Nonlinear Model Predictive Control in the Application of Constrained Manipulator Control

Bachelor Thesis
by
Frederik Ebert

geb. am 16.01.1993
Maduschkastraße 18
80995 München
Tel.: 015770477145

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK (Elektrotechnik)
Technische Universität München

Prof. Dongheui Lee, Ph.D

Lehrstuhl für
REGELUNGSTECHNIK (Maschinenwesen)
Technische Universität München

Prof. Dr.-Ing. habil. Boris Lohmann

Betreuer:	M. Sc. Sang-ik An
Beginn:	14.05.14
Zwischenbericht:	16.09.14
Abgabe:	14.11.14

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbstständig durchgeführt zu haben und keine weiteren Hilfsmittel und Quellen als die angegebenen genutzt zu haben. Mit ihrer unbefristeten Aufbewahrung in der Lehrstuhlbibliothek erkläre ich mich einverstanden.

München, den November 14, 2014 _____

In your final hardback copy, replace this page with the signed exercise sheet.

Abstract

The goal of this thesis is to evaluate model predictive control (MPC) in the context of constrained redundant robot manipulators. At first MPC is applied to linear systems and compared with traditional finite-horizon optimal control. In the second part the model predictive- and finite-horizon kinematic control strategy proposed by Schuetz [1] is implemented. Additionally an alternative MPC and finite horizon control algorithm is developed based on the shooting method. To contrast these methods with the widely-used 'instantaneous' kinematic control, a simulation is carried out, where a redundant manipulator is controlled to perform obstacle avoidance. In this simulation it is shown that Schuetz' algorithm only approximates the optimal solution. This is achieved by demonstrating that the newly suggested method yields lower accumulated cost-values for the finite horizon than Schuetz' method.

Zusammenfassung

Das Ziel dieser Arbeit ist es, modellprädiktive Regelung (MPC) auf redundante Robotermanipulatoren anzuwenden und das Verhalten zu untersuchen. Dazu wird MPC im ersten Teil bei linearen Systemen eingesetzt und analysiert. Im zweiten Teil wird die modellprädiktive Regelung für Roboterkinematiken von Schütz [1] implementiert. Zusätzlich wird eine alternative MPC-Regelung und Optimalsteuerung vorgeschlagen, die auf dem Schieß-Verfahren basiert. In einer Simulation werden diese beiden Methoden mit der weitverbreiteten "instantanen" Kinematik-Regelung verglichen. Es wird ein Szenario simuliert, in dem der Manipulator Hindernissen ausweicht. Dabei wird festgestellt, dass der Algorithmus von Schütz' nur eine Näherung für das Optimum liefert und die neu entwickelte Methode näher am Optimum liegt. Dies wird dadurch gezeigt, dass die Kostenfunktion bei der vorgeschlagenen Methode niedrigere Werte liefert als bei der Methode von Schütz.

Contents

Chapter 1

Introduction

1.1 Motivation

One of the main reasons why robots are not ready to support humans in everyday-life yet is their insufficient ability to adapt to changing tasks and environments. To fulfill its tasks even in complicated environments the robot needs a high degree of redundancy e.g. to avoid obstacles and joint limits. Resolving this redundancy in a useful way poses a major problem to ongoing research. A promising approach is to convert abstract manipulation tasks into simple objective functions that reward for keeping distance to obstacles, avoiding singularities and joint limits as well as minimizing the actuation energy. Up to date the most common approach is to calculate the optimal joint speeds to minimize the cost at the current time-instant only. This is referred to as "local" or "instantaneous" control and benefits from relatively low computational burden. However it does not provide an optimal solution and leads to myopic behavior. It also requires an adjustment of parameters like the collision avoidance gain for each situation. A controller is needed that does not only account for the present situation but also for the future evolution of constraints and tasks by finding the optimal behavior in a moving time window. This leads to the idea of Model-Predictive Control which will be addressed in this thesis.

1.2 Related Work

Model predicitve control originated from the process industry and only recently came into focus of robotics researchers. In 2000 Poignet and Gautier [2] suggested a MPC-style control scheme for a manipulator that consists of performing a feedback linearization and using Predictive Functional Control, a variant of MPC. In 2011 Tassa et al. [3] proposed a method based on differential dynamic programming (DDP) to compute MPC for dynamic systemes like a full-size humanoid robot including contacts. However this algorithm approximates the dynamics and smoothens the contacts making it diffcult to employ it in real-world environments. In 2014 Schuetz [1] suggested a model-predictive kinematic controller to find optimal nullspace-velocities for a redundant manipulator. His work is based on Nakamura's

formulation of the optimal control problem of redundancy.

