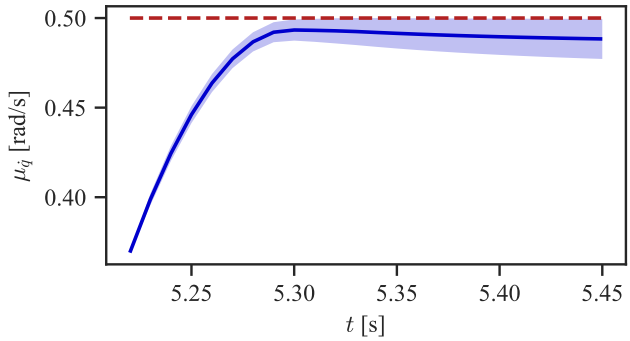


Figure 4.15: Open loop solution of GP-MPC showing the chance constraints, here with 3.5-sigma bounds for the joint velocities of a robot manipulator.

4.4.6 Sparse GP dynamics

As discussed in Section 4.3, evaluation of the GP prediction equations in Eq. (4.35) has $\mathcal{O}(M^2)$ complexity. For the requirements of real-time MPC this heavily restricts the size of the dataset, which in turn means the GP prediction will be poor and prone to overfitting. Sparse GP methods can be applied in order to use larger datasets while still keeping the complexity low. In the following the SVGP method discussed in Section 4.3.3 is applied to learn the inducing points and make predictions. The prediction equations, as defined previously in Eq. (4.28), are given for a single test point and output dimension by

$$\begin{aligned} \mu_j^d(\boldsymbol{z}) &= \boldsymbol{K}_{z^*,\tilde{z}}^j (\boldsymbol{K}_{\tilde{z},\tilde{z}}^j)^{-1} \boldsymbol{m}_j, \\ \Sigma_j^d(\boldsymbol{z}) &= \boldsymbol{K}_{z^*,z^*}^j - \boldsymbol{K}_{z^*,\tilde{z}}^j (\boldsymbol{K}_{\tilde{z},\tilde{z}}^j)^{-1} \boldsymbol{K}_{\tilde{z},z^*}^j \\ &\quad + \boldsymbol{K}_{z^*,\tilde{z}}^j (\boldsymbol{K}_{\tilde{z},\tilde{z}}^j)^{-1} \boldsymbol{S}_j (\boldsymbol{K}_{\tilde{z},\tilde{z}}^j)^{-1} \boldsymbol{K}_{\tilde{z},z^*}^j, \end{aligned} \tag{4.52}$$

with $\boldsymbol{m}_j$ and $\boldsymbol{S}_j$ being hyperparameters and $j \in \mathbb{N}_{n_x}$.

4.4.7 Tractable MPC problem

By applying the expectation of quadratic form cost function in Eq. (4.45) and the linear two-sided chance constraint bounds in Eq. (4.51), the following tractable MPC problem is obtained:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \sum_{i=0}^{N-1} \left(\|\boldsymbol{\mu}_i^x - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \operatorname{tr}(\boldsymbol{Q}\boldsymbol{\Sigma}_i^x) + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2 \right) + \|\boldsymbol{\mu}_N^x - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2 + \operatorname{tr}(\boldsymbol{P}\boldsymbol{\Sigma}_N^x) \tag{4.53a}$$

$$\text{s.t.} \quad \boldsymbol{\mu}_{i+1}^x = \boldsymbol{f}(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i) + \boldsymbol{B}_d \boldsymbol{\mu}^d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i), \quad i = 0, \ldots, N-1, \tag{4.53b}$$

$$\boldsymbol{\Sigma}_{i+1}^x = \widetilde{\boldsymbol{A}}_i \boldsymbol{\Sigma}_i \widetilde{\boldsymbol{A}}_i^\top, \quad i = 0, \ldots, N-1, \tag{4.53c}$$

$$\boldsymbol{\mu}_i \in \mathcal{Z}(\boldsymbol{\Sigma}_i^x), \quad i = 0, \ldots, N, \tag{4.53d}$$

$$\boldsymbol{u}_i \in \mathcal{U}, \quad i = 0, \ldots, N-1, \tag{4.53e}$$

$$\boldsymbol{\mu}_0^x = \bar{\boldsymbol{x}}_0, \tag{4.53f}$$

$$\boldsymbol{\Sigma}_0^x = \boldsymbol{0}, \tag{4.53g}$$

where the initial state mean $\boldsymbol{\mu}_0^x$ is set to be the current state measurement, and known exactly, such that the initial state variance $\boldsymbol{\Sigma}_0^x$ is assumed to be zero.

The increased dimensionality of the optimization problem in Eq. (4.53) from including the state variance dynamics increases the computational demands significantly. Specifically, instead of having n optimization variables, one gets $\frac{1}{2}n(n+3)$ optimization variables, which, especially for higher-dimensional dynamical systems, increases dimensionality considerably. To avoid adding the variance dynamics to the optimization problem, one can make use of the fact that the deviation between the current solution and the previous solution shifted one time step is presumably small. Therefore, by instead evaluating $\boldsymbol{\Sigma}^x$ over the time horizon using the shifted previous solution trajectory, as done in Hewing, Kabzan, et al. 2019, one obtains an approximation that decreases computation time drastically. Furthermore, it should be noted that this approximation makes the two discussed cost functions in Eq. (4.45) and Eq. (4.46) equivalent, since the $\text{tr}(\boldsymbol{Q}\boldsymbol{\Sigma}_i^x)$ term is now constant.

5 Design and implementation

In the following chapter, various design and implementation aspects will be considered. Firstly, the MPC methods described in Chapter 3 and Chapter 4 will be applied to the fixed-base robot manipulator model from Chapter 2 to formulate joint space and task space trajectory tracking OCPs for robot manipulators. Trajectory blending will also be considered, in order to generate smooth trajectories from any initial robot configuration. The developed MPC controllers will also be applied to a free-floating space manipulator system. Modeling and trajectory tracking MPC for this system will be presented. Finally, the software tools used for simulation and implementation of the discussed methods will be presented.

5.1 Trajectory tracking MPC for robot manipulators

In this section, trajectory tracking MPC for robot manipulator arms will be explored. A feedback linearization-based MPC controller will be developed using the inverse dynamics, and a NMPC approach based on the forward dynamics will be discussed. The GP-based MPC approach discussed in Section 4.4 will be applied to the feedback linearized system, as is done in Carron et al. 2019. Then corresponding formulations are also considered for task space trajectories, i.e., end effector pose trajectories.

5.1.1 Joint space trajectory tracking

In the following, we will consider joint space trajectory tracking for robot manipulators using feedback linearization. As outlined in Siciliano et al. 2010, by using the inverse dynamics directly in the control law:

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}), \tag{5.1}$$

and inserting into the dynamics Eq. (2.26), one obtains a feedback linearized system. The resulting system, with control input $\boldsymbol{u}$, will the be n decoupled double integrators

$$\ddot{\boldsymbol{q}} = \boldsymbol{u}. \tag{5.2}$$

The traditional approach to robot manipulator control using feedback linearization is then to apply a variation of a PD control law, as discussed in Siciliano et al. 2010. In the following, a linear MPC controller will instead be formulated to solve the trajectory tracking problem with double integrator dynamics.

The double integrator system in Eq. (5.2) can be discretized exactly, assuming a piecewise constant control input. Exact discretization results in the following discrete-time dynamical system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k, \tag{5.3}$$

with $k \in \mathbb{N}^0$ and system matrices

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I} & \Delta T \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \frac{1}{2}\Delta T^2 \boldsymbol{I} \\ \Delta T \boldsymbol{I} \end{bmatrix}, \tag{5.4}$$

where ΔT is the time step of the discrete-time system. The reader is referred to Chen 1999 for details on exact discretization.

The following trajectory tracking OCP with linear dynamics and quadratic cost function can then be formulated:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \sum_{i=0}^{N-1} \left(\|\boldsymbol{x}_i - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2 \right) + \|\boldsymbol{x}_N - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2 \tag{5.5a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i, \quad i = 0, \dots, N-1, \tag{5.5b}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_i \leq \boldsymbol{x}_{\max}, \quad i = 0, \dots, N, \tag{5.5c}$$

$$\ddot{\boldsymbol{q}}_{\min} \leq \boldsymbol{u}_i \leq \ddot{\boldsymbol{q}}_{\max}, \quad i = 0, \dots, N-1, \tag{5.5d}$$

$$\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0. \tag{5.5e}$$

The cost function Eq. (5.5a) penalizes a deviation from the desired state trajectory $\boldsymbol{r}^x = [\,(\boldsymbol{r}^q)^\top \;\; (\boldsymbol{r}^{\dot{q}})^\top\,]^\top \in \mathbb{R}^{2n}$, where $\boldsymbol{r}^q \in \mathbb{R}^n$ and $\boldsymbol{r}^{\dot{q}} \in \mathbb{R}^n$ are the joint angle and joint angular velocity trajectories. Furthermore, a regularization term is added on the

control input $\boldsymbol{u}_i$, such that the resulting optimization landscape will have a more well-defined minimum. $\boldsymbol{Q} \in \mathbb{R}^{2n \times 2n}$, $\boldsymbol{P} \in \mathbb{R}^{2n \times 2n}$ and $\boldsymbol{R} \in \mathbb{R}^{n \times n}$ are the diagonal and constant state weight matrix, terminal cost weight matrix and control input weight matrix, respectively. State and control input constraints are considered in Eq. (5.5c) and Eq. (5.5d) respectively, where $\boldsymbol{x}_{\text{min}}$ and $\boldsymbol{x}_{\text{max}}$ are the constant minimum and maximum state limits, respectively, and $\ddot{\boldsymbol{q}}_{\text{min}}$ and $\ddot{\boldsymbol{q}}_{\text{max}}$ are the minimum and maximum joint acceleration input limits, respectively. $\bar{\boldsymbol{x}}_0$ is the current sampled state vector. Since Eq. (5.5) is a linear MPC problem, the much used approach of using the unconstrained infinite horizon cost given by the solution of the discrete Ricatti equation Eq. (3.7) is used to find $\boldsymbol{P}$, given $\boldsymbol{Q}$ and $\boldsymbol{R}$.

An alternative to the feedback linearization in Eq. (5.1) is to include the forward dynamics Eq. (2.28) directly in the OCP, such that the motor torque $\boldsymbol{\tau}$ is the control input. This approach results in the following OCP:

$$\min_{\boldsymbol{x},\boldsymbol{\tau}} \quad \sum_{i=0}^{N-1} \left(\|\boldsymbol{x}_i - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \|\ddot{\boldsymbol{q}}_i\|_{\boldsymbol{R}_u}^2 + \|\boldsymbol{\tau}_i\|_{\boldsymbol{R}_\tau}^2 \right) + \|\boldsymbol{x}_N - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2 \tag{5.6a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{i+1} = \boldsymbol{f}_d(\boldsymbol{x}_i, \boldsymbol{\tau}_i), \quad i = 0, \dots, N-1, \tag{5.6b}$$

$$\boldsymbol{x}_{\text{min}} \leq \boldsymbol{x}_i \leq \boldsymbol{x}_{\text{max}}, \quad i = 0, \dots, N, \tag{5.6c}$$

$$\boldsymbol{\tau}_{\text{min}} \leq \boldsymbol{\tau}_i \leq \boldsymbol{\tau}_{\text{max}}, \quad i = 0, \dots, N-1, \tag{5.6d}$$

$$\ddot{\boldsymbol{q}}_{\text{min}} \leq \ddot{\boldsymbol{q}}_i \leq \ddot{\boldsymbol{q}}_{\text{max}}, \quad i = 0, \dots, N-1, \tag{5.6e}$$

$$\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0, \tag{5.6f}$$

where $\boldsymbol{f}_d$ refers to the discrete version of the forward dynamics $\boldsymbol{f}$ in Eq. (2.28). Furthermore, a regularizer term and constraint on the joint acceleration $\ddot{\boldsymbol{q}}$ are added, in order to generate smoother motion. $\boldsymbol{R}_u$ and $\boldsymbol{R}_\tau$ are then the constant and diagonal weight matrices on the joint acceleration and joint torque, respectively. $\boldsymbol{\tau}_{\text{min}}$ and $\boldsymbol{\tau}_{\text{max}}$ are the constant lower and upper limits on the torque input.

A primary concern with both the feedback linearization approach and the NMPC approach is that the underlying assumption is that the system dynamics are perfectly known. In a real system, model mismatch will always be present to some degree. For feedback linearization, this means Eq. (5.1) will map the desired joint acceleration to a wrong torque, which generates an error between the desired and actual joint acceleration. For NMPC the controller will plan a torque sequence that will not generate the same

motion as the prior model.

Therefore, a stochastic MPC problem with added GP disturbance dynamics will also be formulated, as discussed in Section 4.4. This has the potential of accounting for uncertain model parameters, as well as additional unmodeled system dynamics such as motor friction and other actuator dynamics.

In order to apply GP-MPC to robot manipulator trajectory tracking, the discretized vectorial double integrator dynamics are again considered, now with added disturbance and process noise:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{B}_d(\boldsymbol{d}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{w}_k). \tag{5.7}$$

Following Carron et al. 2019, every dimension of $\boldsymbol{d}$ are assumed to be an independent GP, i.e., $d_i \sim \mathcal{GP}(0, k_i)$, with SE covariance function k_i. Furthermore, i.i.d. Gaussian process noise $\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_w)$ is considered, with $\boldsymbol{\Sigma}_w = \text{diag}(\sigma_{w,1}^2, \ldots, \sigma_{w,n}^2) \in \mathbb{R}^{n\times n}$. Furthermore, the process noise matrix $\boldsymbol{B}_d \in \mathbb{R}^{2n\times n}$ is chosen to be

$$\boldsymbol{B}_d = \begin{bmatrix} \boldsymbol{0} \\ \Delta T \boldsymbol{I} \end{bmatrix}, \tag{5.8}$$

which incidentally coincides with the forward Euler discretization of the continuous $\boldsymbol{B}$ matrix. This choice of $\boldsymbol{B}_d$ was picked over the exact discretization of $\boldsymbol{B}$ in Eq. (5.4) for its simpler structure, such that the disturbance is only affects the discrete velocity states.

The stochastic OCP with GP dynamics can then be formulated as:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \sum_{i=0}^{N-1} \left(\|\boldsymbol{\mu}_i^x - \boldsymbol{r}_i^x\|_{\boldsymbol{Q}}^2 + \text{tr}(\boldsymbol{Q}\boldsymbol{\Sigma}_i^x) + \|\boldsymbol{u}_i\|_{\boldsymbol{R}}^2 \right) + \|\boldsymbol{\mu}_N^x - \boldsymbol{r}_N^x\|_{\boldsymbol{P}}^2 + \text{tr}(\boldsymbol{P}\boldsymbol{\Sigma}_N^x) \tag{5.9a}$$

$$\text{s.t.} \quad \boldsymbol{\mu}_{i+1}^x = \boldsymbol{A}\boldsymbol{\mu}_i^x + \boldsymbol{B}\boldsymbol{u}_i + \boldsymbol{B}_d\boldsymbol{\mu}^d(\boldsymbol{\mu}_i^x, \boldsymbol{u}_i), \quad i = 0, \ldots, N-1, \tag{5.9b}$$

$$\boldsymbol{\Sigma}_{i+1}^x = \widetilde{\boldsymbol{A}}_i \boldsymbol{\Sigma}_i \widetilde{\boldsymbol{A}}_i^\top, \quad i = 0, \ldots, N-1, \tag{5.9c}$$

$$\boldsymbol{\mu}_i \in \mathcal{Z}(\boldsymbol{\Sigma}_i^x), \quad i = 0, \ldots, N, \tag{5.9d}$$

$$\ddot{\boldsymbol{q}}_{\text{min}} \leq \boldsymbol{u}_i \leq \ddot{\boldsymbol{q}}_{\text{max}}, \quad i = 0, \ldots, N-1, \tag{5.9e}$$

$$\boldsymbol{\mu}_0^x = \bar{\boldsymbol{x}}_0, \tag{5.9f}$$

$$\boldsymbol{\Sigma}_0^x = \boldsymbol{0}, \tag{5.9g}$$

with the state covariance propagation Eq. (5.9c) as previously defined in Eq. (4.42), and $\mathcal{Z}$ defined in Eq. (4.51).

In Figure 5.1 a block diagram of the workflow of GP-MPC for robot manipulator trajectory tracking is shown. Given a prior model $\boldsymbol{f}$ and a dataset $\mathcal{D}$, the unknown disturbance function $\boldsymbol{d}$ is estimated offline using GP regression. This model is then used online for closed-loop control using GP-MPC to track the reference trajectory $\boldsymbol{r}^x$. In the figure, the system dynamics refer to the standard forward dynamics, yet it should be noted that additional unmodeled dynamics terms will likely also be present, which $\boldsymbol{d}$ then aims to model.

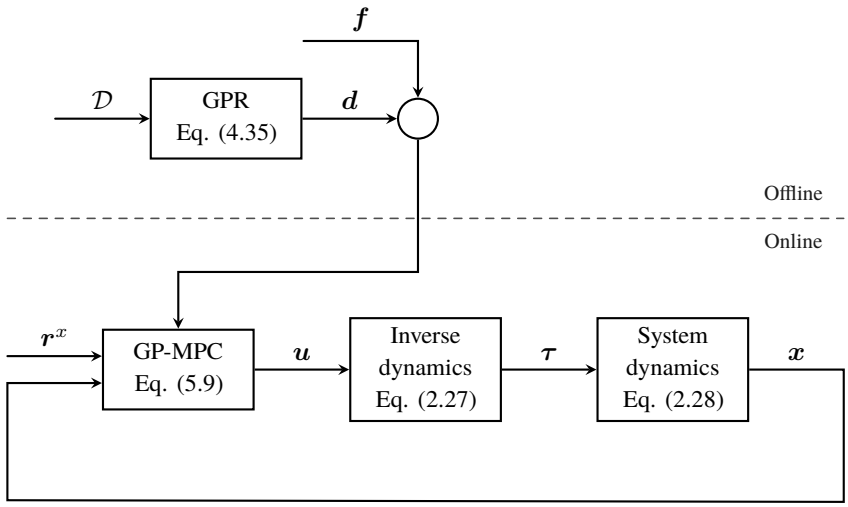


Figure 5.1: Block diagram of GP-MPC loop. GPR is done offline, given a dataset $\mathcal{D}$ and prior model $\boldsymbol{f}$. GP-MPC is used in closed loop given the desired trajectory $\boldsymbol{r}^x$ and the learned GP disturbance model $\boldsymbol{d}$.

5.1.2 Task space trajectory tracking[†]

For many applications, it would be useful to track end effector pose trajectories directly instead of trajectories in joint space. One approach is to use the inverse kinematics to map the desired end effector pose to joint angles and velocities and apply the previously

[†] This section is adapted from Brandt 2020.

discussed MPC approaches to track the computed joint space trajectory. However, it would also be of interest to consider formulating the cost function directly in task space, using the forward kinematics. This would avoid separating these two problems, which would be a sub-optimal strategy compared to considering the entire task space tracking problem directly as a single optimization problem. It would allow computation of the optimal control torques to track the pose trajectory while simultaneously considering secondary goals and constraints, e.g., minimizing the joint torques and accelerations or obstacle avoidance. This would be especially beneficial for redundant manipulator arms, where an infinite number of solutions exist to the inverse kinematics.

An in-depth discussion on different ways to formulate a task space trajectory tracking cost function for an NMPC controller is given in Brandt 2020, and in the following a summary is given. First let $\boldsymbol{r}^e(t) = [\, \boldsymbol{r}^p(t)^\top \;\; \boldsymbol{r}^o(t)^\top \,]^\top$ be the desired end effector pose trajectory, consisting of the position $\boldsymbol{r}^p(t) \in \mathbb{R}^3$ and the rotation quaternion $\boldsymbol{r}^o(t) \in \mathbb{R}^4$. The actual end effector position $\boldsymbol{h}^p(\boldsymbol{q})$, dependent on the joint angles $\boldsymbol{q}$, is given by 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. (2.20):

$$\begin{bmatrix} \boldsymbol{h}^p(\boldsymbol{q}) \\ 1 \end{bmatrix} = \boldsymbol{T}_e^b(\boldsymbol{q}) \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^\top. \tag{5.10}$$

The end effector orientation is given by the rotation matrix $\boldsymbol{R}_e^b$ in $\boldsymbol{T}_e^b(\boldsymbol{q})$, and is converted to a unit quaternion by Eq. (A.11). The end effector pose is then $\boldsymbol{h}^e(\boldsymbol{q}) = [\, \boldsymbol{h}^p(\boldsymbol{q}) \;\; \boldsymbol{h}^o(\boldsymbol{q}) \,]^\top$.

With the desired pose $\boldsymbol{r}^e(t)$ and actual pose $\boldsymbol{h}^e(\boldsymbol{q})$ defined, the pose trajectory tracking error can be formulated. The position error is trivially

$$\boldsymbol{e}^p = \boldsymbol{r}^p(t) - \boldsymbol{h}^p(\boldsymbol{q}). \tag{5.11}$$

For the orientation error, however, several possible formulations exists. Firstly, consider the quaternion error between $\boldsymbol{h}^o(\boldsymbol{q}) = [\, \eta_h \;\; \boldsymbol{\varepsilon}_h^\top \,]^\top$ and $\boldsymbol{r}^o(t) = [\, \eta_r \;\; \boldsymbol{\varepsilon}_r^\top \,]^\top$:

$$\delta \boldsymbol{h}^o = \boldsymbol{r}^o(t)^* \otimes \boldsymbol{h}^o(\boldsymbol{q}) = \begin{bmatrix} \eta_r \eta_h - \boldsymbol{\varepsilon}_r^\top \boldsymbol{\varepsilon}_h \\ \eta_r \boldsymbol{\varepsilon}_h - \eta_h \boldsymbol{\varepsilon}_r - [\boldsymbol{\varepsilon}_r]_\times \boldsymbol{\varepsilon}_h \end{bmatrix}. \tag{5.12}$$

As discussed in Siciliano et al. 2010, since the quaternion error approaches the identity unit quaternion $[\,\pm 1 \;\; 0 \;\; 0 \;\; 0\,]^\top$ for perfect tracking, one possibly orientation error formulation is simply the vector part of the quaternion error, i.e.,

$$\boldsymbol{e}^o = \eta_r \boldsymbol{\varepsilon}_h - \eta_h \boldsymbol{\varepsilon}_r - [\boldsymbol{\varepsilon}_r]_\times \boldsymbol{\varepsilon}_h. \tag{5.13}$$

From the unit length constraint on $\delta \boldsymbol{h}^o$ it is seen that $\boldsymbol{e}^o = \boldsymbol{0}$ will then track the orientation trajectory $\boldsymbol{r}^o$ exactly.

In order to formulate the task space trajectory tracking error cost, the full pose error is denoted as $\boldsymbol{e} = [\,(\boldsymbol{e}^p)^\top \;\; (\boldsymbol{e}^o)^\top\,]^\top$. The new stage cost for task space trajectory tracking is then

$$\ell_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = \|\boldsymbol{e}(\boldsymbol{x}_i)\|^2_{\boldsymbol{Q}} + \|\boldsymbol{u}_i\|^2_{\boldsymbol{R}}, \tag{5.14}$$

where the dependence on time is omitted for notational clarity. Here $\boldsymbol{Q} \in \mathbb{R}^6$ is the diagonal and constant pose error weight matrix. This cost function can be used directly in the MPC based on feedback linearization in Eq. (5.5), yet it should be emphasized that the nonlinear least squares cost turns the QP into a NLP, and will thus be harder to solve.

The task space cost Eq. (5.14) can also be used in the GP-MPC in Eq. (4.53), but it should be noted that this is under the assumption that the forward kinematics are known exactly and thereby deterministic. Furthermore, taking the expectation of a highly nonlinear function with respect to the random variable is generally very difficult, so in the following the certainty equivalence approach is taken by assuming that $\mathbb{E}\left[\ell_i(\boldsymbol{x}_i, \boldsymbol{u})\right] \approx \ell_i(\boldsymbol{\mu}_i, \boldsymbol{u})$. Assuming the kinematics are deterministic is a fair assumption, but the certainty equivalence principle does not apply here. It is, however, a necessary approximation to get a tractable OCP. Alternatives could be to try Monte Carlo approaches or polynomial chaos expansions to estimate the expectation, but this is not explored further in this work.

Finally, as explored in Brandt 2020, the task space trajectory tracking error can also be included in the NMPC problem in Eq. (5.6), such that the new cost is

$$\ell_i(\boldsymbol{x}_i, \boldsymbol{\tau}_i) = \|\boldsymbol{e}(\boldsymbol{x}_i)\|^2_{\boldsymbol{Q}} + \|\ddot{\boldsymbol{q}}_i\|^2_{\boldsymbol{R}_u} + \|\boldsymbol{\tau}_i\|^2_{\boldsymbol{R}_\tau}. \tag{5.15}$$

In Brandt 2020, several optional additional cost terms are discussed, one of which is the joint jerk. Penalizing the joint jerk $\dddot{\boldsymbol{q}}$ will force a smoother commanded acceleration profile and thus generate safer and more natural motions. This can be achieved with several

different approaches. One is to use finite differences to approximate the jerk at every time step. However, this is not always feasible, as many solvers optimized for real-time use will not allow cost terms with coupling between states for different time steps. One way to circumvent this is to augment the state vector by adding the previous input $\boldsymbol{u}_{i-1}$, such that costs and constraints can be formulated using the finite differences approximation of the joint jerk $\dot{\boldsymbol{u}}$. However, what will be used later in this work, is to augment the state by letting $\dot{\boldsymbol{u}}$ be the new input and formulate a cost term on that directly.