1.3 Greedy Control

The most primitive way to perform optimal control for a discrete time system is "Greedy Control" [4], herein also called instantaneous optimal control. We assume the following model 1.1 with state $\boldsymbol{x}_k$, input $\boldsymbol{u}_k$ and cost function q:

$$\begin{gathered} \boldsymbol{x}_{k+1} = \boldsymbol{F}(\boldsymbol{x}_k, \boldsymbol{u}_k) \qquad (1.1) \\ k = 0, 1, ..., N \\ \boldsymbol{u}(t) \in \mathcal{U} \in \mathbb{R}^m, \ \boldsymbol{x}(t) \in \mathcal{X} \in \mathbb{R}^n \\ \text{minimize: } \sum_{k=0}^{N} q(\boldsymbol{u}, \boldsymbol{x}_{k+1}, t) \in \mathbb{R} \qquad (1.2) \end{gathered}$$

Then the input of greedy control at time t can be found by solving the m-dimensional minimization problem 1.3 at each time step:

$$\boldsymbol{u}_k = \underset{\boldsymbol{u}_k}{\operatorname{argmin}} \ \{q(\boldsymbol{u}_k, \boldsymbol{x}_{k+1}) \mid \boldsymbol{u}_k \in \mathcal{U}, \ \boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k, \boldsymbol{u}_k) \in \mathcal{X}\} \qquad (1.3)$$

The Controller of 1.3 calculates the input $\boldsymbol{u}_k$ that results in the minimum cost q at instant k. It does neither account for future nor past costs. In most cases using greedy control the cost summed up over the time interval $k = 0, 1, ..., N$ is not the globally optimal cost, i.e. there exists another input sequence $\boldsymbol{u}'_k$ that gives a lower cost:

$$\sum_{k=0}^{N} q_k(\boldsymbol{u}_k, \boldsymbol{x}_{k+1}) \geq \sum_{k=0}^{N} q_k(\boldsymbol{u}'_k, \boldsymbol{x}'_{k+1}) \qquad (1.4)$$

1.4 General Problem Formulation of Optimal Control

Model-predictive control evolved from the methods of optimal control. The general definition of optimal control is:

$$\text{minimize: } \ J = h(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} g(\boldsymbol{x}(t), \boldsymbol{u}(t), t) dt \qquad (1.5)$$

$$\text{subject to : } \ \dot{\boldsymbol{x}} = a(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \qquad (1.6)$$

As stated in [5], the goal of optimal control is to find the input $\boldsymbol{u} \in \mathbb{R}^m$ so that the cost function J in 1.5 is minimized for any kind of non-linear, time-varying model 1.6. The control is operating between an initial time t_0 and a final time t_f. In the cost function J the instantaneous cost $g(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$ is integrated over time and

the final stage cost $h(\boldsymbol{x}(t_f))$ is added to it. Both the input $\boldsymbol{u}$ and the state $\boldsymbol{x}$ have to be within the admissible sets $\mathcal{U}, \mathcal{X}$ respectively.

$$J^* = h(\boldsymbol{x}^*(t_f), t_f) + \int_{t_0}^{t_f} g(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t)dt \tag{1.7}$$

$$\leq h(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} g(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{1.8}$$

$$\forall \boldsymbol{u}(t) \in \mathcal{U}, \quad \boldsymbol{x} \in \mathcal{X} \tag{1.9}$$

Equations 1.7 and 1.8 express that if there exists an optimal input $\boldsymbol{u}^*(t)$ and an optimal state trajectory $\boldsymbol{x}^*(t)$ then these two yield the optimal cost value J^*. This value has to be lower than the cost caused by any other non-optimal input $\boldsymbol{u}(t)$.

Chapter 2

Control of Linear Systems

In this chapter the theory of linear optimal control and linear model-predictive control is revised and several simulations are carried out showing the characteristic properties of these control schemes. This serves as a basis for nonlinear robot kinematic control which will be the focus of chapter 2.

2.1 Linear Optimal Control

This section deals with discrete time systems and controls only. However this does not mean a loss of generality since any continuous linear model can be discretized and controlled with the methods presented here.

$$\text{minimize: } J_0(\boldsymbol{x}_0, \boldsymbol{U}_0) = \boldsymbol{x}'_N \boldsymbol{P} \boldsymbol{x}_N \; + \; \sum_{k=0}^{N-1} (\boldsymbol{x}'_k \boldsymbol{Q} \boldsymbol{x}_k \; + \; \boldsymbol{u}'_k \boldsymbol{R} \boldsymbol{u}_k) \tag{2.1}$$

$$\text{subject to: } \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k, \; k = 0, 1, ..., N-1 \tag{2.2}$$

$$\boldsymbol{Q} = \boldsymbol{Q}' \succeq 0, \; \boldsymbol{P} = \boldsymbol{P}' \succeq 0, \; \boldsymbol{R} = \boldsymbol{R}' \succ 0 \tag{2.3}$$

A discrete time state space model with input vector $\boldsymbol{u}_k \in \mathbb{R}^m$ and state vector $\boldsymbol{x}_k \in \mathbb{R}^n$ and initial state $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ is used. The index k denotes the time step ranging from 0 to N. Matrix $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the stage cost for the state and input respectively whereas $\boldsymbol{P}$ is called the final stage cost. All cost matrices have to be symmetric and positive (semi)-definite to ensure J being positive at all times. J containing the state $\boldsymbol{x}$ will cause the controller to drive the system to the origin with minimal cost. First this section will focus on the regulation problem, later the tracking problem is discussed.

2.1.1 Batch Approach

The most straightforward method to solve the linear optimal control problem 2.1 to 2.3 is by finding a way of expressing the cost J as a function of the the initial state $\boldsymbol{x}(0)$ and the sequence of control inputs $\boldsymbol{U}_0 = [\boldsymbol{u}'_0, ..., \boldsymbol{u}'_{N-1}] \in \mathbb{R}^{mN}$. According to Borelli [6, p. 166] this can be done using the following expression:

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}}_{\mathcal{X}} = \underbrace{\begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A} \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix}}_{\mathcal{S}^{\mathrm{x}}} \boldsymbol{x}_0 + \underbrace{\begin{bmatrix} 0 & \dots\dots & 0 \\ \boldsymbol{B} & 0 \;\dots & 0 \\ \boldsymbol{AB} & \ddots \;\; \ddots & \vdots \\ \vdots & \ddots \;\; \ddots & \vdots \\ \boldsymbol{A}^{N-1}\boldsymbol{B} & \dots\dots & \boldsymbol{B} \end{bmatrix}}_{\mathcal{S}^{\mathrm{u}}} \underbrace{\begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix}}_{\boldsymbol{U}_0} \tag{2.4}$$

In Equation 2.4 the sequence of states is given by the vector $\mathcal{X} = [x_0', ..., x_N'] \in \mathbb{R}^{n(N+1)}$. In shorter form 2.4 can be written as:

$$\mathcal{X} = \mathcal{S}^{\mathrm{x}}\boldsymbol{x}(0) + \mathcal{S}^{\mathrm{u}}\boldsymbol{U}_0 \tag{2.5}$$

After inserting $\mathcal{X}$ and $\boldsymbol{U}_0$ and defining $\bar{\boldsymbol{Q}}$ and $\bar{\boldsymbol{R}}$ the cost function J_0 of 2.1 can be rewritten in the following form:

$$J_0(\boldsymbol{x}(0), \boldsymbol{U}_0) = \mathcal{X}'\bar{\boldsymbol{Q}}\mathcal{X} + \boldsymbol{U}_0'\bar{\boldsymbol{R}}\boldsymbol{U}_0 \tag{2.6}$$

$$\bar{\boldsymbol{Q}} = \mathrm{diag}\{\boldsymbol{Q}, ..., \boldsymbol{Q}, \boldsymbol{P}\} \quad \bar{\boldsymbol{R}} = \mathrm{diag}\{\boldsymbol{R}, ..., \boldsymbol{R}\} \tag{2.7}$$

Note that the final stage cost $\boldsymbol{P}$ has been moved to the last element of the diagonal of the $\bar{\boldsymbol{Q}}$ matrix. Using 2.5 equation 2.6 can be written as follows:

$$J_0(\boldsymbol{x}(0), \boldsymbol{U}_0) = \boldsymbol{U}_0' \underbrace{(\mathcal{S}^{\mathrm{u}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\mathrm{u}} + \bar{\boldsymbol{R}})}_{\boldsymbol{H}} \boldsymbol{U}_0 \tag{2.8}$$

$$+ 2\boldsymbol{x}'(0) \underbrace{(\mathcal{S}^{\mathrm{x}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\mathrm{u}})}_{\boldsymbol{F}} \boldsymbol{U}_0 + \boldsymbol{x}'(0) \underbrace{(\mathcal{S}^{\mathrm{x}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\mathrm{x}})}_{\boldsymbol{Y}} \boldsymbol{x}(0) \tag{2.9}$$

$$= \boldsymbol{U}_0'\boldsymbol{H}\boldsymbol{U}_0 + 2\boldsymbol{x}'(0)\boldsymbol{F}\boldsymbol{U}_0 + \boldsymbol{x}'(0)\boldsymbol{Y}\boldsymbol{x}(0) \tag{2.10}$$