5.1.3 Slack variables[†]

Finally, we will consider extending the trajectory tracking OCPs in Section 5.1.1 and Section 5.1.2 to use slack variables where applicable. The main concept behind slack variables is that certain inequality constraints can be extended to be soft by adding slack variables $\boldsymbol{s}$, meaning that the constraints may be violated but at a high cost when violated. Consider the slack variable extension of a general inequality constraint:

$$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \leq 0 \quad \rightarrow \quad \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) - \boldsymbol{s} \leq 0, \quad \boldsymbol{s} \geq 0, \tag{5.16}$$

where $\boldsymbol{s} \in \mathbb{R}^{n_s}$. By adding a linear and quadratic cost on the slack variable in the cost function, i.e.,

$$\ell_s(\boldsymbol{s}) = \boldsymbol{z}^\top \boldsymbol{s} + \boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{s}, \tag{5.17}$$

with $\boldsymbol{z} \in \mathbb{R}^{n_s}$ and $\boldsymbol{W} \in \mathbb{R}^{n_s \times n_s}$ being the linear and quadratic slack weights, respectively, one can improve the feasibility of the OCPs, as the MPC problem will no longer be infeasible if the system enters a state outside the original feasible set. Considering factors such as uncertain dynamics, measurement noise, and feedback delay, this is likely to happen in a practical setting.

For the trajectory tracking MPC problem specifically, the chosen limits for $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are typically far lower than their actual maximum values. The slack variable principle can then be applied to make the limits on $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ soft, such that $\boldsymbol{s} \in \mathbb{R}^{2n}$ in this case. However, this approach does come at the cost of adding additional optimization variables and complicating the tuning process with more tuning parameters, namely $\boldsymbol{z}$ and $\boldsymbol{W}$.

[†] This section is adapted from Brandt 2020.

5.2 Trajectory blending[†]

In the previous section, trajectory tracking formulations of the MPC problem for robot manipulators were considered. However, for real-world applications, the desired trajectory might be far away from the initial joint configuration $\boldsymbol{q}_0$, which could result in an unwanted transient before the substantial initial tracking error has been reduced. It is, therefore, considered how to provide a smooth approach from $\boldsymbol{q}_0$ to 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. It is desirable to smoothly and safely approach the target, especially in a collaborative setting. Feeding the target trajectory directly into the trajectory tracking controller will create a significant error signal, which might lead to overshoot. In this work, trajectory blending methods are applied to solve this problem of generating smooth motion from one trajectory to another. Note that other approaches could be considered as well, such as filtering the desired trajectory.

5.2.1 Position trajectory blending

The position $\boldsymbol{h}^p$ is blended from the initial position $\boldsymbol{p}_0$ to the desired position trajectory $\boldsymbol{r}^p$ by the interpolation

$$\boldsymbol{h}^p(s) = \boldsymbol{p}_0 + \beta(s)(\boldsymbol{r}^p - \boldsymbol{p}_0) \tag{5.18}$$

at every time step. The characteristics of the blended motion are determined by the monotonically increasing function $\beta(s) : [0, 1] \mapsto [0, 1]$. The simple case $\beta(s) = s$ results in standard linear interpolation. The first such function that will be considered is the logistic function

$$\beta_l(s) = \frac{a}{1 + e^{-ks}} - b, \tag{5.19}$$

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

$$a = 2\,\frac{1 + e^{-k}}{1 - e^{-k}}, \quad b = \frac{1 + e^{-k}}{1 - e^{-k}}. \tag{5.20}$$

[†] This section is adapted from Brandt 2020.

This choice of blending function allows linear motion towards the target initially, and then the motion will slow down as the end effector approaches the target. One downside of this choice of $\beta(s)$ is that it will result in a step in the initial velocity, which consequently means an initial error spike.

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

$$\beta_e(s) = c\left(1 - e^{-(\alpha s)^3}\right) \tag{5.21}$$

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 $\beta_e(1) = 1$. It can be shown that this is achieved for

$$c = \frac{1}{1 - e^{-\alpha^3}}. \tag{5.22}$$

The two blending shape functions are compared for different values of k and α in Figure 5.2. Quintic polynomials are also frequently used for blending references, for instance, as considered in Macfarlane and Croft 2003.

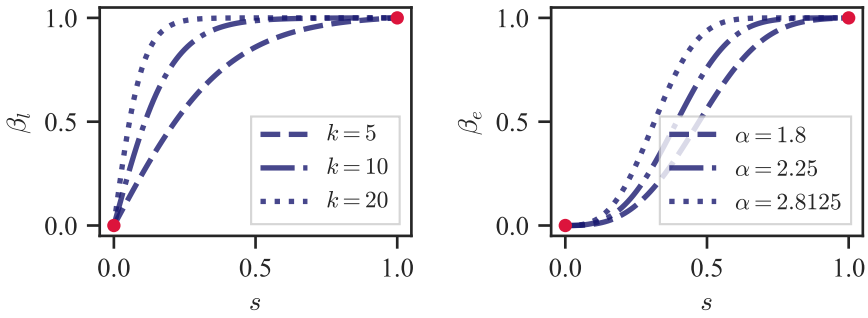


Figure 5.2: Logistic function β_l and third-order exponential β_e for different values of k and α.

It should be noted that Eq. (5.18) can also be used for blending trajectories in joint space, for the same reason of having an initial smooth motion from the initial joint configuration with zero velocity to the desired trajectory. Since the model-predictive trajectory tracking controller in Eq. (5.5) naturally considers both a joint angle and a joint velocity reference,

the desired velocity must also be derived. This can be found to be:

$$\dot{\boldsymbol{h}}_p(s) = \dot{\beta}(s)(\boldsymbol{r}^p - \boldsymbol{p}_0) + \beta(s)\dot{\boldsymbol{r}}^p, \tag{5.23}$$

where $\dot{\beta} = \frac{\mathrm{d}\beta}{\mathrm{d}s}\dot{s}$ can be used. For the third-order exponential case, it can be shown that $\frac{\mathrm{d}\beta}{\mathrm{d}s} = 3c\alpha^3 s^2 e^{-(\alpha s)^3}$.

5.2.2 Quaternion trajectory blending

Blending orientation trajectories is not quite as trivial as blending position trajectories. The linear interpolation expression Eq. (5.18) cannot be applied directly since the rotation quaternion is constrained by the unit length constraint. One option is to normalize the quaternion after doing linear interpolation, as introduced in Kavan and Žára 2005, which is referred to as Quaternion Linear Blending (QLB). This approach takes the shortest possible path but does not result in a constant angular velocity for a linear β. In this work, spherical linear interpolation (SLERP) is used, which linearly interpolates between the initial quaternion and the desired quaternion along the 4-dimensional unit hypersphere. It is defined by

$$\boldsymbol{h}^o(s) = \boldsymbol{q}_0 \otimes (\boldsymbol{q}_0^{-1} \otimes \boldsymbol{r}^o)^{\,\beta(s)}, \tag{5.24}$$

using the quaternion product, power, and inverse operations, as defined in Section 2.1.4. The resulting transform results in a rotation with a constant angular velocity around a fixed axis for $\beta(s) = s$. This is visualized in Figure 5.3, projected down in 3 dimensions. Again by choosing a smooth, monotonically increasing $\beta(s)$ such as Eq. (5.19) or Eq. (5.21) smooth motion can be generated between the initial and desired orientation.

An equivalent expression for SLERP given in Solà 2017, which is more suited for implementation in code, is

$$\boldsymbol{h}^o(s) = \frac{\sin((1-\beta(s))\Delta\theta)}{\sin(\Delta\theta)}\boldsymbol{q}_0 + \frac{\sin(\beta(s)\Delta\theta)}{\sin(\Delta\theta)}\boldsymbol{r}^o, \tag{5.25}$$

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

In Figure 5.4 a simple example is considered, where a periodic position trajectory is tracked while keeping a fixed orientation reference. Both the logistic function Eq. (5.19) and the third-order exponential Eq. (5.21) are studied. From the resulting blended reference

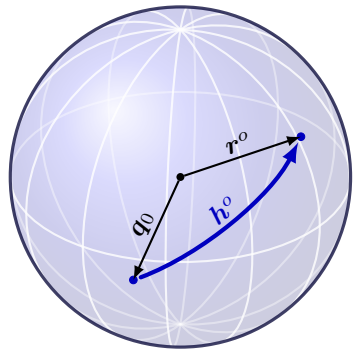


Figure 5.3: SLERP from q_0 to r^o along unit hypersphere, projected to 3 dimensions.

trajectories, it is seen how the logistic function will result in a nonzero initial target velocity. In contrast, the exponential function will start with zero velocity.

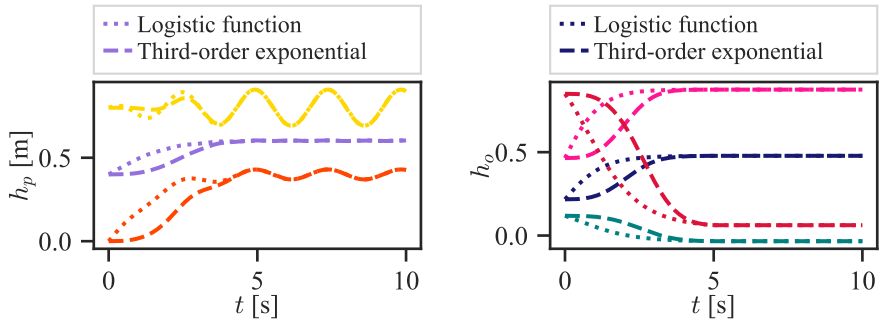


Figure 5.4: Blended position and orientation trajectories for a periodic motion with fixed orientation reference. The blending is done by linear interpolation of position and SLERP interpolation of the quaternion, where both a logistic function blend and a third-order exponential blending is shown.

5.3 Space manipulator modeling and control

In the following section the kinematics and dynamics of space manipulators will be presented, based on Wilde et al. 2018 and Nanos and Papadopoulos 2017. The resulting model will be applied to an NMPC controller, as well as a GP-based MPC.

5.3.1 Space manipulator kinematics and dynamics

The space manipulator system shown in Figure 5.5 is considered. It consists of a satellite base with position $\boldsymbol{r}_s^I$ and attitude quaternion $\boldsymbol{Q}_s^I = [\,\eta_s \;\; \boldsymbol{\varepsilon}_s^\top\,]^\top$ in inertial frame, and an n DOF manipulator arm with joint angles $\boldsymbol{q}$. The satellite base has mass m_s and inertia $\boldsymbol{I}_s$, and the mass and inertia of the i-th link are denoted as m_i and $\boldsymbol{I}_i$, where $i \in \mathbb{N}_n$.

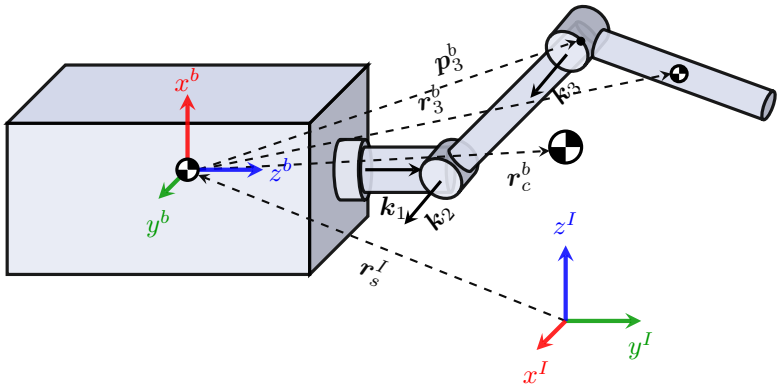


Figure 5.5: Space manipulator system, with inertial frame coordinate system and satellite body frame coordinate system shown. The satellite body center of mass, link center of mass, and total space manipulator system center of mass is indicated.

Let $\boldsymbol{p}_i^I$ be the position vector to the i-th joint, and let $\boldsymbol{r}_i^I$ be the position of the center of mass of the i-th link, both from the satellite body frame. The corresponding joint-fixed and link-fixed coordinate systems are referred to as j_i and ℓ_i, respectively. Finally, the rotation matrix from the i-th link-fixed frame to body-fixed frame is denoted as $\boldsymbol{R}_{\ell_i}^b$. $\boldsymbol{p}_i^I$, $\boldsymbol{r}_i^I$, and $\boldsymbol{R}_{\ell_i}^b$ are found by formulating the forward kinematics of the system using homogeneous transformation matrices as discussed in Section 2.2.

The joint axis in body frame is denoted as $\boldsymbol{k}_i^b$, given by the transformation

$$\boldsymbol{k}_i^b = \boldsymbol{R}_{j_i}^b \boldsymbol{k}_i^{j_i}, \tag{5.26}$$

where $\boldsymbol{k}_i^{j_i}$ is the i-th joint axis in the local joint frame, of unit length. In the DH convention the local joint axes are always along the local z-axis.

Finally, the vector from the satellite body center of mass to the space manipulator center

of mass is defined as

$$\boldsymbol{r}_c^b = \frac{1}{m_{\text{tot}}} \sum_{i=1}^{n} m_i \boldsymbol{r}_i^I, \tag{5.27}$$

where $m_{\text{tot}} = m_s + \sum_{i=1}^{n} m_i$ is the total mass of the space manipulator system.

The Lagrangian formalism can then be used to derive the system dynamics. Following Wilde et al. 2018, $[\,(\boldsymbol{x}_s^b)^\top \;\; \boldsymbol{q}^\top\,]^\top$ are used as the generalized coordinates, where $\boldsymbol{x}_s^b = [\,(\boldsymbol{r}_s^b)^\top \;\; (\boldsymbol{Q}_s^b)^\top\,]^\top$ is the satellite body pose in body frame. It is assumed that there are no external forces or torques on the system, such that the potential energy is zero. Furthermore, it is assumed that the satellite is free floating, i.e., that only the joint motors are controlled and not the satellite body states. The Lagrangian of the system $\mathcal{L}$ is therefore equal to the kinetic energy T and is given in Wilde et al. 2018 as:

$$\mathcal{L} = T = \frac{1}{2} \begin{bmatrix} (\dot{\boldsymbol{x}}_s^b)^\top & \dot{\boldsymbol{q}}^\top \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{M}_s & \boldsymbol{M}_{sm} \\ \boldsymbol{M}_{sm}^\top & \boldsymbol{M}_m \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_s^b \\ \dot{\boldsymbol{q}} \end{bmatrix}, \tag{5.28}$$

where the satellite mass matrix $\boldsymbol{M}_s$ is

$$\boldsymbol{M}_s = \begin{bmatrix} \boldsymbol{M}_v & \boldsymbol{M}_{v\omega} \\ \boldsymbol{M}_{v\omega}^\top & \boldsymbol{M}_\omega \end{bmatrix}, \tag{5.29}$$

with

$$\boldsymbol{M}_v = m\boldsymbol{I}, \quad \boldsymbol{M}_{v\omega} = -m[\boldsymbol{r}_c]_\times, \tag{5.30}$$

$$\boldsymbol{M}_\omega = \boldsymbol{I}_s^b + \sum_{i=1}^{n} \left(\boldsymbol{I}_i^b - m_i [\boldsymbol{r}_i]_\times [\boldsymbol{r}_i]_\times \right). \tag{5.31}$$

It is important to note that the link center of mass inertia matrices $\boldsymbol{I}_i^b$ are given in body frame, and can be transformed from the local link frames by

$$\boldsymbol{I}_i^b = \boldsymbol{R}_{\ell_i}^b \boldsymbol{I}_i^{\ell_i} (\boldsymbol{R}_{\ell_i}^b)^\top. \tag{5.32}$$

The mass matrix for the coupling between the manipulator and the satellite body $\boldsymbol{M}_{sm}$ is

given by

$$\boldsymbol{M}_{sm} = \begin{bmatrix} \boldsymbol{M}_{vm} \\ \boldsymbol{M}_{\omega m} \end{bmatrix}, \tag{5.33}$$

where

$$\boldsymbol{M}_{vm} = \sum_{i=1}^{n} m_i \boldsymbol{J}_{T,i}, \quad \boldsymbol{M}_{\omega m} = \sum_{i=1}^{n} \left(\boldsymbol{I}_i^b \boldsymbol{J}_{R,i} + m_i [\boldsymbol{r}_i]_\times \boldsymbol{J}_{T,i} \right). \tag{5.34}$$

$\boldsymbol{J}_{T,i}$ is the translational part of the Jacobian for the center of mass of link i in body frame:

$$\boldsymbol{J}_{T,i} = \begin{bmatrix} [\boldsymbol{k}_1^b]_\times (\boldsymbol{r}_i - \boldsymbol{p}_1) & \cdots & [\boldsymbol{k}_i^b]_\times (\boldsymbol{r}_i - \boldsymbol{p}_i) & \boldsymbol{0}_{3,n-i} \end{bmatrix}, \tag{5.35}$$

and $\boldsymbol{J}_{R,i}$ is the corresponding rotational part:

$$\boldsymbol{J}_{R,i} = \begin{bmatrix} \boldsymbol{k}_1^b & \cdots & \boldsymbol{k}_i^b & \boldsymbol{0}_{3,n-i} \end{bmatrix}, \tag{5.36}$$

as derived in Siciliano et al. 2010 and similar to the end effector Jacobian discussed earlier in Section 2.3. Finally the manipulator mass matrix $\boldsymbol{M}_m$ is given by

$$\boldsymbol{M}_m = \sum_{i=1}^{n} \left(\boldsymbol{J}_{R,i}^\top \boldsymbol{I}_i^b \boldsymbol{J}_{R,i} + m_i \boldsymbol{J}_{T,i}^\top \boldsymbol{J}_{T,i} \right). \tag{5.37}$$

The linear and angular momenta of the space manipulator system around the system center of mass are

$$\begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{l} \end{bmatrix} = \boldsymbol{M}_s \dot{\boldsymbol{x}}_s^b + \boldsymbol{M}_{sm} \dot{\boldsymbol{q}}, \tag{5.38}$$

and are conserved given the assumption of a free floating space manipulator system. By assuming zero initial momentum, i.e., $\boldsymbol{p}_0 = \boldsymbol{l}_0 = \boldsymbol{0}$, one can show as in Wilde et al. 2018 that the reduced equations of motion are given by

$$\boldsymbol{M}^*(\boldsymbol{q}) \ddot{\boldsymbol{q}} + \boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} = \boldsymbol{\tau}, \tag{5.39}$$

where only $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ now appear in the dynamics. The reduced mass matrix $\boldsymbol{M}^*$ is given

by

$$\boldsymbol{M}^* = \boldsymbol{M}_m - \boldsymbol{M}_{sm}^\top \boldsymbol{M}_s^{-1} \boldsymbol{M}_{sm}, \tag{5.40}$$

and the reduced Coriolis and centripetal matrix have several forms, one of which is

$$\begin{aligned} \boldsymbol{C}^* = & \frac{\partial(\boldsymbol{M}_s \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}} - \frac{1}{2} \left(\frac{\partial(\dot{\boldsymbol{q}}^\top \boldsymbol{M}_s)}{\partial \boldsymbol{q}} \right)^\top \\ & + \frac{1}{2} \left(\frac{\partial(\dot{\boldsymbol{q}}^\top \boldsymbol{M}_{sm}^\top \boldsymbol{M}_s^{-1} \boldsymbol{M}_{sm})}{\partial \boldsymbol{q}} \right)^\top - \frac{\partial(\boldsymbol{M}_{sm}^\top \boldsymbol{M}_s^{-1} \boldsymbol{M}_{sm} \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}}, \end{aligned} \tag{5.41}$$

as given in Nanos and Papadopoulos 2017.

The reduced equations of motion in Eq. (5.39) are written in the same convenient form as the fixed-base manipulator dynamics in Eq. (2.26), and provide the basis for simulation and control of space manipulators with zero initial momenta. It would however be useful to retrieve back the satellite base velocity and pose states. Under the assumption of zero initial momenta, the linear and angular velocity in body frame can be retrieved by solving Eq. (5.38) for $\dot{\boldsymbol{x}}_s^b$:

$$\dot{\boldsymbol{x}}_s^b = \begin{bmatrix} \boldsymbol{v}_s^b \\ \boldsymbol{\omega}_s^b \end{bmatrix} = -\boldsymbol{M}_s^{-1} \boldsymbol{M}_{sm} \dot{\boldsymbol{q}}. \tag{5.42}$$

The body velocity in body-fixed frame can be transformed to inertial frame by:

$$\begin{bmatrix} \dot{\boldsymbol{r}}_s^I \\ \dot{\boldsymbol{Q}}_s^I \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_b^I(\boldsymbol{Q}_s^I) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}(\boldsymbol{Q}_s^I) \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_s^b \\ \boldsymbol{\omega}_s^b \end{bmatrix}, \tag{5.43}$$

where

$$\boldsymbol{E}(\boldsymbol{Q}_s^I) = \frac{1}{2} \begin{bmatrix} -\boldsymbol{\varepsilon}_s^I & \eta_s^I \boldsymbol{I} + [\boldsymbol{\varepsilon}_s^I]_\times \end{bmatrix}^\top \tag{5.44}$$

provides the transformation from angular velocity to the unit quaternion derivative, as given in Siciliano et al. 2010.

5.3.2 Space manipulator MPC

Various approaches have been proposed for control of free-floating space manipulator systems. One frequently used approach is to formulate the generalized Jacobian for the space manipulator system, as introduced in Umetani and Yoshida 1989, such that traditional

inverse kinematics methods, like the Jacobian inverse discussed in Section 2.3, can be applied to track the desired trajectory. Alternatively, one can do feedback linearization of Eq. (5.39), similar to fixed-base systems as discussed earlier in Section 5.1. Then PD control laws can be formulated for the acceleration input, possibly with acceleration feedforward, which is done in Wilde et al. 2018. Applying such a feedback linearization with MPC is considered in Wang et al. 2016. Alternatively, NMPC with the full nonlinear dynamics of the system is used in Rybus et al. 2017. In the following, we consider the feedback linearized system as in Wang et al. 2016, but extend the model with a GP disturbance model.

The system is feedback linearizated by letting

$$\boldsymbol{\tau} = \boldsymbol{M}^*(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}, \tag{5.45}$$

such that a double integrator system is obtained with the joint acceleration $\boldsymbol{u}$ as the control input, like in Eq. (5.3). Then a linear MPC can be used to sample a dataset, such that a GP can be trained and used in GP-MPC control of the space manipulator system, in the presence of unmodeled dynamics. This might include unknown joint damping, uncertain base mass from thrusters using fuel, or unknown end effector mass from gripping space debris or other satellites.

For joint space trajectory tracking, the same MPC problem as previously considered with a fixed-base manipulator in Eq. (5.5) is used. For task space trajectory tracking, it is important to note that an additional transform from body-fixed frame to inertial frame is needed. For fixed-base systems, this transform is static or is even assumed to coincide with the inertial frame, such that this is not a problem. However, for floating-base systems, this means the pose of the robot body needs to be known in the inertial frame over the MPC time horizon, meaning that the position and orientation dynamics need to be added to the optimization problem. This increases computational complexity and the dimensionality of the problem significantly. For instance, for the 3 DOF example, the number of optimization variables increases from 6 to 18, and the dynamics will be significantly more involved. While this is still possible to do, as is done in Rybus et al. 2017, a different approach is taken here to remedy this complication.

Is it assumed that since the initial momenta of the system are zero, the pose of the satellite body will not change significantly over the current prediction horizon. This

assumption is also supported by the fact that in a real-world use case, the mass of the satellite body will be significantly larger than the total mass of the manipulator arm. Given this assumption, the satellite body position $\boldsymbol{r}_s^I$ and orientation $\boldsymbol{Q}_s^I$ can be considered fixed over the current MPC time horizon and is therefore treated as constant input parameters to the MPC solver at every time step. The end effector pose in inertial frame can then be formulated by adding one additional transform from inertial frame to body frame $\boldsymbol{T}_b^I(\boldsymbol{r}_s^I, \boldsymbol{Q}_s^I)$ to the chain of homogeneous transformation matrix products in Eq. (2.20), such that the final homogeneous transformation matrix is

$$\boldsymbol{T}_e^I = \boldsymbol{T}_b^I(\boldsymbol{r}_s^I, \boldsymbol{Q}_s^I)\, \boldsymbol{T}_e^b(\boldsymbol{q}). \tag{5.46}$$

By using this transformation in the cost function the task space MPC discussed in Section 5.1.2 can be applied. To the best of the author's knowledge, this approximation has not previously been used to simplify the space manipulator trajectory tracking MPC problem.

5.4 Software tools[†]

In this section, the practical implementation of the GP-MPC and NMPC controllers, and the software tools used to implement them, will briefly be outlined. The implementation was written in Python and uses the optimal control library acados presented in Verschueren et al. 2019 to generate a real-time feasible SQP RTI solver, given the robot kinematics and dynamics, and the hyperparameters of a GP. The kinematics and prior dynamics can either be given manually or generated using the library urdf2casadi by johannessen2019robot, given a robot model. The hyperparameters of the GP are trained using GPflow, presented in Matthews et al. 2017. The basic workflow for the GP-MPC implementation is illustrated in Figure 5.6. These tools and various notable implementation details will be further explained in the following.

[†] This section is adapted from Brandt 2020.

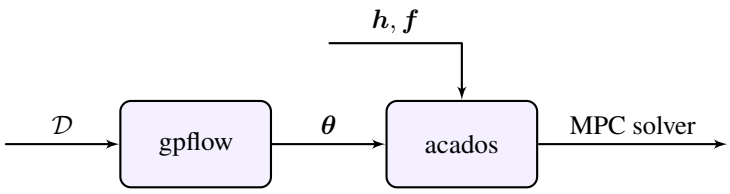


Figure 5.6: Workflow of the GP-MPC implementation for robot manipulator trajectory tracking. $\boldsymbol{h}$ and $\boldsymbol{f}$ are the robot kinematics and dynamics, respectively, and $\boldsymbol{\theta}$ are the hyperparameters that parametrize the GP disturbance model.