Hence J_0 is a quadratic form depending on the input sequence $\boldsymbol{U}_0$ and the initial state $x(0)$. The optimal input is obtained by computing the gradient of J_0 with respect to $\boldsymbol{U}_0$ and setting to zero.

$$\frac{\partial}{\partial \boldsymbol{U}_0} J_0 = 0 \tag{2.11}$$

$$\boldsymbol{U}_0^*(\boldsymbol{x}_0) = -\boldsymbol{H}^{-1}\boldsymbol{F}'\boldsymbol{x}(0) \tag{2.12}$$

$$\boldsymbol{U}_0^*(\boldsymbol{x}_0) = -(\mathcal{S}^{\mathrm{u}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\mathrm{u}} + \bar{\boldsymbol{R}})^{-1}(\mathcal{S}^{\mathrm{x}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\mathrm{u}})'\boldsymbol{x}(0) \tag{2.13}$$

In equations 2.12 and 2.13 we can see that the optimal input sequence is solely depending on the initial state and the model information being the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

Solving the Tracking Problem The batch approach discussed above deals with the regulation problem. The tracking problems can be solved by introducing the reference vector $\boldsymbol{w}_k \in \mathbb{R}^n$ for $k = 1...N$ and subtracting it from the state vector $\boldsymbol{x}_k$ in the cost function:

$$\text{min: } J_0(\boldsymbol{x}_0, \boldsymbol{U}_0) = (\boldsymbol{x}_N - \boldsymbol{w}_N)'\boldsymbol{P}(\boldsymbol{x}_N - \boldsymbol{w}_N) \; + \; \sum_{k=0}^{N-1}[(\boldsymbol{x}_k - \boldsymbol{w}_k)'\boldsymbol{Q}(\boldsymbol{x}_k - \boldsymbol{w}_k) \; + \; \boldsymbol{u}_k'\boldsymbol{R}\boldsymbol{u}_k)]$$

By defining $\mathcal{W} = [\boldsymbol{w}_1', ..., \boldsymbol{w}_{N-1}']'$ the equation above can be written in batch format:

$$J_0(\boldsymbol{x}(0), \boldsymbol{U}_0) = (\mathcal{X}' - \mathcal{W}')\bar{\boldsymbol{Q}}(\mathcal{X} - \mathcal{W}) + \boldsymbol{U}_0'\bar{\boldsymbol{R}}\boldsymbol{U}_0 \tag{2.14}$$

$$\bar{\boldsymbol{Q}} = \text{diag}\{\boldsymbol{Q}, ..., \boldsymbol{Q}, \boldsymbol{P}\} \quad \bar{\boldsymbol{R}} = \text{diag}\{\boldsymbol{R}, ..., \boldsymbol{R}\} \tag{2.15}$$

After computing the multiplication and substituting by several variables 2.14 becomes:

$$J_0 = \boldsymbol{U}_0'\boldsymbol{H}\boldsymbol{U}_0 + 2(\boldsymbol{x}'(0)\boldsymbol{F} - \boldsymbol{k})\boldsymbol{U}_0 + L \tag{2.16}$$

$$\boldsymbol{H} = \mathcal{S}^{\text{u}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\text{u}} + \bar{\boldsymbol{R}} \tag{2.17}$$

$$\boldsymbol{F} = \mathcal{S}^{\text{x}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\text{u}} \tag{2.18}$$

$$\boldsymbol{Y} = \mathcal{S}^{\text{x}\prime}\bar{\boldsymbol{Q}}\mathcal{S}^{\text{x}} \tag{2.19}$$

$$\boldsymbol{k} = \mathcal{W}'\bar{\boldsymbol{Q}}\mathcal{S}^{\text{u}} \tag{2.20}$$

$$L = \boldsymbol{x}'(0)\boldsymbol{Y}\boldsymbol{x}(0) - \mathcal{W}'\bar{\boldsymbol{Q}}\mathcal{W} - 2\boldsymbol{x}(0)\mathcal{S}^{\text{u}\prime}\bar{\boldsymbol{Q}}\mathcal{W} \tag{2.21}$$

Hence J_0 is a quadratic positive definite function of $\boldsymbol{U}_0$ and the optimal control input can be found by deriving with respect to $\boldsymbol{U}_0$ and proceeding as in 2.11.

2.1.2 Introduction to Dynamic Programming

Besides the Batch approach there exist several other mathematical concepts to calculate an optimal input minimizing an objective function. In this section Bellman's Dynamic Programming (DP) techniques will be presented briefly and the application to linear systems is demonstrated in a simulation. Although Dynamic Programming will not be used for nonlinear control in chapter 2 it shall be explained and evaluated in this section, since it helps to better understand the fundamental principles of optimal control.

Bellman's Principle of Optimality:

> For a trajectory $\boldsymbol{x}_0, \boldsymbol{x}_1^*, ..., \boldsymbol{x}_N^*$ to be optimal, the trajectory starting from an intermediate point $\boldsymbol{x}_j^*, \boldsymbol{x}_{j+1}^*, ..., \boldsymbol{x}_N^*$ must also be optimal.

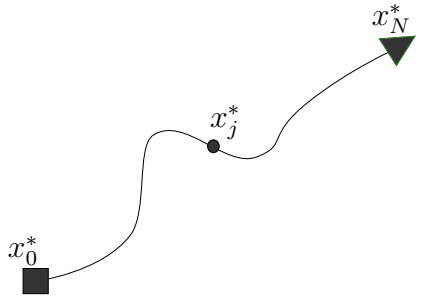

Figure 2.1: Bellman's Principle of Optimality

According to Borrelli [6, p. 152] the DP-scheme works as follows: The optimal cost-to-go function, in literature also referred to as the value-function, is defined as the lowest possible cost to go from an intermediate point $\boldsymbol{x}_j$ to the final point $\boldsymbol{x}_n$.

$$J_{j\to N}^*(\boldsymbol{x}_j) = \min_{\boldsymbol{u}_j,...,\boldsymbol{u}_{N-1}} p(\boldsymbol{x}_N) + \sum_k^{N-1} q(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.22}$$

$$\text{subj. to} \quad \boldsymbol{x}_{k+1} = g(\boldsymbol{x}_k, \boldsymbol{u}_k),\ k = j, ..., N-1 \tag{2.23}$$

$$h(\boldsymbol{x}_k, \boldsymbol{u}_k) \leq 0 \tag{2.24}$$

$$\boldsymbol{x}_N \in \mathcal{X}_f \tag{2.25}$$

Here $g(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is any dynamic system equation, $h(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is a vector of inequality constraints and $\mathcal{X}_f$ denotes the final region. The optimal cost-to-go $J_{j\to N}^*(\boldsymbol{x}_j)$ only depends on the state $\boldsymbol{x}_j$. In Equation 2.26 Bellman's Principle of Optimality is used to express the cost-to-go from instant $j-1$, $J_{j-1\to N}^*$ as a sum of the minimal stage cost $q(\boldsymbol{x}_{j-1}, \boldsymbol{u}_{j-1})$ and the minimal cost-to-go from instant j, $J_{j\to N}^*$:

$$J_{j-1\to N}^*(\boldsymbol{x}_{j-1}) = \min_{\boldsymbol{u}_{j-1}} q(\boldsymbol{x}_{j-1}, \boldsymbol{u}_{j-1}) + J_{j\to N}^*(\boldsymbol{x}_j) \tag{2.26}$$

$$\text{subj. to} \quad \boldsymbol{x}_{k+1} = g(\boldsymbol{x}_k, \boldsymbol{u}_k),\ k = j, ..., N-1 \tag{2.27}$$

$$h(\boldsymbol{x}_{j-1}, \boldsymbol{u}_{j-1}) \leq 0 \tag{2.28}$$

$$\boldsymbol{x}_j \in \mathcal{X}_{j\to N} \tag{2.29}$$

Here $\mathcal{X}_{j\to N}$ is the set of $\boldsymbol{x}_j$ that can be steered into $\boldsymbol{x}_N$.

Using 2.26 the original optimization problem over N time-steps can be converted into a series of N optimizations over one time-step: Starting with the final state cost,

$$J^*_{N\to N}(\boldsymbol{x}_N) = p(\boldsymbol{x}_N) \tag{2.30}$$
$$\mathcal{X}_{N\to N} = \mathcal{X}_f \tag{2.31}$$

the algorithm proceeds backwards in time:

$$\begin{aligned} J^*_{N-1\to N}(\boldsymbol{x}_{N-1}) = \quad & \min_{\boldsymbol{u}_{N-1}} \; q(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) + J^*_{N\to N}(g(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1})) \\ \text{subj. to} \quad & h(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) \le 0 \\ & g(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) \in \mathcal{X}_{N\to N} \end{aligned} \tag{2.32}$$

$$\vdots$$

$$\begin{aligned} J^*_{0\to N}(\boldsymbol{x}_0) = \quad & \min_{\boldsymbol{u}_0} \; q(\boldsymbol{x}_0, \boldsymbol{u}_0) + J^*_{1\to N}(g(\boldsymbol{x}_0, \boldsymbol{u}_0)) \\ \text{subj. to} \quad & h(\boldsymbol{x}_0, \boldsymbol{u}_0) \le 0 \\ & g(\boldsymbol{x}_0, \boldsymbol{u}_0) \in \mathcal{X}_{1\to N}, \;\; \boldsymbol{x}_0 = \boldsymbol{x}(0) \end{aligned} \tag{2.33}$$