5.4.1 GPflow

The Python package GPflow was used for learning the GP hyperparameters, both with maximum likelihood for exact hyperparameter inference in Eq. (4.21), and for the sparse FITC, VFE and SVGP methods, described in Section 4.3. Alternatives include the Python libraries GPy by GPy 2021, GPyTorch by Gardner et al. 2018, and scikit-learn by Pedregosa et al. 2011. GPflow was chosen since it has simple interfaces to all the aforementioned sparse methods and is built on the machine learning TensorFlow given in Abadi et al. 2016, which allows automatic gradients and GPU training. Tensorflow provides, among others, the Adam optimizer from Kingma and Ba 2015, which was used for training the GPs. Initial testing with the Adam optimizer on the GPU showed an increase in training time compared to the CPU for the hardware and datasets used in this work. This is likely because the dataset sizes and the number of variables were not large enough to use GPU training efficiently. The GPs were therefore trained only on the CPU in this work. The hardware used for training is mentioned later in Section 6.1.

5.4.2 acados

The MPC formulations described in Section 5.1 were implemented using the optimal control library acados. 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.

Furthermore, it allows formulating the dynamics, cost, and constraints as CasADi expressions. CasADi is a framework for nonlinear numerical optimization based on generating expression graphs from symbolic expressions in order to compute algorithmic

derivatives. It allows for easily formulating symbolic expressions using trigonometric functions, matrix multiplication, trace, matrix inverse, and matrix decomposition like Cholesky decomposition, and is therefore very useful for formulating the robot kinematics, dynamics, and the GP equations like the prediction equations in Eq. (4.35).

Alternatives for real-time optimal control frameworks include the predecessor of acados, ACADO by Houska et al. 2011, and FORCES PRO by Zanelli et al. 2017. The latter was also tested for this work, but initial, rudimentary tests found acados to be faster and more stable for NMPC with GP dynamics. It was, however, used for the GP-MPC implementations in Hewing, Liniger, et al. 2018, Hewing, Kabzan, et al. 2019 and Carron et al. 2019, and is as such most certainly viable for GP-MPC with complex dynamics and high sample rates.

5.4.3 urdf2casadi

urdf2casadi is a Python package for generating the kinematics and dynamics of robot systems as CasADi expressions, given a Unified Robot Description Format (URDF) model of the robot. The URDF format is an XML file format for representing robot models, used in Robot Operating System (ROS). urdf2casadi was used to generate the robot kinematics and dynamics and include them into the OCP formulation in acados. There are alternatives, like Pinocchio by Carpentier et al. 2019, but urdf2casadi was chosen for the ease of integration with acados using CasADi expressions.

5.4.4 PyBullet

The Python package PyBullet from Coumans and Bai 2016–2021 was used to test the developed MPC controllers in simulation. PyBullet lets the user load robots into the simulation environment from URDF and simulates the robot dynamics with joint limits while allowing joint position, joint velocity, and joint torque control. Many other robotics simulation tools exists, such as Gazebo by Koenig and Howard 2004, CoppeliaSim by Rohmer et al. 2013, and MuJoCo by Todorov et al. 2012. PyBullet was used since it is open-source and most of the underlying model terms, like the Jacobian and mass matrix, are accessible.

6 Results

In this chapter, results from simulations and experiments of deterministic NMPC and GP-MPC for both fixed-base and floating-base robot manipulator systems will be presented and analyzed. Firstly, in order to illustrate the process of sampling a dataset, training a GP and controlling a robot using GP-MPC, a 2 DOF planar manipulator is considered. During this section, the hardware and various configurations used for training and simulation for all the results are given. Results from experiments with a fixed-base 6 DOF UR10e robot manipulator and a free-floating space manipulator robot are then presented. Both of these systems were tested in simulation, and for the UR10e, the trajectory tracking methods were also tested on a real robot. For all tests, both joint space and task space trajectories were tested. The joint space results on the UR10e robot in the lab and the task space result on the space manipulator simulation are given as supplementary results in Appendix D.

6.1 Trajectory tracking for 2 DOF planar robot manipulator

In this section, trajectory tracking for the 2 DOF planar robot manipulator arm depicted in Figure 6.1 is considered as an initial illustrative example. Modeling of the planar manipulator is given in detail in Appendix C but follows the modeling principles for kinematics and dynamics outlined in Chapter 2. The DH parameters for the robot manipulator arm are given in Table 6.1, and the parameters defining the links and joints of the robot are given in Table 6.2 and Table 6.3, respectively.

The linear MPC using feedback linearization in Eq. (5.5) is first used, in order to collect a dataset to learn the disturbance dynamics using GPR, as well as to serve as a benchmark for comparison with the GP-MPC controller in Eq. (5.9). It is assumed that the mass and inertia of the links are not perfectly known, such that the prior model used by the linear

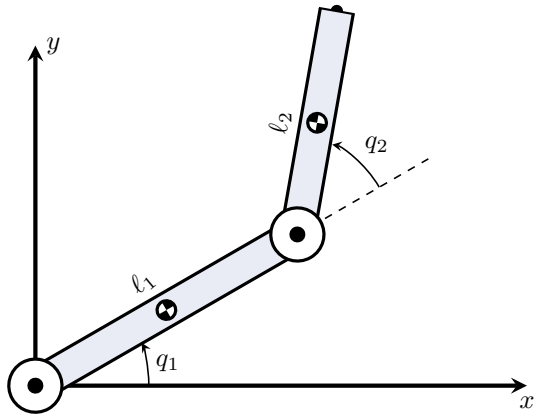


Figure 6.1: 2 DOF planar robot manipulator arm, with joint angles q_1 and q_2, and link lengths ℓ_1 and ℓ_2.

Table 6.1: DH parameters for the 2 DOF planar manipulator.

Link	a_i	α_i	d_i	θ_i
1	ℓ_1	0	0	q_1
2	ℓ_2	0	0	q_2

MPC differs from the true model used in simulation. Specifically, as seen in Table 6.2, the prior mass and inertia of link 1 are scaled down by 20%, and the mass and inertia of link 2 are scaled up by 25%. While such high uncertainty in the dynamical parameters of the model is unrealistic for most applications, it is done here to illustrate the degradation of tracking performance with model mismatch and the capability of GP-MPC to account for this model mismatch.

In the following an ERK4 integrator with step size $\Delta t = 1\,\mathrm{ms}$ was used to simulate the system. Zero mean Gaussian measurement noise was added to the joint velocity measurements with standard deviation $\sigma_n = 2 \cdot 10^{-4}\,\mathrm{rad/s}$, which was used for all subsequent simulations in the chapter. The MPC controllers were configured with $N = 24$ steps and sampling time $t_s = 10\,\mathrm{ms}$. In the following presented results, the acados SQP RTI solver was used to solve the presented OCPs, using the multiple-shooting method. The Hessian was approximated using the Gauss-Newton approximation. The resulting

Table 6.2: Link parameters for the 2 DOF planar manipulator. $\hat{m}$ and $\hat{I}_{zz}$ are the prior mass and inertia, respectively.

Link	ℓ [m]	m [kg]	$\hat{m}$ [kg]	I_{zz} [kgm^2]	$\hat{I}_{zz}$ [kgm^2]
1	0.5	5.0	4.0	$6.25 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$
2	0.5	5.0	6.25	$6.25 \cdot 10^{-3}$	$7.813 \cdot 10^{-3}$

Table 6.3: Joint parameters for 2 DOF planar manipulator.

Joint	$q_{\max}$ [rad]	$\dot{q}_{\max}$ [rad/s]	$\ddot{q}_{\max}$ [rad/s^2]	F_v [Nms/rad]
1	π	1.0	8.0	1.5
2	π	1.0	8.0	1.5

QPs were solved with KKT tolerance $1 \cdot 10^{-5}$, using the interior point QP solver HPIPM by Frison and Diehl 2020. All subsequent simulations and experiments were run on a computer with an AMD Ryzen 9 3900X CPU with 16 GB RAM.

6.1.1 Linear MPC

The feedback linearization-based MPC Eq. (5.5) is first used to collect a training set and a test set in order to do GPR. For the training set a planar trefoil knot curve is considered:

$$\boldsymbol{r}_1^e(t) = a_1 \begin{bmatrix} \sin(\omega_1 t) + 2\sin(2\omega_1 t) \\ \cos(\omega_1 t) - 2\cos(2\omega_1 t) \end{bmatrix} + \bar{\boldsymbol{x}}_1, \tag{6.1}$$

with parameters $a_1 = 0.14, \omega_1 = 0.1$ and shifted by $\bar{\boldsymbol{x}}_1 = [\,0.9 \;\; 1.1\,]^\top$. For the test set a Lissajous curve is considered:

$$\boldsymbol{r}_2^e(t) = \begin{bmatrix} a_2 \cos(\omega_2 t + \varphi) \\ b_2 \sin(n\omega_2 t) \end{bmatrix} + \bar{\boldsymbol{x}}_2, \tag{6.2}$$

with $a_2 = 0.4, b_2 = 0.2, \omega_2 = 1, \varphi = \frac{\pi}{2}, n = 1.5$ and shifted by $\bar{\boldsymbol{x}}_2 = [\,0.9 \;\; 1.2\,]^\top$. Both trajectories are shown in Cartesian space in Figure 6.2. The end effector trajectories are mapped to joint angle and joint velocity trajectories by the inverse kinematics and the

Jacobian, found in Eq. (C.2) and Eq. (C.3), respectively.

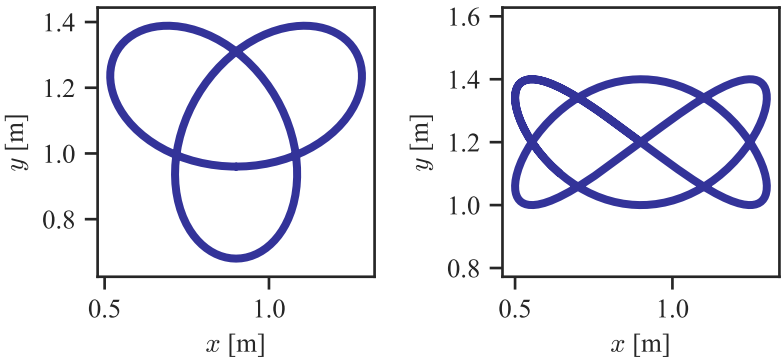


Figure 6.2: Trefoil knot curve $\boldsymbol{r}_1^e(t)$ (left) and Lissajous curve $\boldsymbol{r}_2^e(t)$ (right) in the xy-plane.

The system was simulated for $T = 10\,\mathrm{s}$, with simulation and control parameters such as tuning and initial conditions given in Table 6.4. Chance constraint on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ were added, as well as deterministic constraints on $\boldsymbol{u}$, with the limits given earlier in Table 6.3. The constraint violation probability for the two-sided bounds on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ was set to $\epsilon = 0.0456$, which corresponds to a 2-sigma probability in the one-dimensional case. However, the state is naturally a multivariate Gaussian, so this does not directly translate to the multivariate case, as rather the sum of the violation probabilities for every constraint adds up to this 2-sigma probability, as explained in Section 4.4.5. However, it is a useful way to interpret the constraint violation probability during the tuning process. For all subsequent tests, the initial joint velocities are zero. The discrete Ricatti equation Eq. (3.7) was used to find the terminal cost $\boldsymbol{P}$.

In Figure 6.3 the computed joint acceleration and corresponding control torque is shown, the latter computed using the inverse dynamics prior model. The resulting state trajectory is presented with respect to time in Figure 6.4. The end effector trajectory in the xy-plane is visualized in Figure 6.5, with the blue and orange lines indicating joint 1 and joint 2 at certain equally spaced steps. A substantial tracking error is seen in Figure 6.6, since there is model mismatch present in the system. In Figure 6.7a the computation time is shown. Finally, the computation time of the preparation stage and the feedback stage are shown separately in Figure 6.7b, where in this example with a linear MPC the time taken by the two stages of the RTI algorithm are comparable in length.

Table 6.4: Parameter values for joint space trajectory tracking test with 2 DOF planar manipulator.

Parameter	Value
N	24
t_s [ms]	10.0
$\boldsymbol{q}_0$ [deg]	$[10, 75]$
ϵ	0.0456
$\boldsymbol{Q}$	$\text{blkdiag}(10^2 \cdot \boldsymbol{I}_2, 10 \cdot \boldsymbol{I}_2)$
$\boldsymbol{R}$	$\boldsymbol{I}_2$

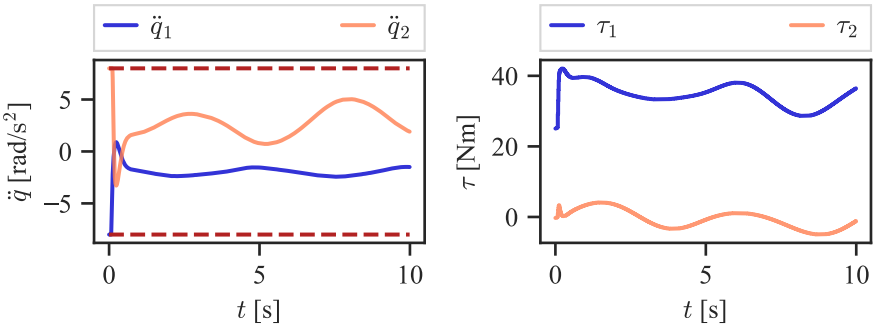


Figure 6.3: Joint acceleration input and computed joint torque for 2 DOF planar manipulator following joint space trajectory with linear MPC.

A training dataset was collected, with sample rate matching the MPC time step of 10 ms. The datasets for all subsequent tests were collected with a sample rate matching the MPC time step. For this example, that resulted in a dataset size of 1000 data points.

Another test was done to collect a test set, now tracking the Lissajous curve shown to the right in Figure 6.2. This dataset was collected over 15 s, resulting in 1500 samples, and with initial joint configuration $\boldsymbol{q}_0 = [30\ 90]^\top$, but is otherwise identical to the previously used parameters in Table 6.4. This dataset was collected to analyze the performance of the GP-MPC in a region of the state space around the training trajectory, and not just in the training trajectory itself, i.e., to judge how well the GP generalizes. The end effector trajectory from this test is shown in Figure 6.8, and the corresponding joint space tracking error is shown in Figure 6.9.

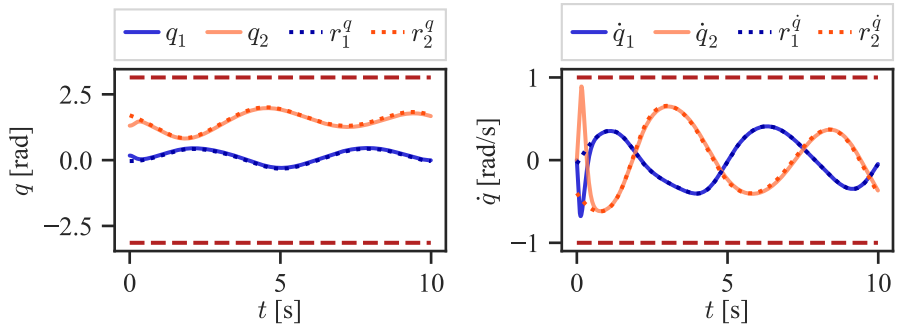


Figure 6.4: Joint angles and joint velocities for 2 DOF planar manipulator following joint space trajectory with linear MPC.

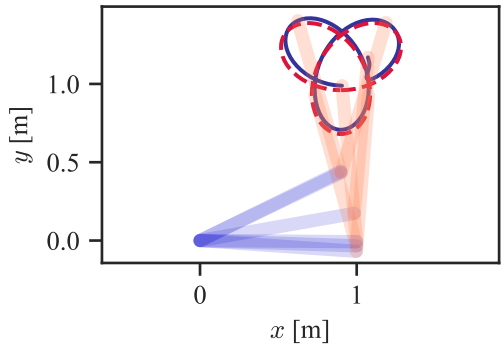


Figure 6.5: Trajectory of 2 DOF planar manipulator with linear MPC in blue, and desired trajectory in red.

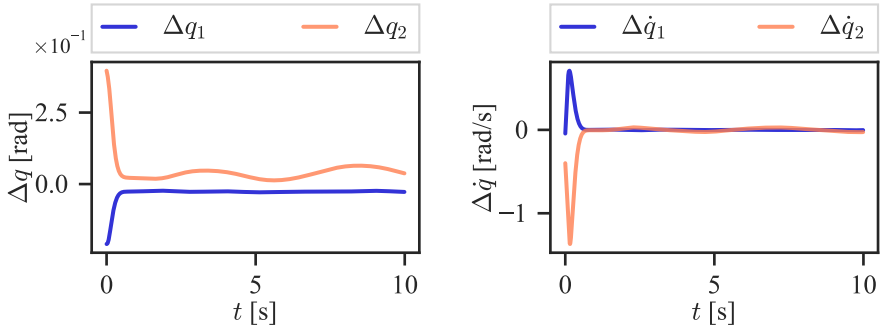
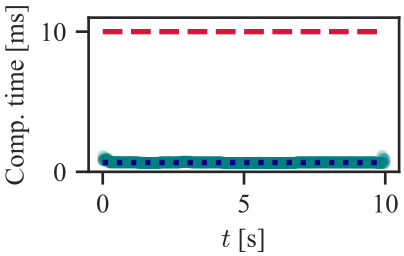
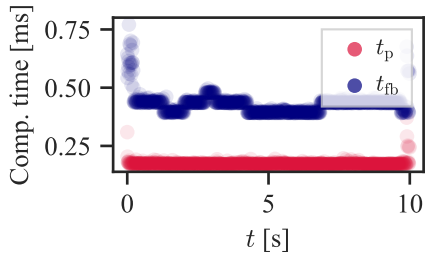


Figure 6.6: Joint trajectory tracking error for 2 DOF planar manipulator with linear MPC.



(a) Computation time of linear MPC for 2 DOF joint space trajectory tracking example. The average is indicated by the dotted line.



(b) Computation time for preparation stage t_p and feedback stage t_fb of RTI with linear MPC in 2 DOF joint space trajectory tracking example.

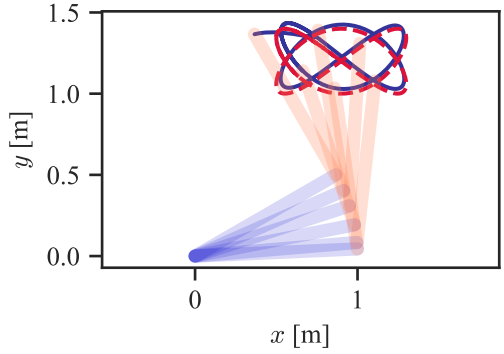


Figure 6.8: Trajectory of 2 DOF planar manipulator with linear MPC in blue, and desired trajectory in red.

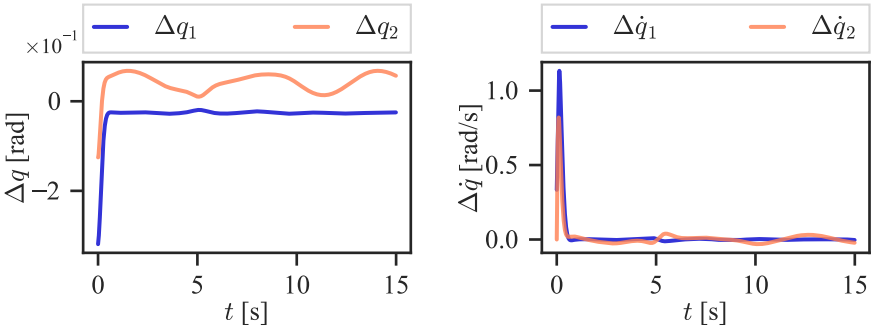


Figure 6.9: Joint trajectory tracking error for 2 DOF planar manipulator with linear MPC.

6.1.2 GP training

A GP was then trained on the training set collected using linear MPC in the previous section. The dataset is visualized in Figure 6.10. The GP hyperparameters were trained for 50 000 iterations using the SVGP method with the Adam optimizer for $\widetilde{M} = 20$ inducing variables. This and all subsequent GPs in this chapter were trained on a AMD Ryzen 9 3900X CPU with 16 GB RAM using the Adam optimizer.

Furthermore, for the SVGP method, for which SGD is possible, a minibatch size of 128 was used in this work. Finally, the training in GPflow was implemented such that a lower limit of $1 \cdot 10^{-4}$ was imposed on σ_n during training. This, in effect, makes the noise variance act as jitter on the diagonals on the covariance matrix in Eq. (4.19), such that the matrix remains positive definite and the inverse well-conditioned.

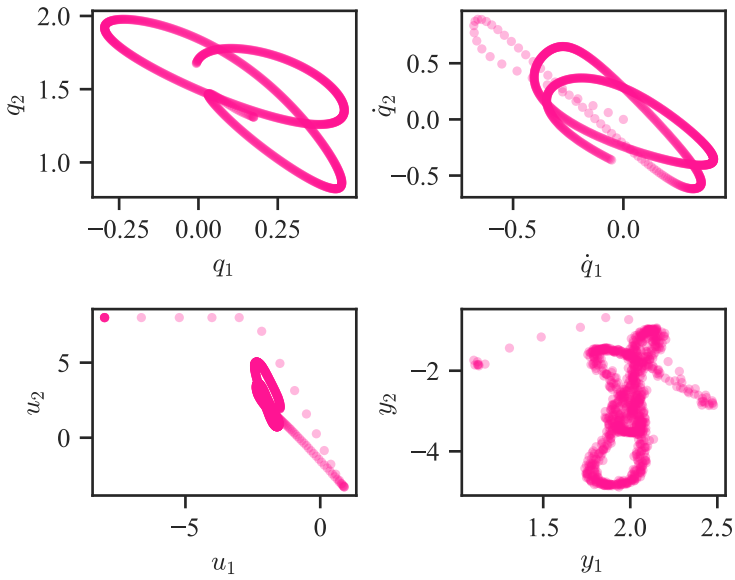


Figure 6.10: Training dataset $\mathcal{D}$ split into $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{u}$ and $\boldsymbol{y}$.

In Figure 6.11 the ELBO and root mean square error (RMSE) of the prediction on the training set are shown over the training duration. The evolution of the hyperparameters while training are shown in Figure 6.12.

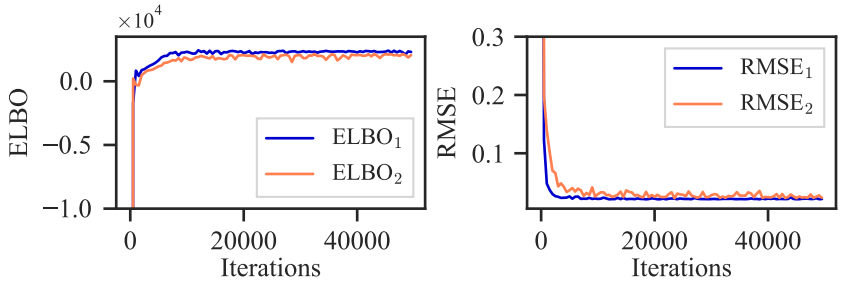


Figure 6.11: ELBO and RMSE on training set for every output dimension.

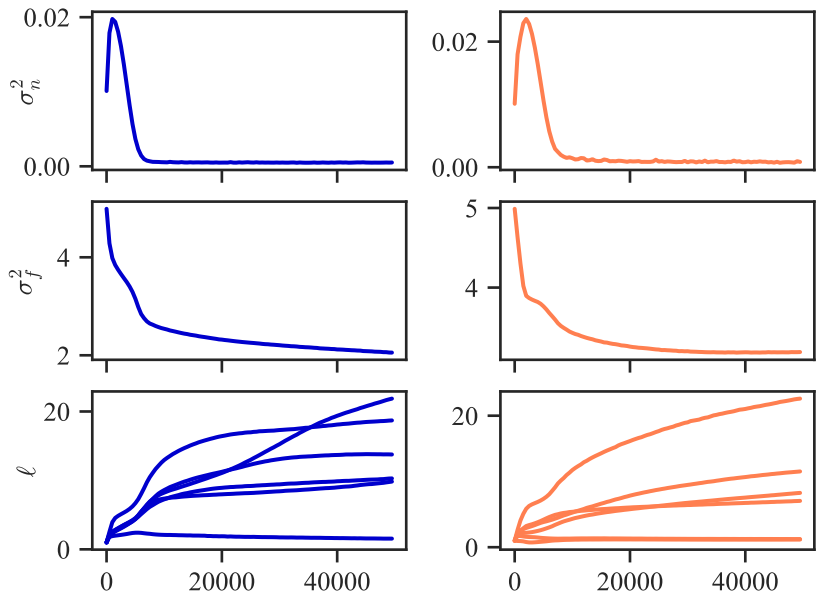


Figure 6.12: Evolution of hyperparameter values while training. Noise variance σ_n^2 is shown in the top row, signal variance σ_f^2 is shown in the middle row and length scales ℓ are shown in the bottom row, for dimension 1 to the left and dimension 2 to the right. For brevity the individual length scale indices are not indicated.

Using the GP to predict the output of the test set, an RMSE of $0.0221\,\mathrm{rad}$ for joint 1 and $0.0223\,\mathrm{rad}$ for joint 2 is obtained. The resulting predicted posterior is visualized in Figure 6.13, and compared to the true distribution of the output of the test set.

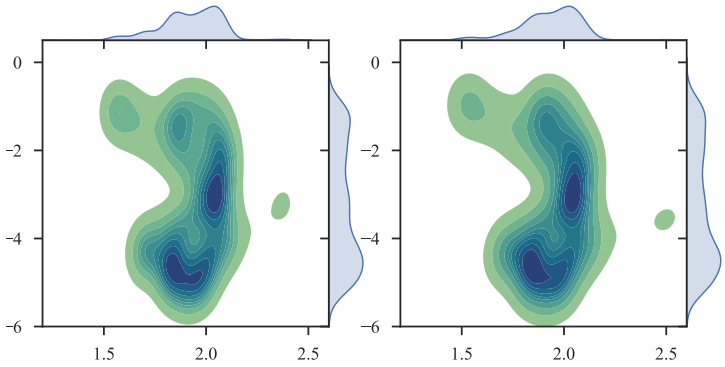


Figure 6.13: Predicted (left) and true posterior distribution (right) on test set.