It is important to note that in case of nonlinear models or nonconvex cost-functions it is difficult to find analytic forms for $J^*_{j\to N}$ and therefore the value-function has to be approximated at grid points in the state space of $\boldsymbol{x}_j$ for each step $1...N$. For higher dimensional system this can lead to great computational complexity.

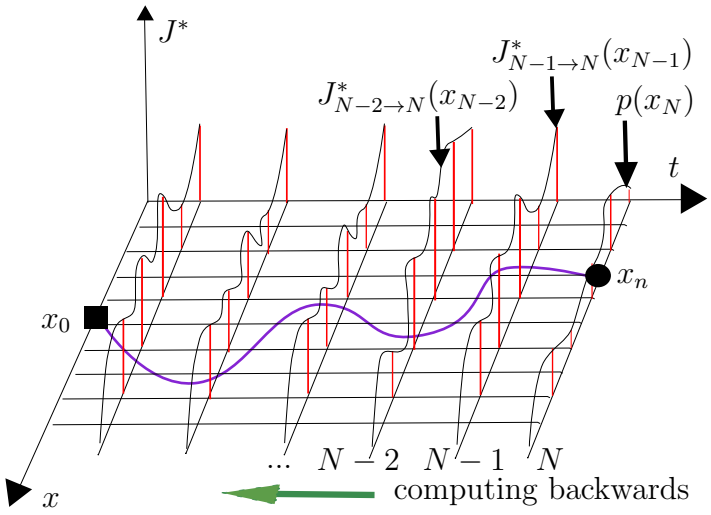

Figure 2.2: Dynamic Programming for the case of a single statevariable x

2.1.3 Recursive Approach for Linear Systems

In this section the Dynamic Programming algorithm is applied on the control of a linear system as explained in [6, p. 167]. The optimal cost-to-go or value function

starting from state $\boldsymbol{x}_j$ is:

$$J^*_{j\to N}(\boldsymbol{x}_j) = \min_{\boldsymbol{u}_j,\ldots,\boldsymbol{u}_{N-1}} \boldsymbol{x}'_N\boldsymbol{P}\boldsymbol{x}_N + \sum_{k=j}^{N-1} \boldsymbol{x}'_k\boldsymbol{Q}\boldsymbol{x}_k + \boldsymbol{u}'_k\boldsymbol{R}\boldsymbol{u}_k \tag{2.34}$$

Applying the principle of optimality, the optimization can be broken down in series of one-step optimizations. The optimal cost-to-go from $\boldsymbol{x}_{N-1}$ to $\boldsymbol{x}_N$ can be written as:

$$J^*_{N-1\to N}(\boldsymbol{x}_{N-1}) = \min_{\boldsymbol{u}_{N-1}} \boldsymbol{x}'_N\boldsymbol{P}_N\boldsymbol{x}_N \;+\; \boldsymbol{x}'_{N-1}\boldsymbol{Q}\boldsymbol{x}_{N-1} \;+\; \boldsymbol{u}'_{N-1}\boldsymbol{P}\boldsymbol{u}_{N-1} \tag{2.35}$$

$$\boldsymbol{P}_N \;=\; \boldsymbol{P} \tag{2.36}$$

After eliminating $\boldsymbol{x}_N$ with $\boldsymbol{x}_N \;=\; \boldsymbol{A}\boldsymbol{x}_{N-1} + \boldsymbol{B}\boldsymbol{u}_{N-1}$ in 2.35 the optimal one step cost-to-go depends on $\boldsymbol{x}_{N-1}$ and $\boldsymbol{u}_{N-1}$:

$$J^*_{N-1\to N}(\boldsymbol{x}_{N-1}) = \min_{\boldsymbol{u}_{N-1}}\{\boldsymbol{x}'_{N-1}(\boldsymbol{A}'\boldsymbol{P}_N\boldsymbol{A} + \boldsymbol{Q})\boldsymbol{x}_{N-1} \tag{2.37}$$

$$+ 2\boldsymbol{x}_{N-1}(\boldsymbol{A}'\boldsymbol{P}_N\boldsymbol{B})\boldsymbol{u}_{N-1} \tag{2.38}$$

$$+ \boldsymbol{u}'_{N-1}(\boldsymbol{B}'\boldsymbol{P}_N\boldsymbol{B} + \boldsymbol{R})\boldsymbol{P}\boldsymbol{u}_{N-1}\} \tag{2.39}$$