To further evaluate the GP's ability to generalize to new datasets, cross-validation (CV) is used. Specifically, K-fold CV is considered, meaning the training set is split into K sets of equal size, and for every set, that set is used as the test set, and the union of the remaining $K-1$ is used as the training set. Finally, the average error metric over the K iterations, here the prediction RMSE, provides a better way of evaluating the performance of the GP compared to only using a single test set. For the 2 DOF example considered in this section, the training set was split into $K=5$ folds. The resulting average RMSE is $0.021\,59\,\mathrm{rad/s^2}$ for joint 1 and $0.027\,58\,\mathrm{rad/s^2}$ for joint 2.

In order to further investigate the effect of how the number of inducing variables is chosen, K-fold CV is again done, now repeated for different values of M. The resulting RMSEs are shown in Figure 6.14, and it can be seen that for this system and choice of trajectory, the RMSE seems to stop improving noticeably when M reaches about 25.

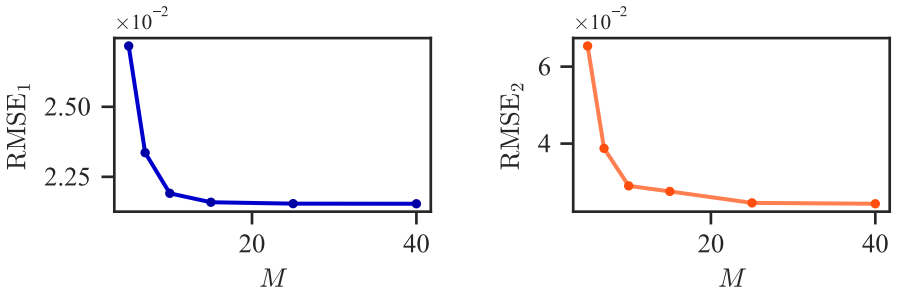


Figure 6.14: Average RMSE from K-fold cross-validation with $K = 5$ and $40\,000$ iterations as a function of number of inducing variables.

6.1.3 GP-based MPC

The GP-MPC controller Eq. (5.9) was tested with the trained GP and compared to linear MPC, for the same parameters in Table 6.4. The resulting trajectory of the manipulator in the xy-plane is shown in Figure 6.15, and the joint trajectory tracking error is presented in Figure 6.16. The predicted disturbance from the GP is shown in Figure 6.17, and the GP-MPC computation time is plotted in Figure 6.18a. These results show how the GP can infer the disturbance dynamics to a high accuracy, and thereby reduce the tracking error.

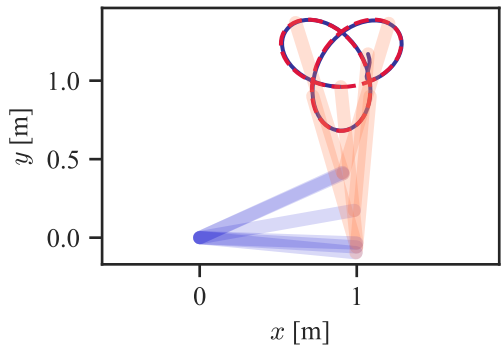


Figure 6.15: Trajectory of 2 DOF planar manipulator with GP-MPC in blue, and desired trajectory in red.

Finally, the respective computation times of the preparation stage and the feedback stage are shown in Figure 6.18b, which when compared to Figure 6.7b shows that the

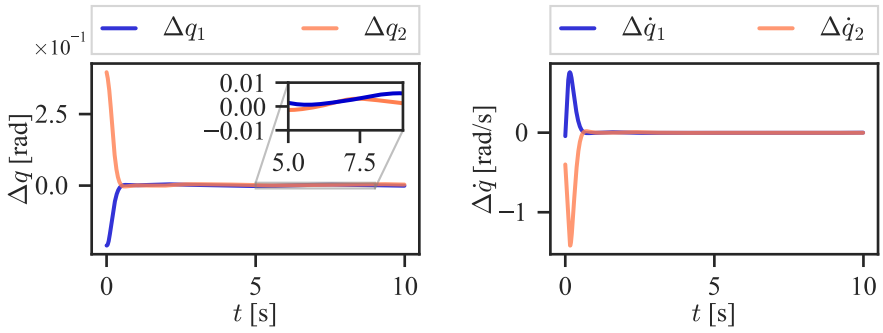


Figure 6.16: Joint trajectory tracking error on trefoil knot trajectory for 2 DOF planar manipulator with GP-MPC.

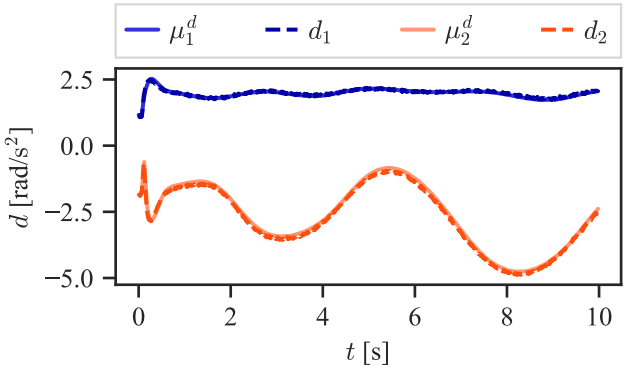
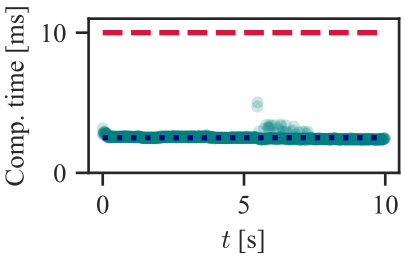


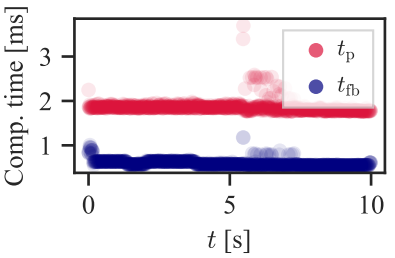
Figure 6.17: True and predicted disturbance for trefoil knot trajectory.

preparation stage now takes the majority of the computation time, while the timing of the feedback stage looks similar to how it was for linear MPC. This makes sense, as evaluating the values in the approximate QP in Eq. (3.13) may take more time depending on the complexity of the cost function, dynamics, and constraints, yet solving the resulting QP will not differ as much in time. This shows how the preparation stage will typically take the most time for OCPs with nonlinear dynamics.

When evaluating the GP-MPC controller on the test set with the Lissajous trajectory, a similar performance is observed, showing how the GP generalizes from the training set. The end effector trajectory is presented in Figure 6.19 and the state trajectory tracking error is shown in Figure 6.20.



(a) Computation time for GP-MPC for 2 DOF joint space trajectory tracking example. The average is indicated by the dotted line.



(b) Computation time for preparation stage t_p and feedback stage t_fb of RTI with GP-MPC in 2 DOF joint space trajectory tracking example.

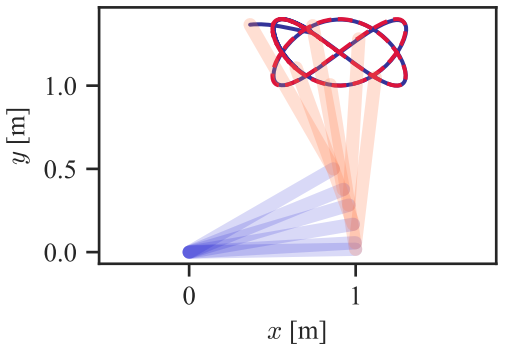


Figure 6.19: Trajectory of 2 DOF planar manipulator with GP-MPC in blue, and desired trajectory in red.

Finally, in Table 6.5 key values that summarize the performance of the controllers for the different cases are shown. Specifically, the trajectory tracking RMSE

$$\mathrm{RMSE}_q = \sqrt{\frac{1}{N_T}\sum_{k=0}^{N_T} \|\boldsymbol{q}_k - \boldsymbol{r}_k^q\|^2} \tag{6.3}$$

is considered, where N_T is the total number of steps. The RMSE of the model predictions, i.e., the predicted state at the next time step of the MPC prediction horizon $i = 1$, and the computation time of the RTI solver are also given.

The results show how adding the GP disturbance model increases the prediction

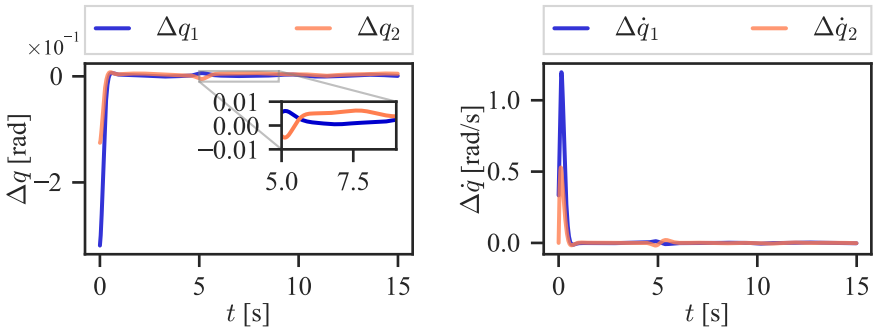


Figure 6.20: Joint trajectory tracking error on Lissajous trajectory for 2 DOF planar manipulator with GP-MPC.

Table 6.5: Comparison of trajectory tracking RMSE, prediction RMSE and computation time for linear MPC and GP-MPC, on training set and test set.

Simulation case	RMSE_q [rad]	$\text{RMSE}_{\text{pred}}$	t_{solver} [ms]
Linear MPC (train)	$7.272 \cdot 10^{-2}$	$3.596 \cdot 10^{-2}$	0.664
GP-MPC (train)	$5.488 \cdot 10^{-2}$	$5.056 \cdot 10^{-4}$	2.515
Linear MPC (test)	$6.222 \cdot 10^{-2}$	$4.104 \cdot 10^{-2}$	0.667
GP-MPC (test)	$3.255 \cdot 10^{-2}$	$1.140 \cdot 10^{-3}$	2.555

accuracy significantly, especially when considering that this error will be propagated N times over the prediction horizon. This results in a somewhat lower tracking error. A larger decrease in tracking error is likely not observed since the initial joint configuration of the robot is far away from the target trajectory. In later sections, trajectory blending will be used to reduce this transient error spike.

This increase in prediction accuracy does, however, come at the cost of significantly higher computation time, as well as the added complexity of having to train the GP hyperparameters. It was also observed that the GP could generalize to other trajectories and provide significantly better predictions also in this case, yet not as well as for the training trajectory itself. Finally, it must be emphasized that these are only single test cases and, as such, do not represent how well the performance improvement will be in general. A Monte Carlo simulation approach, where the trajectory choice and initial conditions are randomized, would give more robust numerical results. The limitation of long training

durations made this time-consuming and was therefore not explored further in this work.

6.2 Trajectory tracking for 6 DOF robot manipulator in simulation

For a more realistic example, trajectory tracking for a 6 DOF robot manipulator is considered. Specifically a UR10e robot is used, with DH parameters given in Table 6.6, and joint angle, velocity, and torque limits given in Table 6.7. Furthermore, all the limits are symmetric, i.e., $q_{\text{min}} = -q_{\text{max}}$, $\dot{q}_{\text{min}} = -\dot{q}_{\text{max}}$, $\tau_{\text{min}} = -\tau_{\text{max}}$. A visualization of the UR10e in PyBullet is shown in Figure 6.21.

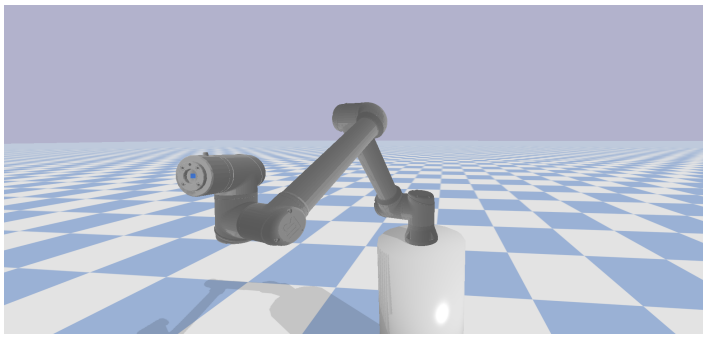


Figure 6.21: UR10e robot in PyBullet simulation environment.

Table 6.6: DH parameters for the UR10e robot, given in *Parameters for calculations of kinematics and dynamics* 2020.

Joint	θ [rad]	a [m]	d [m]	α [rad]
1	0	0	0.180 70	$\pi/2$
2	0	−0.612 70	0.0	0
3	0	−0.571 55	0.0	0
4	0	0	0.174 15	$\pi/2$
5	0	0	0.119 85	$-\pi/2$
6	0	0	0.116 55	0

Table 6.7: Joint limits for the UR10e robot. The maximum joint angle $q_{\max}$, joint angular velocity $\dot{q}_{\max}$ and joint torque $\tau_{\max}$ are given for every joint on the 6 DOF robot, provided in *E-Series From Universal Robots* 2020 and *Max. Joint Torques* 2015.

Joint	$q_{\max}$ [deg]	$\dot{q}_{\max}$ [deg /s]	$\tau_{\max}$ [Nm]
1	360	120	330
2	360	120	330
3	360	180	150
4	360	180	56
5	360	180	56
6	360	180	56

A significant model mismatch is introduced by doubling the mass and inertia of the last link of the manipulator arm, the values of which are detailed in Table 6.8. Furthermore, viscous damping terms are introduced in most of the joints, which are ignored in the prior model. The viscous damping coefficients used in simulation are $\boldsymbol{F}_v = [\,8.0 \;\; 6.0 \;\; 0.5 \;\; 0.005 \;\; 0.01 \;\; 0.0\,]^\top \mathrm{Nms/rad}$. Note that no friction was added to the last joint, as the NMPC controller struggled with having both friction and unknown mass on the last link. The changed mass and inertia of the last linkage is relevant, for instance, when attaching something to the robot end effector with unknown dynamical parameters, which is the case for the lab experiments in Section 6.3. Viscous friction is in many model-based control approaches neglected and therefore a frequent source of model mismatch. Using a GP to learn the disturbance introduced by these factors is therefore of practical interest.

Table 6.8: True and prior parameter values for final link of UR10e model used in simulation and controllers respectively.

Parameter	True	Prior
m_6 [kg]	$2.02 \cdot 10^{-1}$	$4.0 \cdot 10^{-1}$
$I_{6,xx}$ [kgm^2]	$1.444 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
$I_{6,yy}$ [kgm^2]	$1.444 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
$I_{6,zz}$ [kgm^2]	$2.045 \cdot 10^{-4}$	$4.0 \cdot 10^{-4}$

6.2.1 Joint space trajectory tracking

Joint space trajectory tracking is first considered, where a trajectory is generated using the Fourier series

$$\boldsymbol{q}(t) = \boldsymbol{q}_0 + \sum_{l=1}^{L} \boldsymbol{a}_l \sin(l\omega_f t) + \boldsymbol{b}_l \cos(l\omega_f t) - \boldsymbol{b}_l, \tag{6.4}$$

where $L = 5$ and $\omega_f = 0.05\pi$ were used for the tests. The $\boldsymbol{q}_0$ and $-\sum_{l=1}^{L} \boldsymbol{b}_l$ terms were added such that the trajectory is centered around the initial condition $\boldsymbol{q}_0$, i.e., $\boldsymbol{q}(0) = \boldsymbol{q}_0$. The joint velocity reference trajectory is found by the derivative

$$\dot{\boldsymbol{q}}(t) = \sum_{l=1}^{L} \boldsymbol{a}_l l\omega_f \cos(l\omega_f t) - \boldsymbol{b}_l l\omega_f \sin(l\omega_f t). \tag{6.5}$$

The Fourier coefficients $\boldsymbol{a}_l$ and $\boldsymbol{b}_l$ were chosen to generate an appropriate trajectory that spans a significant part of the state space while avoiding joint configurations close to singularities and keeping reasonable required maximum joint accelerations. The Fourier coefficients are given in Table 6.9. The trajectory, mapped to Cartesian space, is shown in Figure 6.22. While the reference trajectory starts in the initial joint configuration of the robot, trajectory blending was applied to avoid an initial discontinuity in the desired velocity by blending the joint angles with Eq. (5.18) and the joint velocities with Eq. (5.23), over a duration of $T_b = 5\,\mathrm{s}$.

Table 6.9: Fourier coefficients for the joint space trajectory of the UR10e robot manipulator arm.

Joint	a_1	b_1	a_2	b_2	a_3	b_3	a_4	b_4	a_5	b_5
1	0.8	−0.6	−0.25	0.3	0.2	0.1	−0.06	0.02	0.1	0.04
2	−0.3	−0.4	0.24	−0.1	−0.02	0.04	0.08	0.12	−0.05	0.0
3	−0.6	0.2	−0.45	0.0	0.0	0.04	−0.1	−0.05	0.03	0.09
4	0.4	0.3	−0.09	−0.03	−0.07	0.014	0.0	0.2	−0.05	0.03
5	0.31	0.36	−0.03	−0.1	0.12	−0.06	−0.01	−0.14	−0.07	0.0
6	0.3	0.2	−0.2	−0.2	0.023	0.0	−0.02	0.16	0.08	0.0

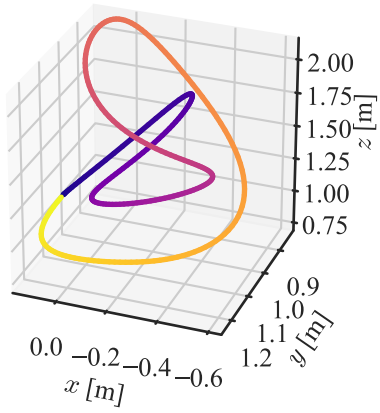


Figure 6.22: End effector trajectory shown in Cartesian space. Time is indicated by the color map from blue to yellow.

6.2.1.1 Joint space linear MPC test

The UR10e robot manipulator arm was then simulated in the PyBullet simulation environment and controlled using the linear MPC for joint space trajectory tracking, with direct torque control of the joint motors with viscous damping. The system was simulated for $T = 40\,\mathrm{s}$ with a simulation time step of $\Delta t = 5\,\mathrm{ms}$. The configuration of the MPC parameters are stated in Table 6.10.

In Figure 6.23 the calculated joint acceleration input and torque input are shown. The resulting state trajectory is shown over time in Figure 6.24 and shown in the simulation environment for specific time steps in Figure 6.25. The tracking error is given in Figure 6.26, and a significant error is seen, especially for joint 4 and joint 5. Finally the computation time of the SQP RTI solver is shown in Figure 6.27.

Table 6.10: Parameter values for joint space trajectory tracking test with 6 DOF manipulator in PyBullet simulation environment.

Parameter	Value
N	20
t_s [ms]	10.0
$\boldsymbol{q}_0$ [deg]	$[100, -150, -50, -70, -70, 90]$
$q_{\max}$ [rad]	2π
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	10.0
ϵ	0.0456
$\boldsymbol{Q}$	$10^3 \cdot \boldsymbol{I}_{12}$
$\boldsymbol{R}$	$10^{-1} \cdot \boldsymbol{I}_6$
$\boldsymbol{R}_{\text{NMPC}}$	$10^{-2} \cdot \boldsymbol{I}_{12}$
$\boldsymbol{S}$	$5 \cdot 10^{-4} \cdot \boldsymbol{I}_6$

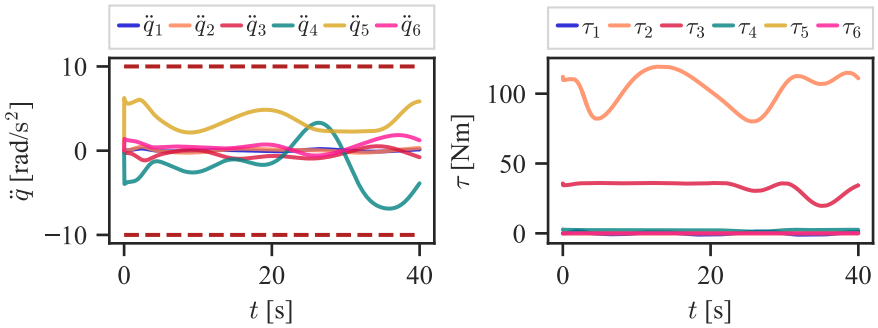


Figure 6.23: Joint acceleration input and computed joint torque for 6 DOF manipulator following joint space trajectory with linear MPC.

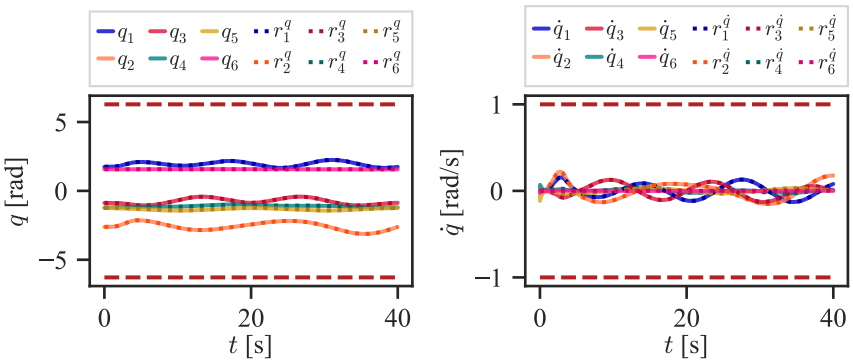


Figure 6.24: Joint angles and joint velocities for 6 DOF manipulator following joint space trajectory with linear MPC.

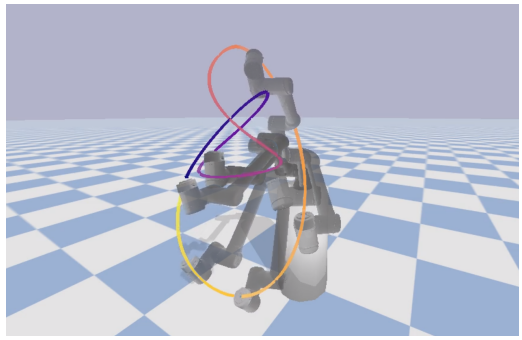


Figure 6.25: End-effector trajectory and configuration of robot at certain time steps.

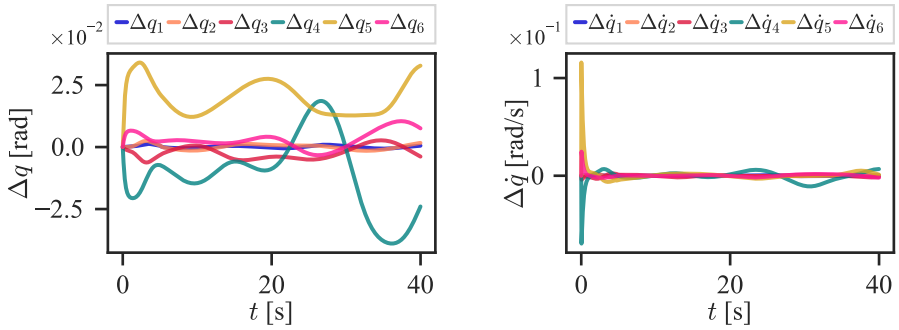


Figure 6.26: Joint trajectory tracking error for 6 DOF manipulator with linear MPC.

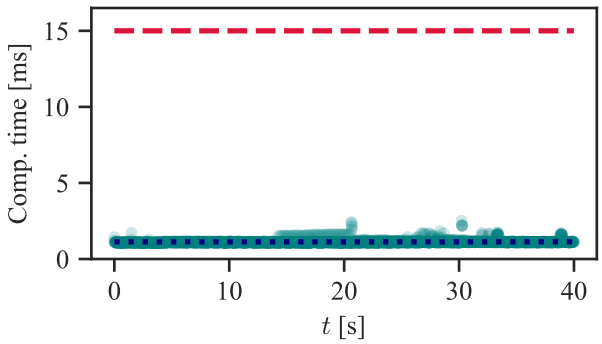


Figure 6.27: Computation time with linear MPC for 6 DOF manipulator. The average is indicated by the dotted line.

6.2.1.2 Joint space GP-based MPC test

A GP was trained used the dataset collected in the previous section, resulting in a total dataset size of 2667 samples. The GP was trained for 50 000 iterations, with $\widetilde{M} = 40$ inducing variables. Limited success was experienced with the SVGP method, as oscillatory behavior in the system was observed when it was controlled with the GP dynamics away from the vicinity of the training trajectory. The VFE method was tested instead, for which no oscillations were observed. These oscillations were likely a result of the SVGP approximation being less accurate and thereby giving a wrong prediction in uncertain regions of the state space, pushing the system further away from the accurate region and thus inducing oscillations. The accuracy of the different sparse GP methods discussed in Section 4.3 is further analyzed later in this section.

Furthermore, these oscillations, induced by the accuracy of the GP prediction being local, were a general challenge for more complex dynamical systems, such as the present fixed-base 6 DOF manipulator, as well as for the free-floating manipulator covered later in Section 6.4. The implementation was tried to be made more robust by adding a cost term on the input rate of the controller, i.e., the joint jerk, as previously discussed in Section 5.1.2. This was formulated as $\|\ddot{\boldsymbol{q}}\|^2_{\boldsymbol{S}}$, with $\boldsymbol{S} \in \mathbb{R}^{n_q \times n_q}$ being the diagonal jerk weight matrix. The numerical value of $\boldsymbol{S}$ for the following simulation is given in Table 6.10. This issue of chattery behavior with GP dynamics and possibly remedies will be discussed further in Section 7.1.

GP-MPC was tested, and compared to the linear MPC. In Figure 6.28 the GP prediction in closed-loop is shown, and it is seen that also for the considerably more complex case of a 6 DOF manipulator arm with viscous motor friction, the prediction is close to the true disturbance. The tracking errors are given in Figure 6.29, showing a substantial reduction compared to the linear MPC without the added GP dynamics. However, in the computation time plot in Figure 6.30 a substantial increase is observed, compared to Figure 6.27.