By deriving with respect to $\boldsymbol{u}_{N-1}$ the optimal input for time-step $N-1$ is obtained:

$$\boldsymbol{u}^*_{N-1} = -(\boldsymbol{B}'\boldsymbol{P}_N\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}'\boldsymbol{P}_N\boldsymbol{A}\boldsymbol{x}_{N-1} \tag{2.40}$$

This expression can be inserted in 2.37, thus the optimal one step cost-to-go $J^*_{N-1\to N}(\boldsymbol{x}_{N-1})$ now only depends on $\boldsymbol{x}_{N-1}$ and no more on $\boldsymbol{u}_{N-1}$:

$$J^*_{N-1\to N}(\boldsymbol{x}_{N-1}) = \boldsymbol{x}'_{N-1}\boldsymbol{P}_{N-1}\boldsymbol{x}_{N-1} \tag{2.41}$$

$$\text{with: } \boldsymbol{P}_{N-1} = \boldsymbol{A}'\boldsymbol{P}_N\boldsymbol{A} + \boldsymbol{Q} - \boldsymbol{A}'\boldsymbol{P}_N\boldsymbol{B}(\boldsymbol{B}'\boldsymbol{P}_N\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}'\boldsymbol{P}_N\boldsymbol{A} \tag{2.42}$$

Going further backwards in time for one step, the optimal cost-to-go from $\boldsymbol{x}_{N-2}$ to $\boldsymbol{x}_N$ can be expressed using 2.42:

$$J^*_{N-2\to N}(\boldsymbol{x}_{N-2}) = \min_{\boldsymbol{u}_{N-2}} \boldsymbol{x}'_{N-1}\boldsymbol{P}_{N-1}\boldsymbol{x}_{N-1} \;+\; \boldsymbol{x}'_{N-2}\boldsymbol{Q}\boldsymbol{x}_{N-2} \;+\; \boldsymbol{u}'_{N-2}\boldsymbol{R}\boldsymbol{u}_{N-2} \tag{2.43}$$

For the optimal input at instant $N-2$ we obtain a similar result as 2.40:

$$\boldsymbol{u}^*_{N-2} = -(\boldsymbol{B}'\boldsymbol{P}_{N-1}\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}'\boldsymbol{P}_{N-1}\boldsymbol{A}\boldsymbol{x}_{N-2} \tag{2.44}$$

Again inserting it into 2.43 yields an expression for $J^*_{N-2\to N}(\boldsymbol{x}_{N-2})$ that only depends on $\boldsymbol{x}_{N-2}$:

$$J^*_{N-2\to N}(\boldsymbol{x}_{N-2}) = \boldsymbol{x}'_{N-2}\boldsymbol{P}_{N-2}\boldsymbol{x}_{N-2} \tag{2.45}$$

$$\text{with: } \boldsymbol{P}_{N-2} = \boldsymbol{A}'\boldsymbol{P}_{N-1}\boldsymbol{A} + \boldsymbol{Q} - \boldsymbol{A}'\boldsymbol{P}_{N-1}\boldsymbol{B}(\boldsymbol{B}'\boldsymbol{P}_{N-1}\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}'\boldsymbol{P}_{N-1}\boldsymbol{A} \tag{2.46}$$

Note that $\boldsymbol{P}_{N-2}$ uses $\boldsymbol{P}_{N-1}$, thus a matrix recursion has been established. By continuing in this manner the optimal control input at some arbitrary instant k can be described as:

$$\boldsymbol{u}^*(k) = -(\boldsymbol{B}'\boldsymbol{P}_{k+1}\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}'\boldsymbol{P}_{k+1}\boldsymbol{A}\boldsymbol{x}(k) \quad \text{for } k = 0, ..., N-1 \tag{2.47}$$

$$\boldsymbol{P}_k = \boldsymbol{A}'\boldsymbol{P}_{k+1}\boldsymbol{A} + \boldsymbol{Q} - \boldsymbol{A}'\boldsymbol{P}_{k+1}\boldsymbol{B}(\boldsymbol{B}'\boldsymbol{P}_{k+1}\boldsymbol{B} + \boldsymbol{R})^{-1}\boldsymbol{B}'\boldsymbol{P}_{k+1}\boldsymbol{A} \tag{2.48}$$

According to equation 2.47 the optimal input at time-step k can be understood as a state-feedback of state $\boldsymbol{x}(k)$. Here the regulation problem was solved, it is also possible to handle the regulation problem but not addressed here.