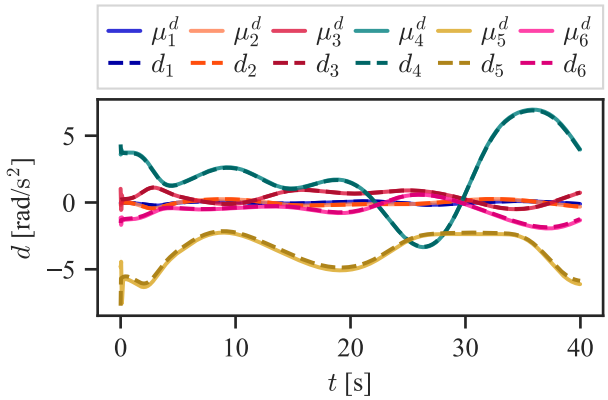


Figure 6.28: GP disturbance prediction μ^d and true disturbance d.

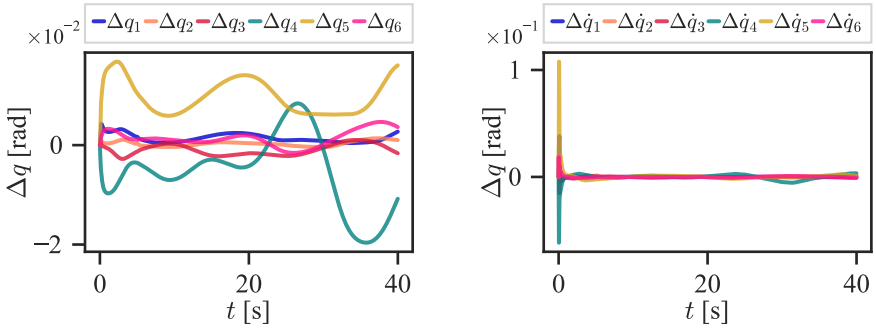


Figure 6.29: Joint trajectory tracking error for 6 DOF manipulator with GP-MPC.

Finally, in order to compare the different sparse GP approaches discussed in Section 4.3, K-fold CV was performed on the 6 DOF joint space training dataset collected in

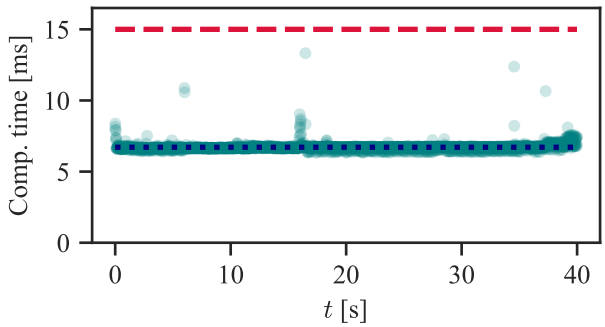


Figure 6.30: Computation time with GP-MPC for 6 DOF manipulator. The average is indicated by the dotted line.

Section 6.2.1.1 using the discussed sparse GP methods, namely FITC, VFE and SVGP. $K = 5$ folds were used, each trained for $40\,000$ iterations with $\widetilde{M} = 20$ inducing points. The resulting average RMSEs for every output dimension are presented in Figure 6.31, with the corresponding standard deviation over the folds indicated. It was observed that while the FITC and VFE errors are comparable in size, the errors from SVGP were significantly higher. This is likely caused by SGD taking longer to converge. However, from the respective average training times per dimension of $T_{\text{FITC}} = 125.66\,\text{s}$, $T_{\text{VFE}} = 115.85\,\text{s}$, $T_{\text{SVGP}} = 49.47\,\text{s}$, it is evident that a significant training speed advantage is gained by using SGD.

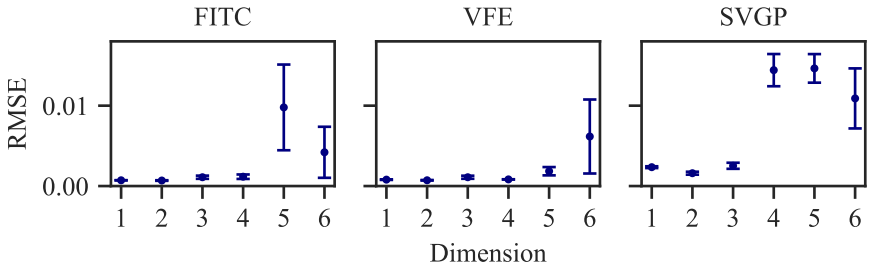


Figure 6.31: Comparison of average RMSE for K-fold CV for FITC, VFE and SVGP methods, on 6 DOF joint space training dataset. Average RMSE is given for every output dimension, with standard deviation indicated as error bars.

6.2.1.3 Joint space NMPC test

Joint space trajectory tracking for the 6 DOF manipulator was also tested with the NMPC controller Eq. (5.6). The nonlinear dynamics were discretized using ERK4. The parameters in Table 6.10 were used, but now using the $\boldsymbol{R}_{\text{NMPC}}$ weight matrix, penalizing $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$. Furthermore, a terminal cost of $\boldsymbol{P} = 20\boldsymbol{Q}$ was used. Torque constraints were added with the bounds given in Table 6.7. The joint tracking error is plotted in Figure 6.32, which is of about the same size as the linear MPC. The computation time is shown in Figure 6.33.

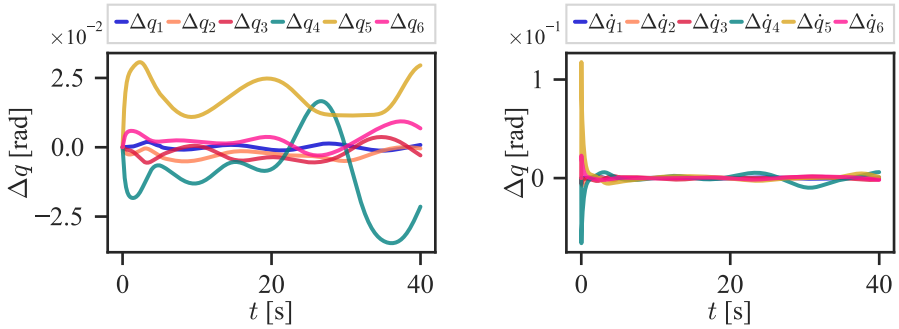


Figure 6.32: Joint trajectory tracking error for 6 DOF manipulator with NMPC.

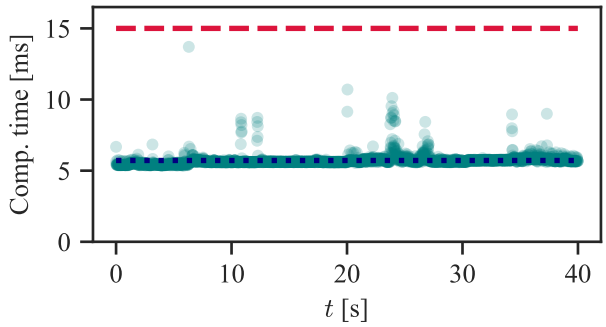


Figure 6.33: Computation time with NMPC for 6 DOF manipulator. The average is indicated by the dotted line.

The main results are summarized in Table 6.11, with the trajectory tracking RMSE, prediction RMSE and solver computation time shown. It is observed that the tracking error and prediction error are similar for linear MPC and NMPC, yet significantly smaller

for GP-MPC. Furthermore, for this problem, GP-MPC is the slowest but comparable to NMPC. Linear MPC is much faster than the other nonlinear methods, needing little time to generate the QPs in SQP, as the problem is already quadratic. The timing is still well within the sample time, and thus real-time feasible. Large spikes in computation time are, however, observed.

Table 6.11: Comparison of trajectory tracking RMSE, prediction RMSE and computation time for linear MPC, GP-MPC and forward dynamics-based NMPC.

Test	$\text{RMSE}_q[\text{rad}]$	$\text{RMSE}_{\text{pred}}$	$t_{\text{solver}}[\text{ms}]$
Linear MPC	$2.859 \cdot 10^{-2}$	$7.673 \cdot 10^{-2}$	1.113
GP-MPC	$1.412 \cdot 10^{-2}$	$2.184 \cdot 10^{-3}$	6.710
NMPC	$2.583 \cdot 10^{-2}$	$7.715 \cdot 10^{-2}$	5.715

6.2.2 Task space trajectory tracking

In this section, task space trajectory tracking is considered by including the forward kinematics directly in the OCP cost function, as discussed in Section 5.1.2. Again tracking of a trefoil knot curve is considered, now in three-dimensional space:

$$\boldsymbol{r}^p(t) = a_1 \begin{bmatrix} \sin(\omega t) + 2\sin(2\omega t) \\ \cos(\omega t) - 2\cos(2\omega t) \\ -3\sin(3\omega t) \end{bmatrix} + \bar{\boldsymbol{x}}, \tag{6.6}$$

with parameters $w = 0.25$, $a = 0.1$ and $\bar{\boldsymbol{x}} = [\,0.2 \;\; 0.7 \;\; 1.3\,]^\top$. Simultaneously a fixed desired end effector orientation is tracked, denoted by the Euler angle reference $\boldsymbol{r}^\Theta$. The desired pose trajectory is shown in Figure 6.34. The position and quaternion references are blended from the initial pose in order to get a smooth approach towards the desired trajectory with blending time T_b. As detailed in Section 5.2, the position is blended by linear interpolation, and the rotation quaternion is blended by SLERP.

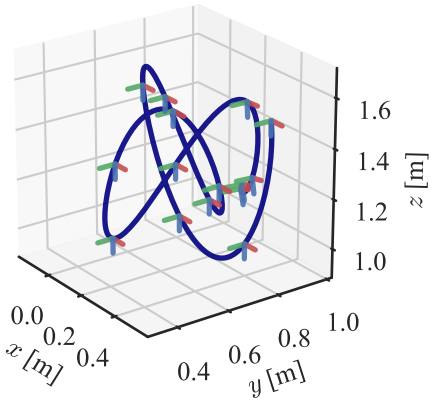


Figure 6.34: Desired end effector pose trajectory $\boldsymbol{r}^e(t)$, with trefoil knot position trajectory and fixed orientation reference.

6.2.2.1 Task space linear MPC

Feedback linearization-based MPC was again tested first, with parameters given in Table 6.12. Note how the terminal cost can no longer be approximated with the discrete Ricatti equation since the cost function is now highly nonlinear. For this experiment, the terminal cost was, therefore, tuned manually to $\boldsymbol{P} = 20\boldsymbol{Q}$. An alternative could be to try to linearize the forward kinematics around the point at the end of the horizon and solve a discrete Ricatti equation for this point. As these computations would have to be done at every time step, they are, however, not as practical to implement, and the simpler approach of manually tuning $\boldsymbol{P}$ was used instead.

The system was simulated for $T = 30\,\mathrm{s}$. The computed joint accelerations and control torques are shown in Figure 6.35. A series of snapshots that visualize the trajectory of the end effector and the robot configurations are shown in Figure 6.36. The joint angles and joint velocities during the test are presented in Figure 6.37. The end effector pose and pose error are shown in Figure 6.38 and Figure 6.39, respectively.

Table 6.12: Parameter values for task space trajectory tracking test with 6 DOF manipulator in simulation.

Parameter	Value
N	20
t_s [ms]	15.0
T_b [s]	10.0
$\boldsymbol{r}^{\Theta}$ [deg]	$[180, 0, 0]$
$\boldsymbol{q}_0$ [deg]	$[80, -120, -100, -50, 70, 0]$
$q_{\max}$ [rad]	2π
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	10.0
ϵ	0.0456
$\boldsymbol{Q}$	$10^4 \cdot \boldsymbol{I}_6$
$\boldsymbol{R}$	$\boldsymbol{I}_6$
$\boldsymbol{R}_{\text{NMPC}}$	blkdiag$(10^{-2} \cdot \boldsymbol{I}_6, 10^{-3} \cdot \boldsymbol{I}_6)$
$\boldsymbol{S}$	$10^{-2} \cdot \boldsymbol{I}_6$
$\boldsymbol{P}$	$20\boldsymbol{Q}$

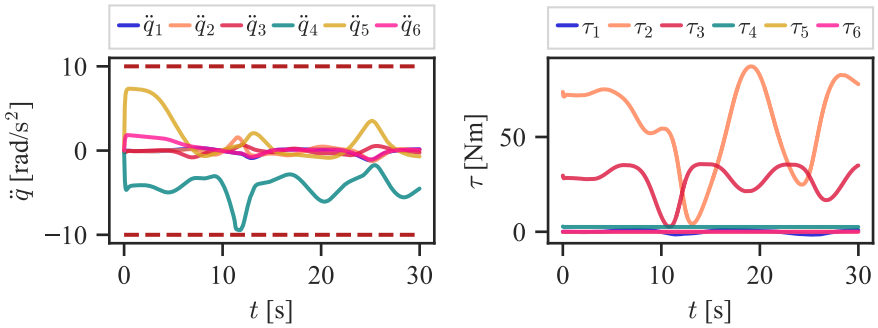


Figure 6.35: Joint acceleration input and computed joint torque for 6 DOF manipulator following joint space trajectory with linear MPC.

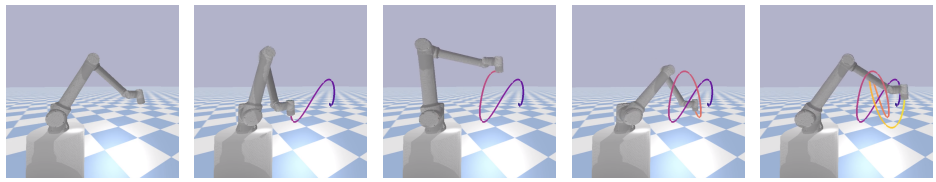


Figure 6.36: 6 DOF UR10e manipulator tracking end effector pose trajectory in simulation.

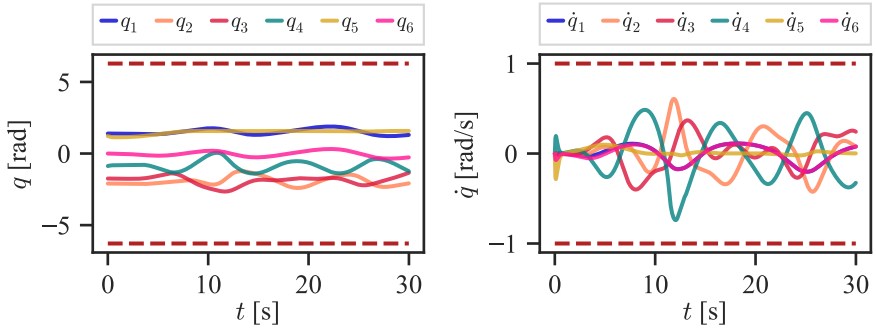


Figure 6.37: Joint angles and joint velocities for 6 DOF manipulator following task space trajectory with linear MPC.

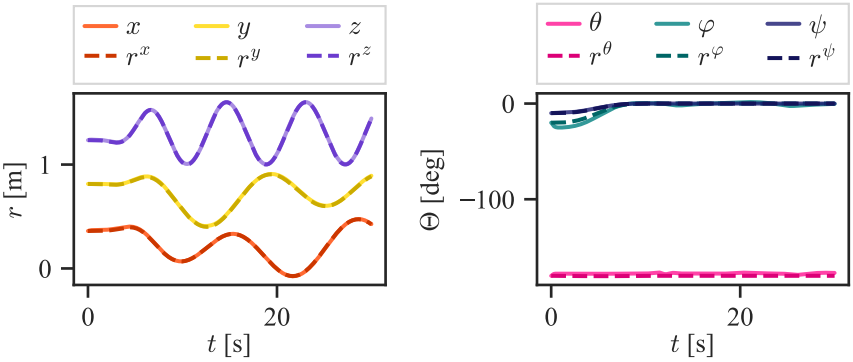


Figure 6.38: Desired and actual end effector pose for 6 DOF manipulator following task space trajectory with linear MPC.

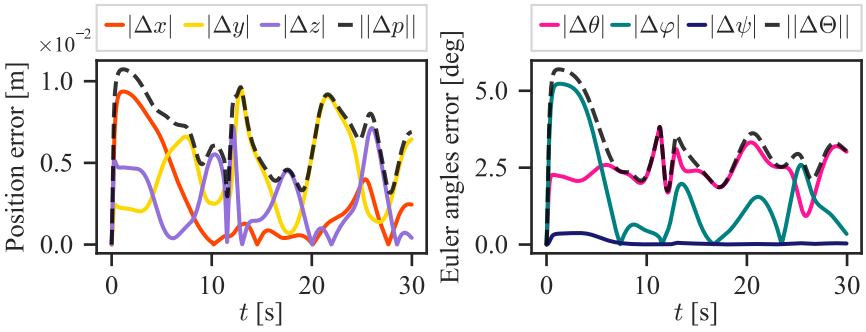


Figure 6.39: End effector pose trajectory tracking error for 6 DOF manipulator following task space trajectory with linear MPC.

6.2.2.2 Task space GP-based MPC test

A GP was trained, this time with the SVGP method for 100 000 iterations, with $\widetilde{M} = 40$ inducing variables and a dataset of 6000 samples. The closed loop GP prediction of the disturbance is plotted in Figure 6.40. The resulting GP-MPC pose tracking error from simulation is shown in Figure 6.41.

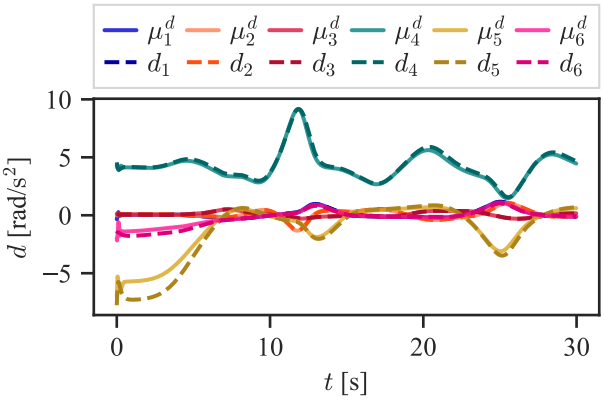


Figure 6.40: GP disturbance prediction μ^d and true disturbance d.

It is seen how the prediction is noticeably worse for the first few seconds, in which the blending occurs. Since there are fewer data points in the dataset representing this,

the prediction accuracy is lower. This highlights the issue of the GP prediction being local, discussed previously in Section 6.2.1.2. Trying different trajectories than the one used to train the GP also leads to chattering and, therefore, an unusable controller, further showing this limitation.

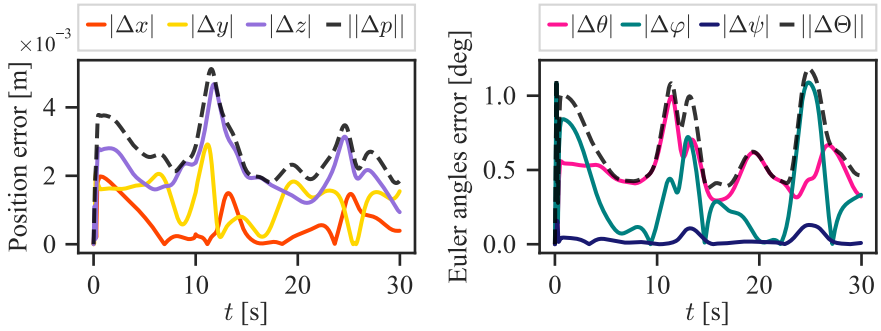


Figure 6.41: End effector pose trajectory tracking error for 6 DOF manipulator following task space trajectory with GP-MPC.

6.2.2.3 Task space NMPC test

Like for the joint space trajectory tracking problem, the NMPC controller in Eq. (5.6) is tested and compared to feedback linearized MPC and GP-MPC. The end effector pose trajectory tracking error is shown in Figure 6.42.

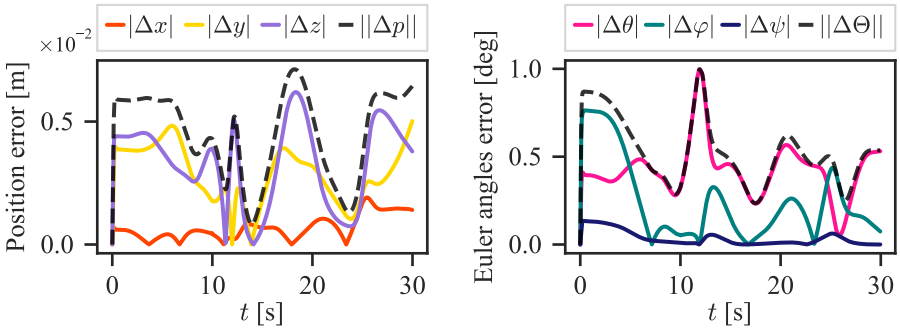


Figure 6.42: End effector pose trajectory tracking error for 6 DOF manipulator following task space trajectory with NMPC.

The position trajectory tracking RMSE

$$\text{RMSE}_p = \sqrt{\frac{1}{N_T}\sum_{k=0}^{N_T} \|\boldsymbol{h}_k^p - \boldsymbol{r}_k^p\|^2}, \tag{6.7}$$

orientation trajectory tracking RMSE

$$\text{RMSE}_o = \sqrt{\frac{1}{N_T}\sum_{k=0}^{N_T} \|\boldsymbol{\Theta}_k - \boldsymbol{r}_k^\Theta\|^2}, \tag{6.8}$$

prediction RMSE, and RTI computation times are given for the preceding tests in Table 6.13. It is seen that GP-MPC and NMPC have similar performance for orientation, but GP-MPC is noticeably better for the position. Again, the prediction error is much lower by including the GP disturbance term. For the feedback linearized MPC a significant increase in computation time is seen, which is to be expected for the added nonlinear cost function. The computation time for GP-MPC and NMPC are quite similar for this problem. Again, it must be emphasized that these results are only for a single test case and therefore do not speak about the general performance of these trajectory tracking approaches.

Table 6.13: Comparison of task space trajectory tracking RMSE, prediction RMSE and computation times for linear MPC, GP-MPC and forward dynamics-based NMPC.

Test	$\text{RMSE}_p[\text{m}]$	$\text{RMSE}_o[\text{deg}]$	$\text{RMSE}_{\text{pred}}$	$t_{\text{solver}}[\text{ms}]$
Linear MPC	$7.077 \cdot 10^{-3}$	3.3013	$5.437 \cdot 10^{-2}$	1.252
GP-MPC	$2.807 \cdot 10^{-3}$	0.6902	$1.598 \cdot 10^{-3}$	6.650
NMPC	$4.865 \cdot 10^{-3}$	0.5615	$5.049 \cdot 10^{-2}$	6.267

6.3 Lab tests of trajectory tracking for UR10e robot

Similar tests were done on a UR10e robot in a lab environment. The controllers were configured to communicate with the UR10e robot using the Real-time Data Exchange

(RTDE) interface, which provides a $500\,\mathrm{Hz}$ interface to the UR controller over an IP connection, and is, therefore, suitable for real-time control of the robot. The ur_rtde Python package by Lindvig 2021 was used to communicate with the robot over the RTDE interface. Since the RTDE interface only provides rotation vectors when sampling the end effector pose of the robot, the conversion formula between unit quaternions and rotation vectors given in Eq. (A.3) was 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.

The lab setup is shown in Figure 6.43, with a camera and gripper attached to the tool flange of the UR10e, which were taken off for the subsequent tests. However, an adapter was still attached, with unknown dynamical properties. The UR software provides an end effector calibration procedure, which was used to give a rough estimate of the mass and center of mass. This end effector attachment still introduced considerable uncertainty in the model, in addition to the uncertainty in the other parameters in the URDF.

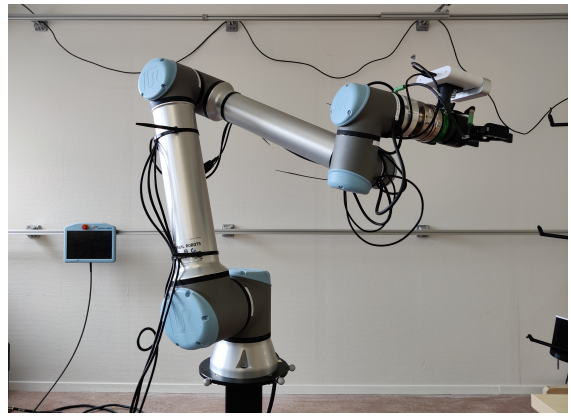


Figure 6.43: Lab setup with UR10e robot.

Several other differences between the simulation tests and the lab tests on the real robot are worth mentioning. One is that so far, direct motor torque control has been considered, yet UR robots do not provide a torque control interface, only joint angle and joint velocity control. In order to overcome this limitation, the velocity control interface was used instead. By applying the predicted velocity solution of the OCP at the next time step, i.e., $\dot{\boldsymbol{q}}_1^\star$, a similar motion can be achieved. This violates the assumption that the torque input in

the discussed MPC controllers are constant during the time step and should as such be treated as a discrepancy. This discrepancy will be discussed further in Chapter 7.

Furthermore, since the UR10e robot has active gravity compensation, the prior dynamics were generated using a zero gravity vector, i.e., $\boldsymbol{g} = \boldsymbol{0}$. Therefore, the computed torques are much smaller in this section, as the large offsets in torque due to gravity are removed and handled by the internal robot controller instead.

Both joint space and task space trajectory tracking was tested on the UR10e robot. The joint space results are given in Appendix D, of which Table D.2 provide the main comparison between linear MPC, GP-MPC and NMPC. In the following, the task space results are presented.

6.3.1 Task space linear MPC test

The task space trajectory detailed in Section 6.3.1 was also tested on the real robot. The controllers were tuned on the robot, with parameters summarized in Table 6.14. The test duration used here was $60\,\mathrm{s}$, resulting in a dataset of 4000 samples.

Table 6.14: Parameter values for task space trajectory tracking test on UR10e robot in lab environment.

Parameter	Value
N	20
t_s [ms]	15.0
$\boldsymbol{q}_0$ [deg]	$[135, -120, -100, -50, 70, 0]$
$q_{\max}$ [rad]	2π
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	1.5
ϵ	0.0456
$\boldsymbol{Q}$	$10^3 \cdot \boldsymbol{I}_6$
$\boldsymbol{R}$	$\boldsymbol{I}_6$
$\boldsymbol{R}_{\text{NMPC}}$	$\text{blkdiag}(2 \cdot 10^{-6} \cdot \boldsymbol{I}_6, 10^{-3} \cdot \boldsymbol{I}_6)$
$\boldsymbol{S}$	$10^{-2} \cdot \boldsymbol{I}_6$

The computed torque is shown in Figure 6.44 and the joint angles and velocities are

shown in Figure 6.45 for the feedback linearized MPC. The end effector pose is given in Figure 6.46 and the pose error is shown in Figure 6.47. Significant error spikes are seen for both position and orientation.

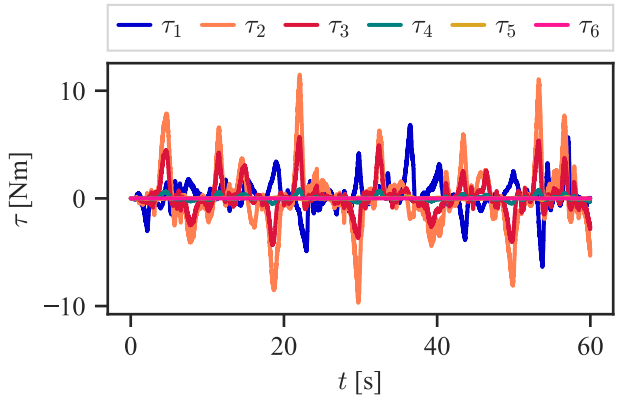


Figure 6.44: Computed torque input for UR10e task space trajectory tracking with linear MPC.

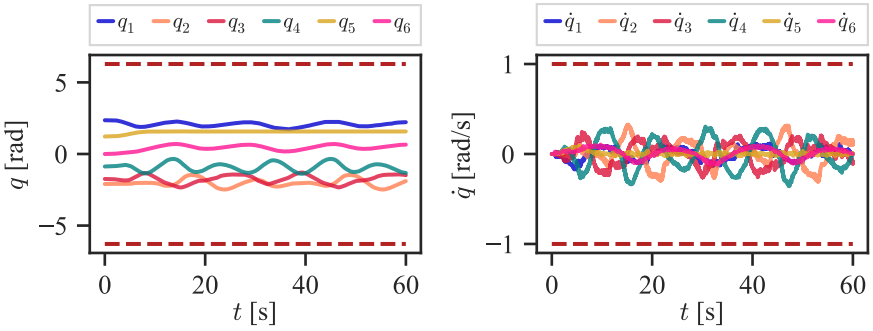


Figure 6.45: Joint angles and joint velocities for UR10e task space trajectory tracking with linear MPC.

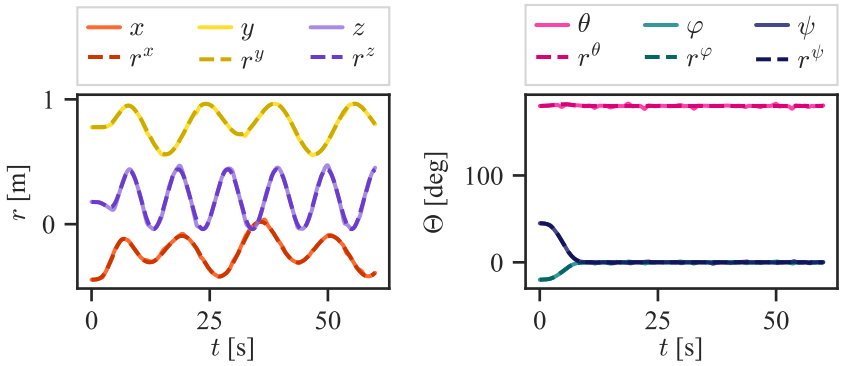


Figure 6.46: Desired and actual end effector pose with linear MPC.

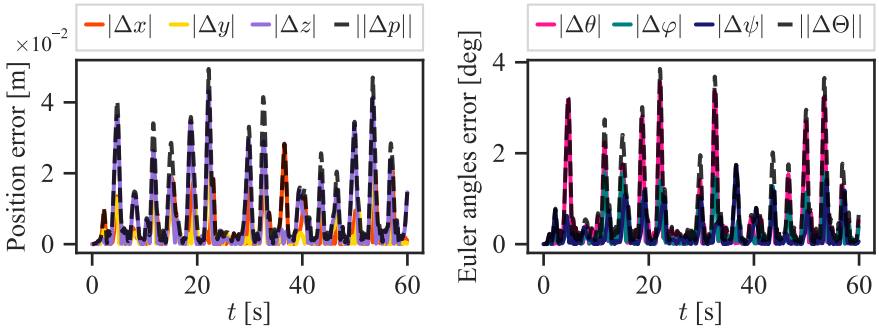


Figure 6.47: End effector pose tracking error for UR10e using linear MPC.

6.3.2 Task space GP-based MPC test

The hyperparameters of a sparse GP were trained using the SVGP method, for $180\,000$ iterations and with $\widetilde{M} = 25$ inducing points. The resulting closed loop GP prediction and end effector pose tracking error are shown in Figure 6.48 and Figure 6.49, respectively.

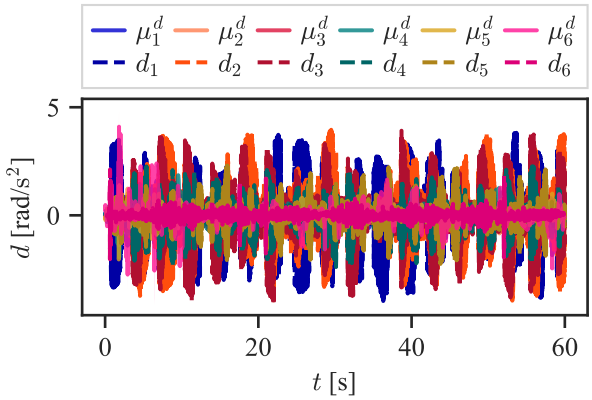


Figure 6.48: GP disturbance prediction μ^d and true disturbance d.

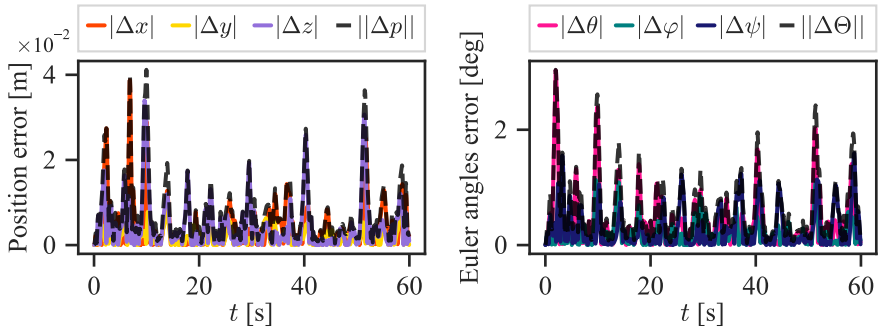


Figure 6.49: End effector pose tracking error for UR10e using GP-MPC.

6.3.3 Task space NMPC test

Finally, NMPC was also tested, with trajectory tracking error given in Figure 6.50.

The task space trajectory tracking errors and computation times are summarized in Table 6.15. Significantly longer computation times are observed, compared to the simulation case in Table 6.13, likely due to measurement noise making the warm start less efficient. Furthermore, NMPC has significantly lower tracking error than the other feedback linearization-based controllers.

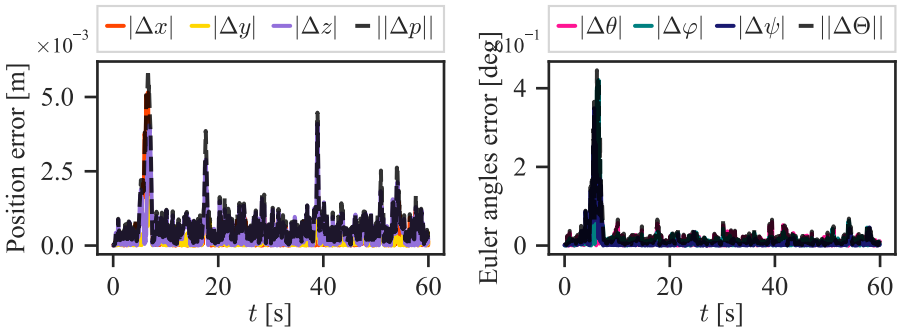


Figure 6.50: End effector pose tracking error for UR10e using NMPC.

Table 6.15: Comparison of joint space trajectory tracking RMSE, prediction RMSE and computation times for linear MPC, GP-MPC and forward dynamics-based NMPC.

Test	RMSE$_p$[m]	RMSE$_o$[deg]	t_{solver}[ms]
Linear MPC	$1.439 \cdot 10^{-2}$	1.060	2.400
GP-MPC	$1.044 \cdot 10^{-2}$	$8.140 \cdot 10^{-1}$	9.010
NMPC	$1.062 \cdot 10^{-3}$	$4.818 \cdot 10^{-2}$	9.414

6.4 Trajectory tracking for space manipulator in simulation

So far, model-predictive approaches to trajectory tracking have been tested under the assumption of a fixed base frame. However, MPC approaches to robot manipulator trajectory tracking also have considerable potential for floating-base systems, such as AUVs, drones, legged robots, and space manipulator systems. Simpler reactive approaches only based on kinematics, which are traditionally used for fixed-base systems, become problematic when the body frame moves in reaction to the applied control forces and torques. Furthermore, this coupling in the dynamics might mean model mismatch will have an even larger impact on performance. Control of a free-floating space manipulator system is considered in this section in order to test the feasibility of GP-MPC and NMPC for floating-base robotic systems.

For the following example a space manipulator with a 3 DOF anthropomorphic manipulator arm is used, as shown in Figure 5.5, and with DH parameters given in Table 6.16, from Siciliano et al. 2010. The length, mass, and inertia properties of the system are given in Table 6.17. Viscous friction is added to the motor joints, shown with the other joint parameters in Table 6.18. The motor friction is not a part of the prior model and thereby a source of model mismatch. The system is simulated in the PyBullet simulation environment, with $\Delta t = 5\,\mathrm{ms}$. The URDF model of the space manipulator used in simulation is shown in Figure 6.51.

Table 6.16: DH parameters for anthropomorphic arm.

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	q_1
2	ℓ_2	0	0	q_2
3	ℓ_3	0	0	q_3

Table 6.17: Link parameters for space manipulator system with anthropomorphic manipulator arm.

Link	ℓ [m]	m [kg]	I_{xx} [kgm^2]	I_{yy} [kgm^2]	I_{zz} [kgm^2]
Base	1.0	25.0	2.604	2.604	1.042
1	0.2	2.0	$3.413 \cdot 10^{-2}$	$3.413 \cdot 10^{-2}$	$160 \cdot 10^{-3}$
2	0.5	4.0	$8.493 \cdot 10^{-2}$	$8.493 \cdot 10^{-2}$	$320 \cdot 10^{-3}$
3	0.5	4.0	$8.493 \cdot 10^{-2}$	$8.493 \cdot 10^{-2}$	$320 \cdot 10^{-3}$

Table 6.18: Joint parameters for space manipulator system with anthropomorphic manipulator arm.

Joint	$q_{\max}$ [rad]	$\dot{q}_{\max}$ [rad/s]	$\ddot{q}_{\max}$ [rad/s^2]	$\tau_{\max}$ [Nm]	F_v [Nms/rad]
1	π	1.5	4.0	10.0	0.2
2	2π	1.5	4.0	10.0	0.1
3	2π	1.5	4.0	10.0	0.1

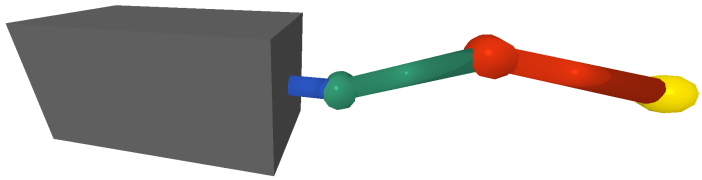


Figure 6.51: Space manipulator URDF model in PyBullet simulation.

In the following, joint space trajectory tracking MPC is applied for the space manipulator system, as discussed in Section 5.3.2. The task space formulation was also tested, the results of which are presented in Appendix D. The Fourier series trajectory Eq. (6.4) was followed, with $L = 4$ and coefficients shown in Table 6.19.

Table 6.19: Fourier coefficients for joint space trajectory for the free-floating manipulator arm.

Joint	a_1	b_1	a_2	b_2	a_3	b_3	a_4	b_4
1	1.2	0.2	0.2	−0.09	−0.05	0.03	0.0	0.0
2	0.6	0.25	−0.35	0.0	0.0	0.1	−0.1	0.0
3	0.0	−0.5	−0.2	0.1	0.15	0.0	0.0	0.1

6.4.1 Joint space linear MPC test

The system was simulated for $T = 50\,\mathrm{s}$ in PyBullet with feedback linearization-based MPC. The simulation parameters are shown in Table 6.20. The satellite base is initialized with position $\boldsymbol{p}_0$ and orientation represented by the Euler angles $\boldsymbol{\Theta}_0$. Furthermore, the system has zero initial momenta, i.e., $\boldsymbol{v}_s = \boldsymbol{\omega}_s = \dot{\boldsymbol{q}} = \boldsymbol{0}$. The joint angle, velocity and acceleration limits in Table 6.18 were used in the state and input constraints. Quadratic jerk costs weighted on $\boldsymbol{S}$ were used for both the linear MPC controller and the GP-MPC controller.

In Figure 6.52, the computed joint acceleration input and torques are shown. The joint states of the space manipulator are shown in Figure 6.53. The body velocities and pose are given in Figure 6.54 and Figure 6.55, respectively.

Table 6.20: Parameter values for joint space trajectory tracking test of space manipulator in simulation.

Parameter	Value
N	20
t_s [ms]	10.0
$\boldsymbol{p}_0$ [m]	$[0, 0, 1.0]$
$\boldsymbol{\Theta}_0$ [deg]	$[90, 0, 0]$
$\boldsymbol{q}_0$ [deg]	$[0.0, -20.0, 30.0]$
ϵ	0.0456
$\boldsymbol{Q}$	blkdiag$(10^3 \cdot \boldsymbol{I}_3, 10^2 \cdot \boldsymbol{I}_3)$
$\boldsymbol{R}$	$10^{-1} \cdot \boldsymbol{I}_3$
$\boldsymbol{R}_{\text{NMPC}}$	$10^{-2} \cdot \boldsymbol{I}_6$
$\boldsymbol{S}$	$10^{-3} \cdot \boldsymbol{I}_3$

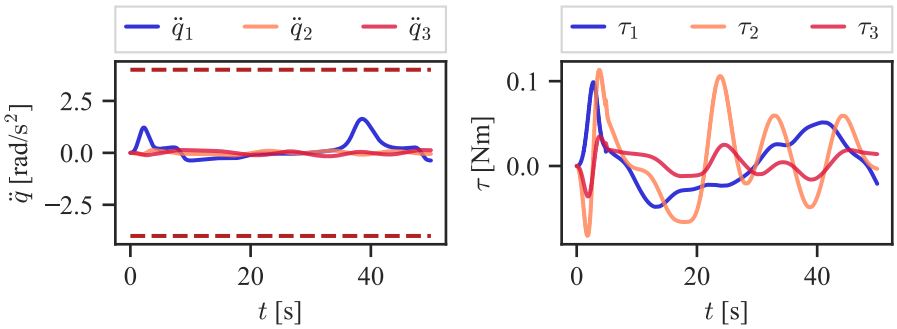


Figure 6.52: Joint acceleration input and computed torque using linear MPC.

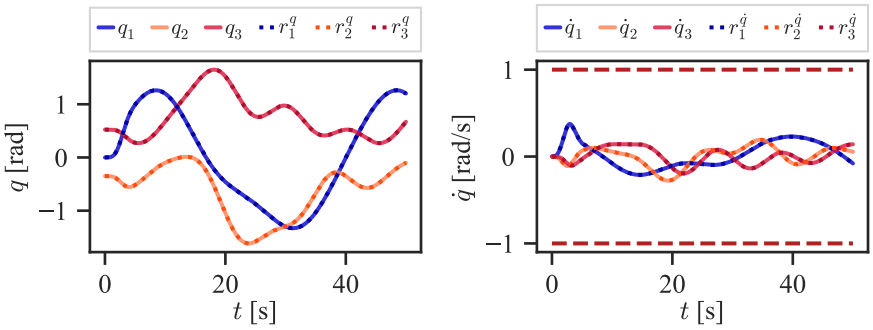


Figure 6.53: Joint angles and joint velocities for space manipulator following joint space trajectory with linear MPC.

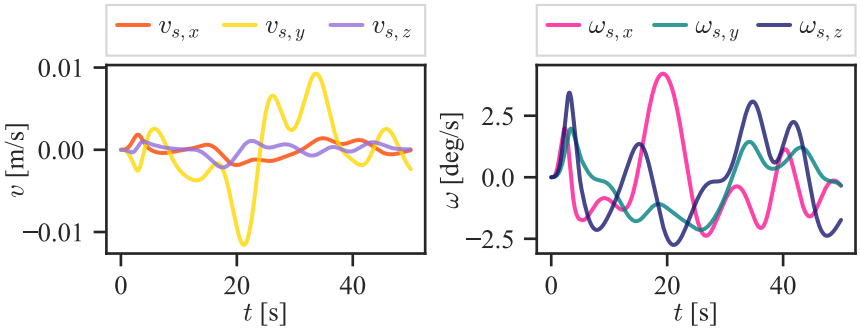


Figure 6.54: Linear and angular velocity of satellite body with linear MPC.

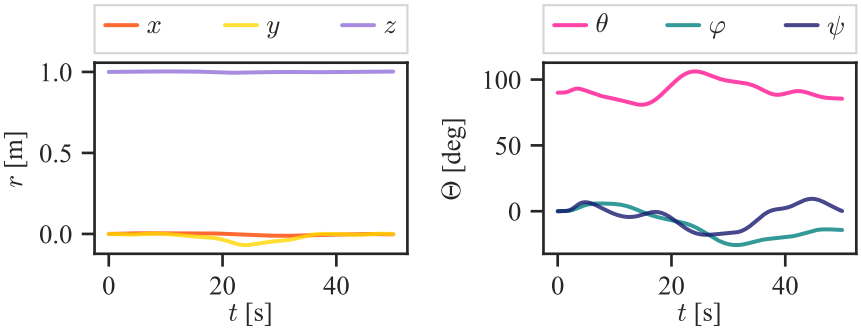


Figure 6.55: Position and orientation of satellite body with linear MPC.

It is observed that the satellite body orientation, and to a lesser extent the position, shifts around while the arm is moving. This is to be expected, as the manipulator mass is a considerable part of the total mass of the system, such that the coupling between the base and the arm is non-negligible. This is partly what makes the considered control problem challenging, especially when the manipulator arm only has 3 DOF. A significant tracking error is therefore seen in Figure 6.56.

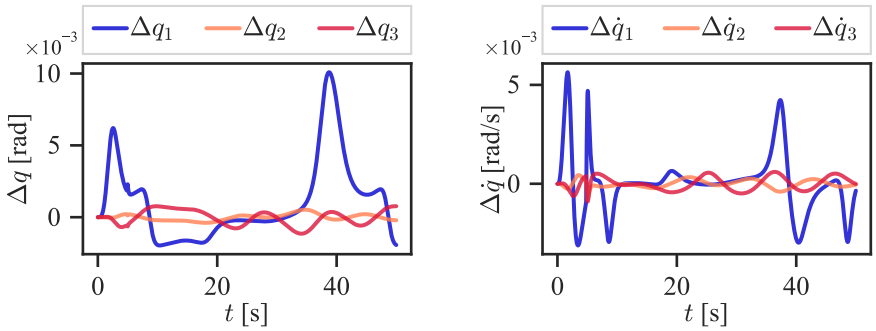


Figure 6.56: Joint trajectory tracking error for space manipulator with linear MPC.

6.4.2 Joint space GP-based MPC test

A GP was trained on the training set collected in the previous section using linear MPC with a total of 3000 samples, for 80 000 iterations using the SVGP method with $\widetilde{M} = 25$ inducing points. The resulting closed loop GP prediction is shown in Figure 6.57, which is seen to follow the disturbance satisfactory. This indicates that GPR is successful in modeling the disturbance, also for the considerable more complex dynamics of floating-base systems.

The corresponding tracking error is given in Figure 6.58, showing an improvement from the linear MPC without GP dynamics, mostly by reducing the severity of the error spikes in q_1. This can be understood by comparing to Figure 6.52, and seeing how the largest tracking errors in q_1 correspond to where the commanded acceleration is the highest, meaning the model mismatch will have the largest impact, which is seen as spikes in the disturbance in Figure 6.57.

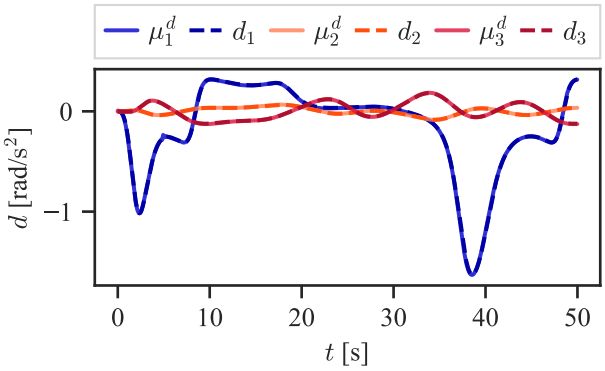


Figure 6.57: GP disturbance prediction μ^d and true disturbance d.

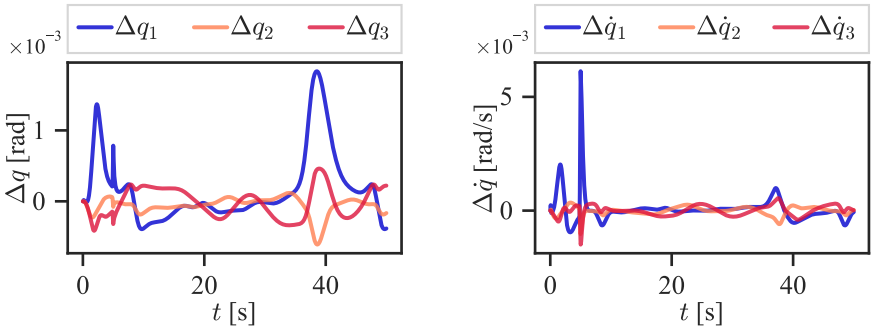


Figure 6.58: Joint trajectory tracking error for space manipulator with GP-MPC.

6.4.3 Joint space NMPC test

The simulation was repeated with NMPC control with the dynamics discretized using ERK4. The resulting tracking error is given in Figure 6.59. It is observed that NMPC struggles with the same error spikes for joint 1 as the linear MPC, which, as discussed, is a consequence of the model mismatch. However, the error is lower, likely a result of including the dynamics directly in the optimization problem, rather than doing feedback linearization, which will be discussed further in Chapter 7.

The main results are shown in Table 6.21. It is seen how including the GP disturbance model decreases the prediction error by about two magnitudes. Furthermore, a significant decrease in tracking error is seen with GP-MPC.

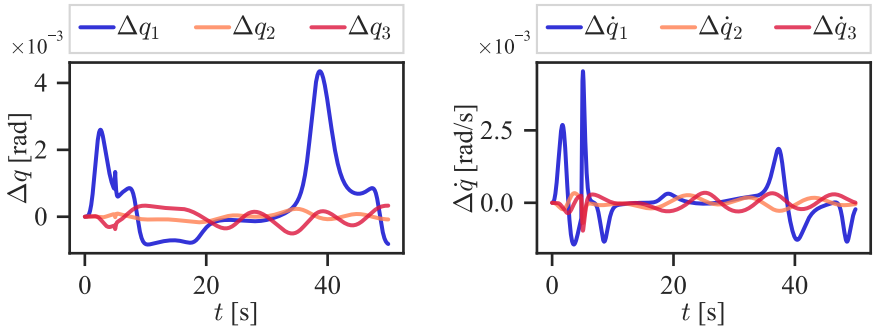


Figure 6.59: Joint trajectory tracking error for space manipulator with NMPC.

Table 6.21: Comparison of trajectory tracking RMSE, prediction RMSE and computation time for linear MPC, GP-MPC, and forward dynamics-based NMPC, for space manipulator joint space trajectory tracking.

Test	$\text{RMSE}_q [\text{rad}]$	$\text{RMSE}_{\text{pred}}$	$t_{\text{solver}} [\text{ms}]$
Linear MPC	$3.042 \cdot 10^{-3}$	$4.900 \cdot 10^{-3}$	0.835
GP-MPC	$5.830 \cdot 10^{-4}$	$4.703 \cdot 10^{-5}$	2.979
NMPC	$1.311 \cdot 10^{-3}$	$4.898 \cdot 10^{-3}$	20.227

The average RTI computation time for NMPC is $20.83\,\text{ms}$, which is considerably higher than the two other approaches based on feedback linearization. This is because the complexity of NMPC grows quickly with the complexity of the nonlinear dynamics when it is directly added as constraints in the OCP. The other approaches, however, use double integrator dynamics, which are independent of the complexity of the dynamics. A large difference in computation time is observed since the floating-base dynamics Eq. (5.39) are considerably more complex than the fixed-base dynamics Eq. (2.26). However, the space manipulator dynamics were implemented with the general Coriolis matrix formula in Eq. (5.41), using the algorithmic differentiation functionality of CasADi, and are, as such, not optimized for speed. It is therefore believed that significant speed improvements could be achieved by using a suited rigid body dynamics algorithm instead, such as RNEA, described in Featherstone 2008.

7 Discussion and further work

In this chapter, the results presented in Chapter 6 will be further analyzed and discussed, and a larger outlook on limitations, discrepancies, and further work will be given. First, the general performance of the linear MPC, GP-MPC, and deterministic NMPC controllers will be discussed and compared, with emphasis on the 2 DOF and 6 DOF simulations. Then the solver strategy and computation time will be discussed before the lab results, and space manipulator simulation results will be analyzed in detail.

7.1 General results

Firstly, the results presented in Chapter 6 and Appendix D show how, even for significant or even arguably exaggerated model mismatch, the feedback linearization-based MPC controller worked sufficiently. The assumption of constant joint acceleration for each time step, which was not satisfied, seems not to be too large of a discrepancy when solving the OCP in a receding-horizon fashion. This highlights the inherent robustness gained by closing the loop with MPC, despite model mismatch and simplifications.

A significant improvement was, however, generally seen with the deterministic NMPC approach compared to the linear MPC without GP dynamics, e.g., in Table 6.13, Table 6.21 and Table D.4. While the two approaches are based on the same prior model, the assumption of a constant control input per time step is less severe for NMPC. By optimizing in the torque domain instead of the joint acceleration domain, one no longer needs to assume piecewise constant acceleration per time step, resulting in a more accurate approximation. Furthermore, having the nonlinear dynamics as constraints naturally makes it easier to plan while considering the system nonlinearity directly.

Moreover, the results showed how GP-MPC was able to achieve substantially more accurate predictions over the prediction horizon. In Section 6.1, significantly better predictions were also seen for a different trajectory than what was used for training.

However, for the more complex dynamics of the 6 DOF fixed-base manipulator in Section 6.2 and the free-floating space manipulator system in Section 6.4, the accuracy of the disturbance predictions were local to some vicinity of the training trajectory. Thus for other test trajectories, the prediction over the prediction horizon was even worse than by not including the GP all together, resulting in oscillatory behavior, as mentioned earlier in Section 6.2.1.2.

There are several reasons for this local behavior of the GP predictions. Firstly, the collected datasets only cover a limited part of the state space, and for higher-dimensional systems, this is only amplified, known as the "curse of dimensionality." Furthermore, the size of the datasets is somewhat small, considering the high dimensionality and highly nonlinear dynamics. In order to achieve real-time feasible computation time, sparse GP methods were used with a low number of inducing points, which significantly limited the expressiveness of the resulting GPs. This resulted in a large bias in the model towards the data it had seen, which for the 6 DOF case was so significant that the controller only worked in a small vicinity of the target trajectory. Especially the SVGP method suffered from this problem, likely because SGD needs more iterations to converge, and the SVGP method has more optimization variables, leading to a more difficult optimization problem. These discussed problems severely limited the usability of GP-MPC, as it effectively means only this single trajectory reference could be used for complex and high-dimensional dynamics.

There are several possible directions for further work on how to remedy this limitation. Firstly, larger datasets could be used, possibly by concatenating several different trials, but this comes at the expense of longer training times. This was also briefly tested by concatenating five different trajectories, but no significant improvements were seen.

An interesting question one could then explore is how to choose test trajectories that excite the system such that the GP generalizes the best possible way. This problem has some similarities to traditional system identification, where one typically optimizes a trajectory, such as a Fourier series trajectory, in order to maximize some measure of the information content in the resulting data, for example, as seen in Swevers et al. 2007. This does, however, seem difficult to do for learning GP models, which are non-parametric.

Another option is to explore an online learning approach, where the inducing points are updated online. In Kabzan et al. 2019, a dictionary of active data points is updated continuously as new measurements are collected. In Hewing, Kabzan, et al. 2019, the

inducing points are chosen along the predicted state trajectory, which is found by shifting the previous solution trajectory. Either of these approaches has potential to help with these local results.

Despite the limited space where the trained model was valid for complex dynamics, the resulting improvement in trajectory tracking error was significant for the 6 DOF simulations. For the joint space trajectory tracking results in Table 6.11, including the GP dynamics resulted in a 51% decrease in joint RMSE, and in the task space case a decrease of 60% and 79% for position RMSE and Euler angle RMSE, respectively. Moreover, the GP-MPC approach mostly outperformed deterministic NMPC in the fixed-base simulations. However, it must be emphasized that these simulations had significant model mismatch added to emulate applications with highly uncertain dynamics.

Furthermore, one important benefit of GP-MPC, as compared to the other deterministic approaches, is that the stochastic nature of including a GP disturbance model in the OCP allows to handle uncertainty. Formulating chance constraints allows creating a controller that is inherently cautious and adjusts the state bounds based on how uncertain the prediction currently is. However, this cautiousness assumes that the information given by the GP is exact, which naturally is not the case. Therefore, there is no guarantee that the chance constraints are satisfied, even if all approximations are ignored, and the OCP is solved exactly.

For the task space trajectories, this uncertainty information was not used to its full potential, as the forward kinematics were evaluated for the state mean in the cost function, as discussed in Section 5.1.2. Further work could be done on propagating the uncertainty from the joint states to the end effector pose, possibly using a Monte Carlo approach or using chaos polynomial expansions. This information would evidently be useful, despite the presumably high computational demands, as it would effectively tell how uncertain the current end effector position and orientation are.

A significant limitation of stochastic MPC, and especially nonlinear stochastic MPC, is the complicated and often even intractable optimization problems that emerge. As seen in this work, even after applying simplifications and approximations to get a tractable and deterministic OCP, it still has high computational demands.

7.2 Solver and computation time

Analyzing the solver computation time in further detail, it was seen that fast enough computation times could be achieved to meet the real-time demands of most robotic systems. For the linear trajectory tracking MPC controller, the solver computation time was generally around the $1\,\mathrm{kHz}$ range, which is especially noteworthy. Since the experiments completed in this work generally had sample frequencies in the $100\,\mathrm{Hz}$ range, even better performance could likely be achieved for linear MPC by lowering the sample rate further towards this limit.

Furthermore, for both GP-MPC and NMPC real-time feasible computation times were seen. For the 6 DOF fixed-base simulations, their respective average computation times were comparable in size. Yet, the feedback linearization-based approach generally scales better since the dynamics for each joint consist of a decoupled double integrator prior and a GP prediction that amounts to a linear combination of kernel function evaluations. So GP-MPC mainly increases in complexity from increased dimensionality and not in how complex the prior dynamics are. The forward dynamics NMPC approach, on the other hand, scales quickly with the nonlinearity of the dynamics model, as the nonlinear dynamics equality constraints have to be propagated over the prediction horizon.

SQP RTI was used to achieve these real-time feasible computation times. While it seems the sample times were low enough for the RTI approximation to work well, it must be noted that RTI is a sub-optimal strategy. In general, RTI does not guarantee constraint satisfaction since the QPs are only solved once per iteration and not repeatedly until convergence. In the process, theoretical rigor is lost, yet the results show that the method works well in practice. A related direction of further work is to investigate how suboptimal the RTI solutions are for trajectory tracking applications, for instance, by comparing with a standard SQP algorithm ran until convergence, or an interior point solver like IPOPT, given in Wächter and Biegler 2006.

Even though only a QP is solved at every time step, there is a significant variation in computation time. This is, for instance, seen in Figure 6.30, where certain outliers result in the sample rate needing to be much lower than otherwise necessary. This highlights a practical limitation of receding-horizon optimization-based approaches in general, in that there might be a significant spread in the time needed to solve the optimization problem. In this case, the outliers are likely a result of the robot reaching some configuration where the

solver is struggling to find a feasible solution. This is, for instance, seen at the beginning of the space manipulator joint space tracking results in Figure 6.58, where error spikes are present likely as a result of the solver not finding a good solution, and thereby not generating a smooth control input.

Another concern with non-convex nonlinear programming approaches, which the GP-MPC and deterministic NMPC fall under, is that even if the solver finds a minimum, it is only local and might, therefore, get stuck in a sub-optimal local minimum. This can also be a problem while training the GP hyperparameters. A related problem, which was seen for training in most of the tests in Chapter 6, is that the optimization landscape might be flat near the optimum, such that the optimizer never really converges. This can be seen in Figure 6.12, where some of the length scales do not converge, even though the ELBO cost converges very quickly.

Another direction for further work is to look further into discretization and numerical optimal control methods. In this work, multiple-shooting was used with uniformly spaced nodes. It would, however, be interesting to try non-uniform spacing of nodes, as well as alternative numerical optimal control methods, especially direct collocation methods.

7.3 Lab results

For the UR10e lab results, it was seen that NMPC generally had lower trajectory tracking error. For this case, the advantage of the more accurate model likely did not outweigh the benefit of planning in the torque domain over the joint acceleration domain, as the prior model was likely accurate enough. Specifically for task space trajectory tracking, NMPC achieved a position RMSE of about $1\,\mathrm{mm}$ and Euler angle RMSE of about $0.05\,\mathrm{deg}$, as seen in Table 6.15, which is sufficient for many applications.

The control allocation simplification of using the predicted joint velocities as control input did, however, seem to induce a lot of oscillations, as seen in Figure 6.45. Furthermore, this simplification means the computed control torques are not actually the same that the internal robot controller generates, which means the torque constraints in the deterministic NMPC especially lose some rigidity.

This work has only considered different model-predictive approaches to trajectory tracking based on the complete dynamics of the manipulator system. The need for such methods, as compared to more traditional and established methods which are purely

reactive, such as the inverse differential kinematics in Eq. (2.25) or feedback linearization-based PD control, as discussed in Eq. (5.1), should be brought into question. One could argue the ability of predictive methods to plan into the future allows for more agile motion. However, it is arguably not needed for most applications of fixed-base robot manipulators. The results on the UR10e, where the forward dynamics-based NMPC approach performed best, further supports this claim. Nevertheless, for situations with significant model mismatch, GP-MPC is still of interest.

7.4 Space manipulator results

Model-predictive trajectory tracking approaches are arguably more interesting for floating-base systems, where the ability to plan ahead and take the coupling between the base pose and the link motion into account is essential. This was tested for a free-floating space manipulator in Section 6.4, which showed how trajectory tracking MPC also works satisfactorily for floating-base systems, both with the feedback linearization approach and the NMPC approach. For joint space trajectories, a significant improvement was seen by adding the GP dynamics, with GP-MPC performing better than NMPC. In the task space case given in Appendix D, however, the reduction in trajectory tracking error was minimal, and NMPC performed best, yet still with a position RMSE of about $11\,\mathrm{mm}$. This error would likely not satisfy the requirements needed for accurately gripping an object in a real-world scenario, like gripping another target satellite or debris. This large error is likely caused by the arm used for simulation only having 3 DOF, which makes the tracking problem challenging with a floating base. It would therefore be interesting for future work to test GP-MPC and NMPC for full pose trajectory tracking with a 6 or 7 DOF free-floating manipulator arm.

Furthermore, the approximation of static base pose over the prediction horizon might also explain this larger error. However, considering the zero initial momenta assumption, this simplification is still reasonable. For future work, it would be interesting to compare this approach to an NMPC approach where the body pose dynamics are included in the optimization problem like in Rybus et al. 2017, to analyze the impact of this approximation further.

The limitation of the zero initial momenta assumption should also be discussed in further detail. While this certainly does pose some limitations for the use of the

developed controllers, it is arguably not that limiting. Suppose the manipulator arm is used, for instance, to grasp some object. In that case, it is fair to assume that the Attitude Determination and Control System (ADCS) of the satellite has already stabilized the spin of the satellite before the grasping operation initiates. The ADCS is then turned off while the grasping occurs, which fits within the given assumptions.

Further work on generalizing to non-zero initial momentum, as well as to consider additional control torques on the satellite body from the ADCS, e.g., using magnetorquers or reaction wheels, is, however, interesting. Adding cost terms to either minimize attitude disturbance while following the trajectory or to track a desired attitude with body torques simultaneously could also be done.

Finally, this work generally assumes all state variables are easily measurable, such that no state estimation methods are needed. While this is a fair assumption for fixed-base industrial robot manipulators, for space manipulator systems, attitude estimation is essential. Further work could, therefore, also consider state estimation, for instance, with a Kalman filter approach like the error-state Kalman filter or a moving-horizon estimation approach.

8 Conclusion

In this work, NMPC approaches for trajectory tracking for robot manipulator systems with uncertain dynamics were investigated. The primary discussed approach, GP-MPC, uses a GP to learn the residual error dynamics between a prior feedback linearized dynamics model and the true robot dynamics. Sparse GP methods were applied to reduce the computational complexity. Moreover, the SQP RTI solver strategy was used to achieve real-time feasible computation times. This method was compared to an NMPC approach using the prior forward dynamics model directly, as well as the feedback linearization-based MPC approach without the additive GP error dynamics. Results were presented for a fixed-base 6 DOF UR10e robot, both in simulation and on a real robot, for both task space and joint space trajectories. Results were also presented for a space manipulator system to test the discussed predictive control methods for a floating-base system. Model mismatch from uncertain model parameters and viscous friction was added in the test cases to have a significant residual between the prior and true model.

It was found that a significant increase in prediction accuracy and trajectory tracking accuracy could be achieved in the fixed-base simulations when including the learned GP model, compared to the NMPC approach using only the prior dynamics model. However, for the 6 DOF robot model, the accuracy of the GP prediction was only local and did not generalize satisfactory to other trajectories. Furthermore, the FITC, VFE and SVGP sparse GP methods were tested in the GP-MPC controller. The SVGP method was primarily used, for its significantly faster training time compared to VFE and FITC. However, it was observed that this came at the cost of worse prediction accuracy.

For the trajectory tracking experiments with the UR10e robot, it was found that adding the GP disturbance model improved the prediction accuracy. Yet the deterministic NMPC generally performed better than GP-MPC. For the simulation results with the space manipulator, GP-MPC performed best for the joint space trajectory, while the deterministic NMPC performed best in the task space case. However, the results have

generally shown that MPC approaches to robot manipulator trajectory tracking can achieve high precision and real-time capable computation time, also for systems with uncertain and highly nonlinear dynamics.

A Conversions between rotation representations[†]

This appendix presents additional conversion formulas between the rotation representations discussed in Section 2.1. The appendix is based on Diebel 2006 and Egeland and Gravdahl 2002.

A.1 Euler angles - unit quaternion conversion

The transformation from ZYX Euler angles $\boldsymbol{\Theta} = [\,\phi \;\; \theta \;\; \psi\,]^\top$ to a unit quaternion $\boldsymbol{q} = [\,\eta \;\; \varepsilon_1 \;\; \varepsilon_2 \;\; \varepsilon_3\,]^\top$ 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 the shorthand $s_\alpha = \sin(\alpha), c_\alpha = \cos(\alpha)$ is used. The inverse transformation is given by

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

[†] This chapter is adapted from Brandt 2020.

A.2 Rotation vector - unit quaternion conversion

From Eq. (2.8) it is seen that a rotation vector $\boldsymbol{v}$ can be mapped to a unit quaternion $\boldsymbol{q}$ by

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

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

The map from a rotation vector $\boldsymbol{v}$ to rotation matrix $\boldsymbol{R}$ 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

$$\boldsymbol{v} = \frac{1}{2}\begin{bmatrix} \boldsymbol{R}_{3,2} - \boldsymbol{R}_{2,3} \\ \boldsymbol{R}_{1,3} - \boldsymbol{R}_{3,1} \\ \boldsymbol{R}_{2,1} - \boldsymbol{R}_{1,2} \end{bmatrix}. \tag{A.9}$$

A.4 Rotation matrix - unit quaternion conversion

Finally, the transformation from unit quaternion $\boldsymbol{q}$ to rotation matrix $\boldsymbol{R}$ 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 of these mappings is

$$\boldsymbol{q}(\boldsymbol{R}) = \frac{1}{2}\begin{bmatrix} \left(1 + \boldsymbol{R}_{1,1} + \boldsymbol{R}_{2,2} + \boldsymbol{R}_{3,3}\right)^{\frac{1}{2}} \\ \left(\boldsymbol{R}_{2,3} - \boldsymbol{R}_{32}\right) / \left(1 + \boldsymbol{R}_{1,1} + \boldsymbol{R}_{2,2} + \boldsymbol{R}_{3,3}\right)^{\frac{1}{2}} \\ \left(\boldsymbol{R}_{3,1} - \boldsymbol{R}_{1,3}\right) / \left(1 + \boldsymbol{R}_{1,1} + \boldsymbol{R}_{2,2} + \boldsymbol{R}_{3,3}\right)^{\frac{1}{2}} \\ \left(\boldsymbol{R}_{1,2} - \boldsymbol{R}_{2,1}\right) / \left(1 + \boldsymbol{R}_{1,1} + \boldsymbol{R}_{2,2} + \boldsymbol{R}_{3,3}\right)^{\frac{1}{2}} \end{bmatrix}, \tag{A.11}$$

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.

B Manipulating Gaussians

In this appendix frequently used formulas for manipulating Gaussians are given. Firstly, given two Gaussian distributed random variables $\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{x}_1 \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\boldsymbol{x}_2 \sim \mathcal{N}(\boldsymbol{x}_2 \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, their joint distribution is also Gaussian, with mean and covariance according to

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{12}^\top & \boldsymbol{\Sigma}_2 \end{bmatrix} \right), \tag{B.1}$$

where $\boldsymbol{\Sigma}_{12}$ is the cross-covariance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Furthermore, given such a joint Gaussian, the conditional distribution of $\boldsymbol{x}_1$ given $\boldsymbol{x}_2$ is

$$p(\boldsymbol{x_1} \mid \boldsymbol{x_2} = \boldsymbol{a}) \sim \mathcal{N}\left(\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{a} - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_{12}^\top\right). \tag{B.2}$$

Finally, for $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$, the linear transformation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ is Gaussian distributed with

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top). \tag{B.3}$$

C Modeling of 2 DOF planar robot manipulator

This appendix presents modeling of the planar 2 DOF robot manipulator arm depicted in Figure 6.1, based on Siciliano et al. 2010 and Egeland and Gravdahl 2002. The model was used in the illustrative example presented in Section 6.1. The mass, inertia (principal axis in z) and length of link i are denoted as m_i, I_i and ℓ_i, respectively, for $i \in \mathbb{N}_2$. Furthermore, it is assumed that the distance to the link center of mass is half the link length for all links.

Firstly, the forward kinematics of the 2 DOF manipulator arm are given by

$$\boldsymbol{r} = \begin{bmatrix} r_x \\ r_y \end{bmatrix} = \begin{bmatrix} \ell_1 \cos q_1 + \ell_2 \cos(q_1 + q_2) \\ \ell_1 \sin q_1 + \ell_2 \sin(q_1 + q_2) \end{bmatrix}, \tag{C.1}$$

and the inverse kinematics are given by

$$\begin{aligned} q_2 &= \pm \arccos\left(\frac{r_x^2 + r_y^2 - \ell_1^2 - \ell_2^2}{2\ell_1\ell_2}\right), \\ q_1 &= \arctan\left(\frac{r_y}{r_x}\right) \mp \arccos\left(\frac{r_x^2 + r_y^2 + \ell_1^2 - \ell_2^2}{2\ell_1\sqrt{r_x^2 + r_y^2}}\right). \end{aligned} \tag{C.2}$$

The inverse differential kinematics are formulated using the Jacobian inverse like in Eq. (2.24), with the Jacobian of the 2 DOF planar manipulator system being

$$\boldsymbol{J} = \begin{bmatrix} -\ell_1 \sin q_1 - \ell_2 \sin(q_1 + q_2) & -\ell_2 \sin(q_1 + q_2) \\ \ell_1 \cos q_1 + \ell_2 \cos(q_1 + q_2) & \ell_2 \cos(q_1 + q_2) \end{bmatrix}. \tag{C.3}$$

The reader is referred to Siciliano et al. 2010 for details on the derivation of the kinematics and Jacobian for the 2 DOF planar arm.

The dynamics of the system are given in Egeland and Gravdahl 2002 as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{C.4}$$

with the mass matrix $\boldsymbol{M}(\boldsymbol{q})$ given as

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{bmatrix}, \tag{C.5}$$

with elements

$$\begin{aligned} M_{11} &= I_1 + I_2 + m_1 \left(\frac{\ell_1}{2}\right)^2 + m_2 \left(\ell_1^2 + \left(\frac{\ell_2}{2}\right)^2 + \ell_1 \ell_2 \cos q_2\right), \\ M_{12} &= I_2 + m_2 \left(\frac{\ell_2}{2}\right)^2 + m_2 \ell_1 \frac{\ell_2}{2} \cos q_2, \\ M_{22} &= I_2 + m_2 \left(\frac{\ell_2}{2}\right)^2. \end{aligned} \tag{C.6}$$

The Coriolis and centripetal matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is given by

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} m_2 \ell_1 \ell_2 \sin q_2 \begin{bmatrix} -2\dot{q}_2 & -\dot{q}_2 \\ \dot{q}_1 & 0 \end{bmatrix}, \tag{C.7}$$

and the gravity vector $\boldsymbol{g}(\boldsymbol{q})$ is given by

$$\boldsymbol{g}(\boldsymbol{q}) = \begin{bmatrix} \left(m_1 \frac{\ell_1}{2} + m_2 \ell_1\right) g \cos q_1 + m_2 \frac{\ell_2}{2} g \cos\left(q_1 + q_2\right) \\ m_2 \frac{\ell_2}{2} g \cos\left(q_1 + q_2\right) \end{bmatrix}. \tag{C.8}$$

D Additional results

In this appendix, supplementary results, which were not included in Chapter 6, are presented. First, results from the joint space trajectory tracking experiments with the UR10e robot are presented. Then, simulation results from task space trajectory tracking with the space manipulator system are presented. For both tests, like the others in Chapter 6, the linear MPC, GP-MPC and deterministic NMPC controllers are tested.

D.1 Joint space trajectory tracking for UR10e robot

D.1.1 Joint space linear MPC test

Joint space trajectory tracking was tested on the UR10e robot, on the same trajectory as in Eq. (6.4), but now centered around a new $\boldsymbol{q}_0$ more suited for the environment. The linear MPC controller was tested first, with parameters given in Table D.1. The computed joint acceleration and torque is given in Figure D.1 and the joint states are shown in Figure D.2. The corresponding tracking error is shown in Figure D.3.

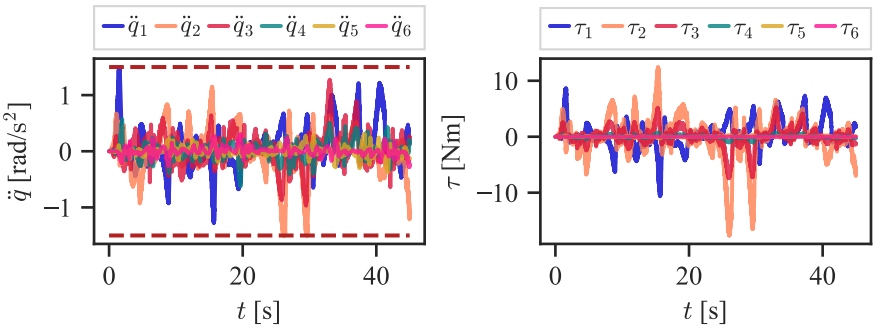


Figure D.1: Joint acceleration input and computed joint torque for joint space trajectory tracking with UR10e using linear MPC.

Table D.1: Parameter values for joint space trajectory tracking test on UR10e robot in lab environment.

Parameter	Value
N	20
t_s [ms]	15.0
$\boldsymbol{q}_0$ [deg]	$[135, -120, -100, -50, 70, 0]$
$q_{\max}$ [rad]	2π
$\dot{q}_{\max}$ [rad/s]	1.0
$\ddot{q}_{\max}$ [rad/s^2]	1.5
ϵ	0.0456
$\boldsymbol{Q}$	$10^3 \cdot \boldsymbol{I}_{12}$
$\boldsymbol{R}$	$\boldsymbol{I}_6$
$\boldsymbol{R}_{\text{NMPC}}$	$\text{blkdiag}(2 \cdot 10^{-6} \cdot \boldsymbol{I}_6, 10^{-2} \cdot \boldsymbol{I}_6)$
$\boldsymbol{S}$	$10^{-2} \cdot \boldsymbol{I}_6$

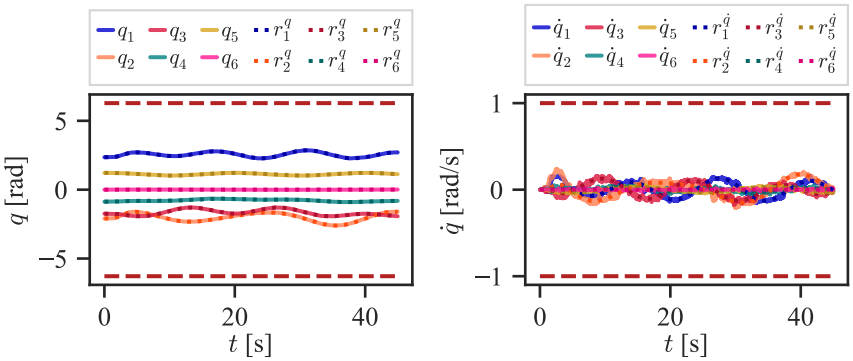


Figure D.2: Joint angles and joint velocities for joint space trajectory tracking with UR10e using linear MPC.

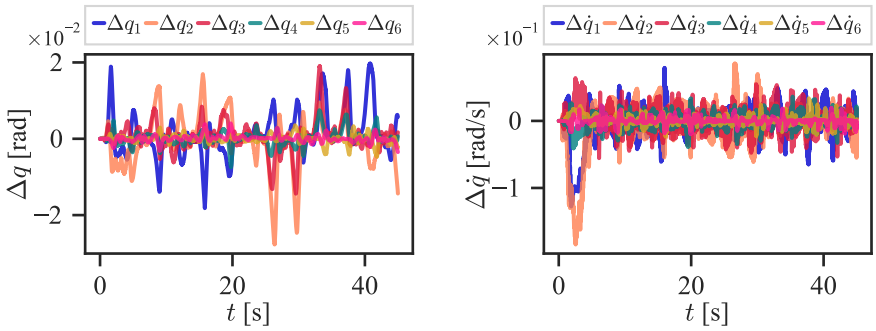


Figure D.3: Joint space trajectory tracking error for UR10e using linear MPC.

D.1.2 Joint space GP-based MPC test

A GP was trained with SVGP for 180 000 iterations with $\widetilde{M} = 25$ inducing points and a dataset of 3000 samples. The resulting trajectory tracking error for the experiment is presented in Figure D.4, showing a considerable improvement by partly mitigating the error spikes.

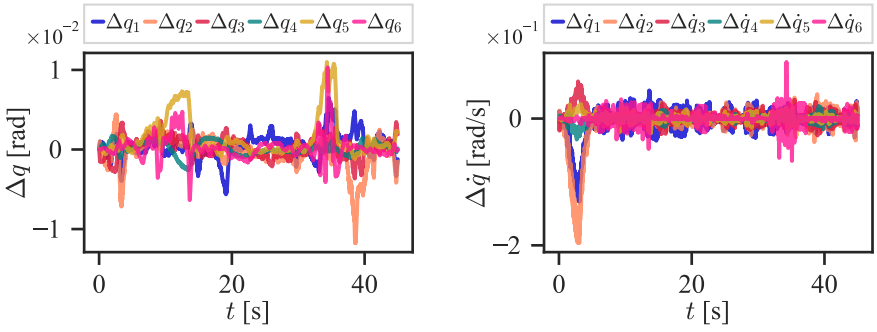


Figure D.4: Joint space trajectory tracking error for UR10e using GP-MPC.

D.1.3 Joint space NMPC test

NMPC was also tested, with ERK4 discretization. The trajectory tracking error for NMPC is shown in Figure D.5.

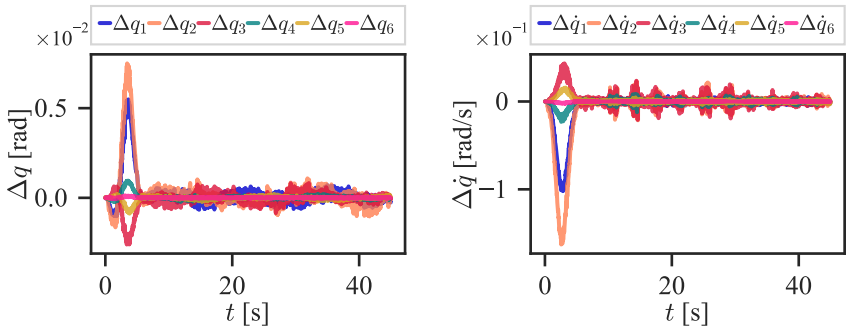


Figure D.5: Joint space trajectory tracking error for UR10e using NMPC.

In Table D.2 the tracking error and solver timings are summarized for the three tests. It is observed that all methods use significantly increased time compared to the simulation case in Section 6.2.2. All methods are still able to stay within the limit of $15\,\mathrm{ms}$, given by the sample rate. Furthermore, while the inclusion of the GP disturbance dynamics does reduce the tracking error, NMPC performs significantly better.

Table D.2: Comparison of joint space trajectory tracking RMSE, prediction RMSE and computation times for linear MPC, GP-MPC and forward dynamics-based NMPC.

Test	$\mathrm{RMSE}_q[\mathrm{rad}]$	$t_{\mathrm{solver}}[\mathrm{ms}]$
Linear MPC	$9.888 \cdot 10^{-3}$	2.123
GP-MPC	$4.513 \cdot 10^{-3}$	9.000
NMPC	$1.669 \cdot 10^{-3}$	8.537

D.2 Task space trajectory tracking for space manipulator

D.2.1 Task space linear MPC test

In order to test task space trajectory tracking for the space manipulator system, a spatial

Lissajous curve is considered in the inertial frame:

$$\boldsymbol{r}^p = a \begin{bmatrix} \sin(\omega t) \\ \sin(n\omega t + \varphi) \\ \sin(m\omega t + \psi) \end{bmatrix}, \tag{D.1}$$

with $a = 0.1$, $\omega = 0.1$, $n = 1$, $m = \frac{1}{2}$, $\varphi = \frac{\pi}{2}$, $\psi = \pi$. The resulting Lissajous curve is visualized in Cartesian space in Figure D.6. Note that only a position trajectory is considered for this example.

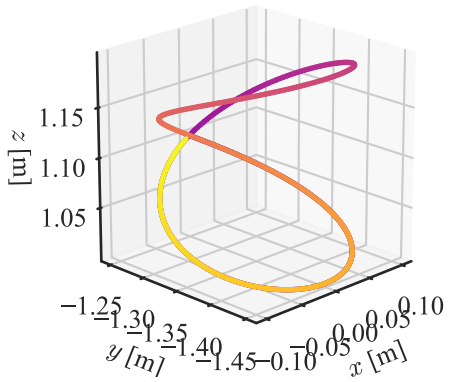


Figure D.6: Lissajous curve in Cartesian space. Time is indicated by the colormap from blue to yellow.

The system was simulated for $T = 50\,\mathrm{s}$ with zero initial momenta. Other initial values and configuration parameters for the simulation are given in Table D.3. The position trajectory was blended from the initial end effector position over a duration of $T_b = 15\,\mathrm{s}$. The inertial frame transform approximation discussed in Section 5.3.2 was used.

The computed joint acceleration and torques for linear MPC are shown in Figure D.7 and the resulting joint states are shown in Figure D.8. In Figure D.9 the blended trajectory reference and the actual end effector position is shown. Snapshots from the simulation are shown with the URDF model in Figure D.10.

The satellite body pose and velocity are shown in Figure D.11 and Figure D.12 respectively. Finally, in Figure D.13 the end effector position and position error is shown.

Table D.3: Parameter values for task space trajectory tracking test of space manipulator in simulation.

Parameter	Value
N	20
t_s [ms]	10.0
$\boldsymbol{p}_0$ [m]	$[0, 0, 1]$
$\boldsymbol{\Theta}_0$ [deg]	$[90, 0, 0]$
$\boldsymbol{q}_0$ [deg]	$[0, 0, 60]$
ϵ	0.0456
$\boldsymbol{Q}$	$10^3 \cdot \boldsymbol{I}_3$
$\boldsymbol{R}$	$\boldsymbol{I}_3$
$\boldsymbol{R}_{\text{NMPC}}$	$\text{blkdiag}(10^{-2} \cdot \boldsymbol{I}_3, 10^{-3} \cdot \boldsymbol{I}_3)$
$\boldsymbol{S}$	$10^{-1} \cdot \boldsymbol{I}_3$
$\boldsymbol{P}$	$20 \cdot \boldsymbol{Q}$

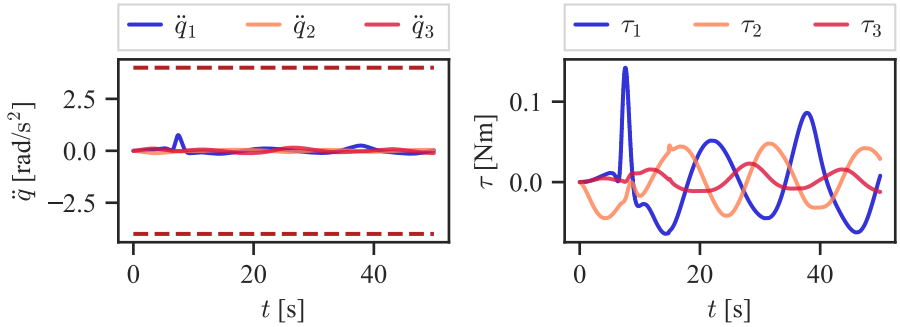


Figure D.7: Joint acceleration input and computed torque using linear MPC.

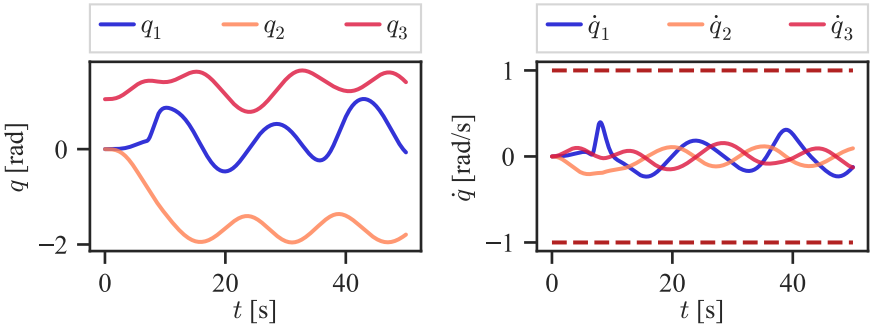


Figure D.8: Joint angles and joint velocities for space manipulator following task space trajectory with linear MPC.

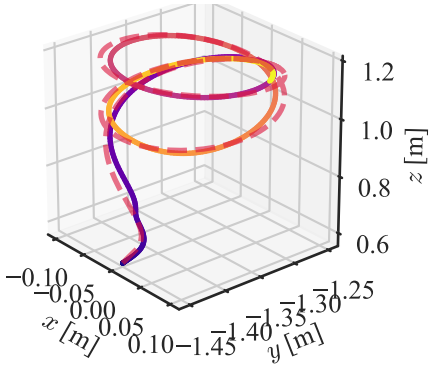


Figure D.9: Desired and actual end effector position trajectory with linear MPC in Cartesian space.

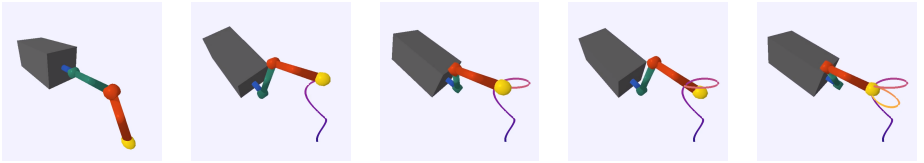


Figure D.10: Space manipulator system tracking Lissajous trajectory in inertial frame.

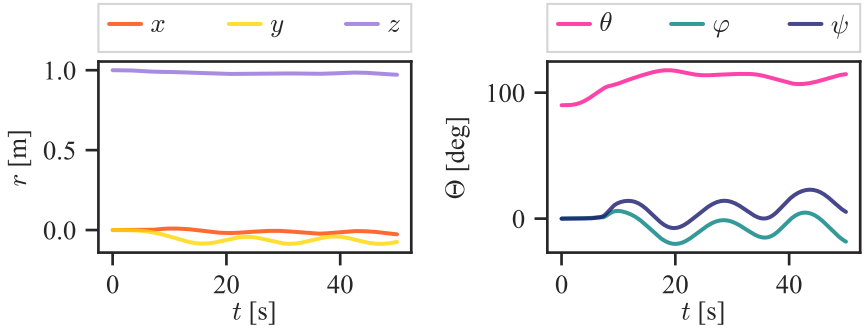


Figure D.11: Position and orientation of satellite body with linear MPC.

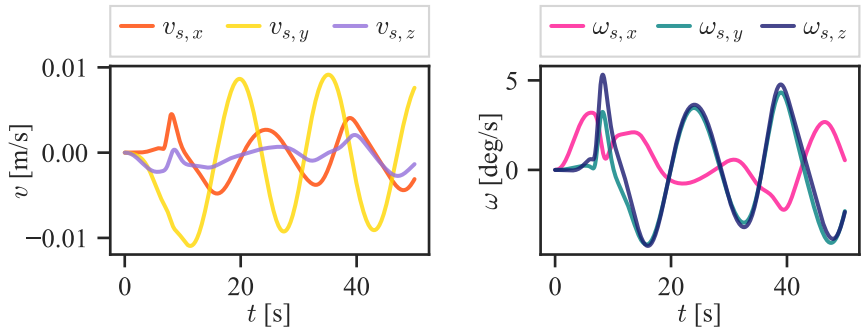


Figure D.12: Linear and angular velocity of satellite body with linear MPC.

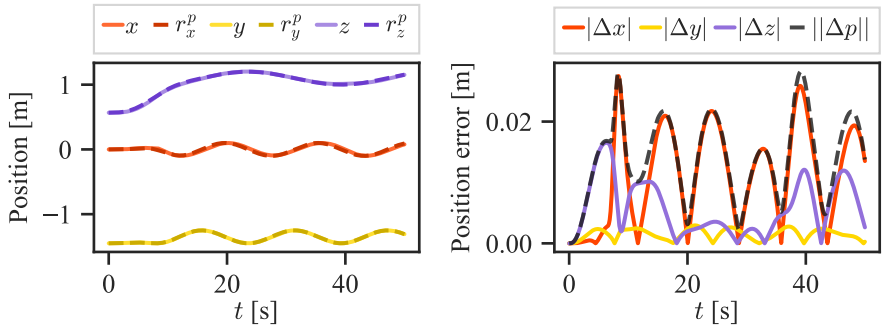


Figure D.13: End effector position trajectory and corresponding tracking error with linear MPC.

D.2.2 Task space GP-based MPC test

A GP was trained with $\widetilde{M} = 25$ for 80 000 iterations using the SVGP method, with the dataset of 10 000 samples collected in the previous section. The closed loop disturbance prediction is shown in Figure D.14. The resulting end effector position and position error are given in Figure D.15.

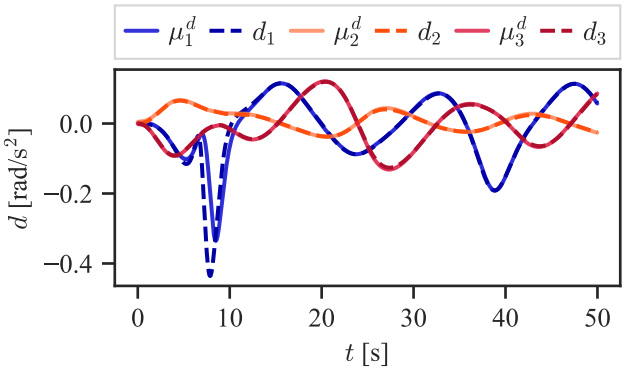


Figure D.14: GP disturbance prediction μ^d and true disturbance d.

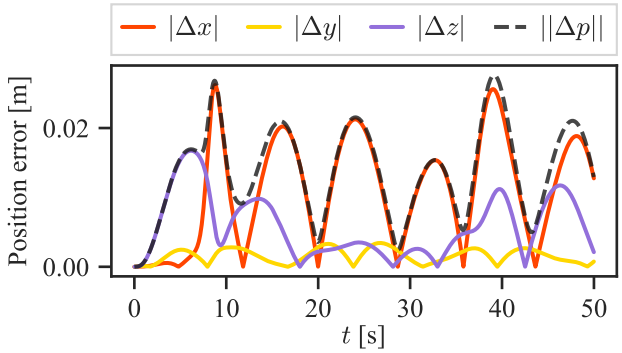


Figure D.15: End effector position tracking error with GP-MPC.

D.2.3 Task space NMPC test

Finally the end effector position and position tracking error are given for deterministic

NMPC in Figure D.16.

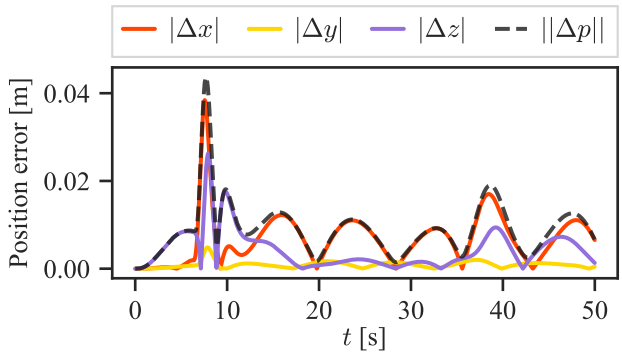


Figure D.16: End effector position tracking error with NMPC.

The main numerical results are presented in Table D.4. It is observed that while the prediction error is significantly lower for GP-MPC, the resulting improvement in tracking error is minimal, and the NMPC strategy performs significantly better.

Table D.4: Comparison of trajectory tracking RMSE, prediction RMSE and computation time for linear MPC, GP-MPC, and forward dynamics-based NMPC, for space manipulator task space trajectory tracking.

Test	RMSE_p[m]	$\text{RMSE}_{\text{pred}}$	t_{solver}[ms]
Linear MPC	$1.567 \cdot 10^{-2}$	$1.239 \cdot 10^{-3}$	0.814
GP-MPC	$1.541 \cdot 10^{-2}$	$3.273 \cdot 10^{-5}$	3.012
NMPC	$1.098 \cdot 10^{-2}$	$1.944 \cdot 10^{-3}$	21.112

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). "TensorFlow: A System for Large-Scale Machine Learning". *12th USENIX Symposium on Operating Systems Design and Implementation*. Savannah, pp. 265–283.

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

Brandt, M. A. (2020). *Nonlinear Model Predictive Control for Robot Manipulator Trajectory Tracking*. Project report. Norwegian University of Science and Technology, Trondheim.

Cao, G., Lai, E. M. K., and Alam, F. (2017). "Gaussian Process Model Predictive Control of an Unmanned Quadrotor". *Journal of Intelligent & Robotic Systems* 88.1, pp. 147–162.

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., and Mansard, N. (2019). "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". *International Symposium on System Integration*. Paris.

Carron, A., Arcari, E., Wermelinger, M., Hewing, L., Hutter, M., and Zeilinger, M. N. (2019). "Data-Driven Model Predictive Control for Trajectory Tracking With a Robotic Arm". *IEEE Robotics and Automation Letters* 4.4, pp. 3758–3765.

Chen, C.-T. (1999). *Linear system theory and design*. Oxford University Press, New York.

Coumans, E. and Bai, Y. (2016–2021). *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. https://pybullet.org.

Diebel, J. (2006). *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Technical report. Stanford University.

Diehl, M., Ferreau, H. J., and Haverbeke, N. (2009). "Efficient numerical methods for nonlinear MPC and moving horizon estimation". *Nonlinear model predictive control*. Springer, Berlin, Heidelberg, pp. 391–417.

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

Egeland, O. and Gravdahl, J. T. (2002). *Modeling and Simulation for Automatic Control*. Marine Cybernetics, Trondheim.

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

Fossen, T. I. (2011). *Guidance and Control of Ocean Vehicles*. John Wiley & Sons, Chichester.

Frison, G. and Diehl, M. (2020). "HPIPM: a high-performance quadratic programming framework for model predictive control". *21th IFAC World Congress* 53.2, pp. 6563–6569.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. (2018). "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration". *Advances in Neural Information Processing Systems*, pp. 7576–7586.

GPy (2021). *GPy: A Gaussian process framework in python*. http://github.com/SheffieldML/GPy.

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

Grüne, L. and Pannek, J. (2011). *Nonlinear model predictive control: Theory and algorithms*. Springer, London.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". *2017 IEEE International Conference on Robotics and Automation*. Singapore, pp. 3389–3396.

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). "Array programming with NumPy". *Nature* 585, pp. 357–362.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). "Gaussian Processes for Big Data". *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*. Bellevue, pp. 282–290.

Hewing, L., Kabzan, J., and Zeilinger, M. N. (2019). "Cautious Model Predictive Control Using Gaussian Process Regression". *IEEE Transactions on Control Systems Technology* 28.6, pp. 2736–2743.

Hewing, L., Liniger, A., and Zeilinger, M. N. (2018). "Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars". *2018 European Control Conference*. Limassol, pp. 1341–1348.

Houska, B., Ferreau, H., and Diehl, M. (2011). "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization". *Optimal Control Applications and Methods* 32.3, pp. 298–312.

Johannessen, L. M. G., Arbo, M. H., and Gravdahl, J. T. (2019). "Robot Dynamics with URDF & CasADi". *2019 7th International Conference on Control, Mechatronics and Automation*. Delft, pp. 185–190.

Johansen, T. A. (2011). "Introduction to nonlinear model predictive control and moving horizon estimation". *Selected topics on constrained and nonlinear control*. Slovak

University of Technology, Bratislava, Norwegian University of Science and Technology, Trondheim, pp. 187–240.

Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M. N. (2019). "Learning-Based Model Predictive Control for Autonomous Racing". *IEEE Robotics and Automation Letters* 4.4, pp. 3363–3370.

Kavan, L. and Žára, J. (2005). "Spherical Blend Skinning: A Real-Time Deformation of Articulated Models". Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games. Washington, pp. 9–16.

Kingma, D. P. and Ba, J. (2015). "Adam: A Method for Stochastic Optimization". *International Conference on Learning Representations*. San Diego.

Koenig, N. and Howard, A. (2004). "Design and use paradigms for Gazebo, an open-source multi-robot simulator". *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 3. Sendai, pp. 2149–2154.

Lindvig, A. P. (2021). *ur_rtde*. Gitlab repository. URL: `https://gitlab.com/sdurobotics/ur_rtde` (visited on 03/22/2021).

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. (2020). "When Gaussian process meets big data: A review of scalable GPs". *IEEE transactions on neural networks and learning systems* 31.11, pp. 4405–4423.

Loquercio, A., Segu, M., and Scaramuzza, D. (2020). "A General Framework for Uncertainty Estimation in Deep Learning". *IEEE Robotics and Automation Letters* 5.2, pp. 3153–3160.

Macfarlane, S. and Croft, E. A. (2003). "Jerk-bounded manipulator trajectory planning: design for real-time applications". *IEEE Transactions on Robotics and Automation* 19.1, pp. 42–52.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). "GPflow: A Gaussian process library using TensorFlow". *Journal of Machine Learning Research* 18.40, pp. 1–6.

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

McKinnon, C. D. and Schoellig, A. P. (2019). "Learn Fast, Forget Slow: Safe Predictive Learning Control for Systems With Unknown and Changing Dynamics Performing Repetitive Tasks". *IEEE Robotics and Automation Letters* 4.2, pp. 2180–2187.

Mesbah, A. (2016). "Stochastic Model Predictive Control: An Overview and Perspectives for Future Research". *IEEE Control Systems Magazine* 36.6, pp. 30–44.

Myhre, T. A. (2016). "Vision-Based Control of a Robot Interacting with Moving and Flexible Objects". PhD thesis. Norwegian University of Science and Technology, Trondheim.

Nanos, K. and Papadopoulos, E. G. (2017). "On the Dynamics and Control of Free-floating Space Manipulator Systems in the Presence of Angular Momentum". *Frontiers in Robotics and AI* 4. Paper 26.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. 2nd ed. Springer, New York.

Parameters for calculations of kinematics and dynamics (2020). Universal Robots. URL: https://www.universal-robots.com/articles/ur/parameters-for-calculations-of-kinematics-and-dynamics (visited on 11/30/2020).

Paulson, J. A., Buehler, E. A., Braatz, R. D., and Mesbah, A. (2020). "Stochastic model predictive control with joint chance constraints". *International Journal of Control* 93.1, pp. 126–139.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12, pp. 2825–2830.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). "A unifying view of sparse approximate Gaussian process regression". *The Journal of Machine Learning Research* 6, pp. 1939–1959.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press, London.

Rawlings, J. B., Mayne, D. Q., and Diehl, M. (2019). *Model Predictive Control: Theory, Computation, and Design*. 2nd ed. Nob Hill Publishing, Santa Barbara.

Rohmer, E., Singh, S. P. N., and Freese, M. (2013). "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework". *Proc. of The International Conference on Intelligent Robots and Systems*. Tokyo.

Rybus, T., Seweryn, K., and Sąsiadek, J. Z. (2017). "Control System for Free-Floating Space Manipulator Based on Nonlinear Model Predictive Control (NMPC)". *Journal of Intelligent & Robotic Systems* 85.3, pp. 491–509.

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

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Springer, London.

Snelson, E. and Ghahramani, Z. (2005). "Sparse Gaussian processes using pseudo-inputs". *Advances in Neural Information Processing Systems* 18, pp. 1257–1264.

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

Swevers, J., Verdonck, W., and De Schutter, J. (2007). "Dynamic Model Identification for Industrial Robots". *IEEE Control Systems Magazine* 27.5, pp. 58–71.

Tipping, M. E. (2004). "Bayesian inference: an introduction to principles and practice in machine learning". In: *Advanced Lectures on Machine Learning*. Berlin, Heidelberg, pp. 41–62.

Titsias, M. (2009). "Variational Learning of Inducing Variables in Sparse Gaussian Processes". *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Vol. 5. Clearwater Beach, pp. 567–574.

Todorov, E., Erez, T., and Tassa, Y. (2012). “MuJoCo: A physics engine for model-based control”. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vilamoura, pp. 5026–5033.

Umetani, Y. and Yoshida, K. (1989). “Resolved motion rate control of space manipulators with generalized Jacobian matrix”. *IEEE Transactions on Robotics and Automation* 5.3, pp. 303–314.

Verschueren, R., Frison, G., Kouzoupis, D., Duijkeren, N. van, Zanelli, A., Novoselnik, B., Frey, J., Albin, T., Quirynen, R., and Diehl, M. (2019). “acados: a modular open-source framework for fast embedded optimal control”. *arXiv:1910.13753*.

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

Wang, M., Luo, J., and Walter, U. (2016). “A non-linear model predictive controller with obstacle avoidance for a space robot”. *Advances in Space Research* 57.8, pp. 1737–1746.

Wilde, M., Kwok Choon, S., Grompone, A., and Romano, M. (2018). “Equations of Motion of Free-Floating Spacecraft-Manipulator Systems: An Engineer’s Tutorial”. *Frontiers in Robotics and AI* 5. Paper 41.

Zanelli, A., Domahidi, A., Jerez, J., and Morari, M. (2017). “FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs”. *International Journal of Control*, pp. 1–17.

Martin Albertsen Brandt

Trajectory Tracking for Fixed-Base and Floating-Base Robot Manipulators

NTNU
Norwegian University of
Science and Technology