2.1.4 Simulation with Batch Approach and Dynamic Programming

For testing the two algorithms presented before two controllable, one dynamically stable and one dynamically unstable system are defined as follows:

3. Order Stable System:

$$G(s) = \frac{(s+1)(s+2)}{(s+1-j)(s+1+j)(s+3)} \tag{2.49}$$

The transfer function above is converted to a discrete time state-space representation with a discretization time of $\Delta t = 0.1s$.

$$\boldsymbol{x}_{k+1} = \begin{pmatrix} 2.5415 & -1.0763 & 0.6065 \\ 2.0000 & 0 & 0 \\ 0 & 0.5000 & 0 \end{pmatrix} \boldsymbol{x}_k + \begin{pmatrix} 0.5000 \\ 0 \\ 0 \end{pmatrix} \boldsymbol{u}_k; \quad \boldsymbol{x}(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.50}$$

$$\boldsymbol{y}_k = \begin{pmatrix} 0.1813 & -0.1562 & 0.1343 \end{pmatrix}' \boldsymbol{x}_k \tag{2.51}$$

3. Order Unstable System:

$$G(s) = \frac{(s+1)(s+2)}{(s+1-j)(s+1+j)(s-3)}; \quad \Delta t = 0.1s \tag{2.52}$$

$$\boldsymbol{x}_{k+1} = \begin{pmatrix} 3.1505 & -1.6247 & 0.5526 \\ 2.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \end{pmatrix} \boldsymbol{x}_k + \begin{pmatrix} 0.5000 \\ 0 \\ 0 \end{pmatrix} \boldsymbol{u}_k; \quad \boldsymbol{x}(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.53}$$

$$\boldsymbol{y}_k = \begin{pmatrix} 0.2436 & -0.2098 & 0.09012 \end{pmatrix}' \boldsymbol{x}_k \tag{2.54}$$

For the cost function variables $\boldsymbol{Q}, \boldsymbol{P}$ and R the following values are chosen:

$$\boldsymbol{Q} = \begin{pmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{pmatrix}; \quad \boldsymbol{P} = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{pmatrix}; \quad R = 500 \tag{2.55}$$

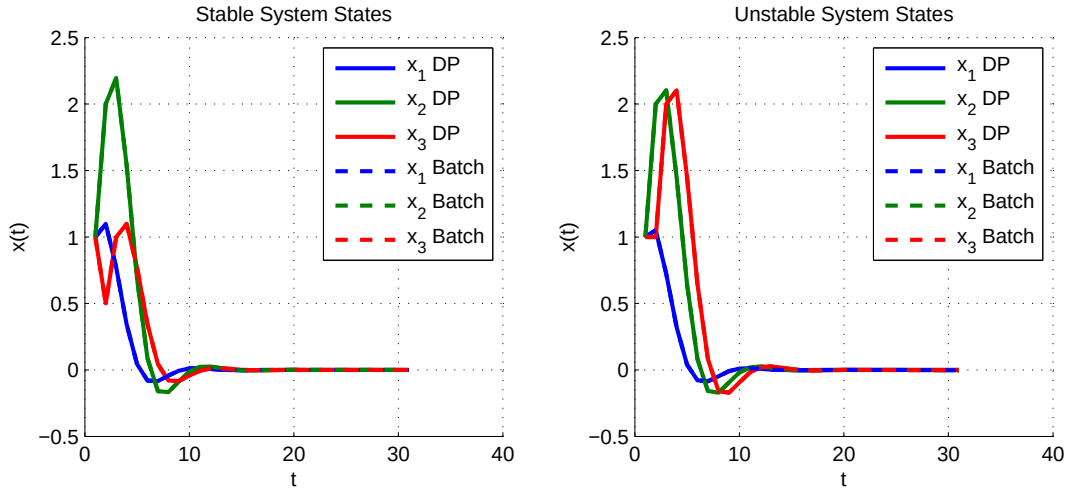

Figure 2.3: Evolution of the states over time for stable system and unstable system

Figure 2.3 shows the evolution of the states over time for the stable system (left) and the unstable system (right). In this simulation the regulation problem is solved i.e. the state is driven to the origin with minimum cost. The state trajectories for both algorithm match each other showing the equivalence of the two approaches. The same behavior can be observed for the inputs:

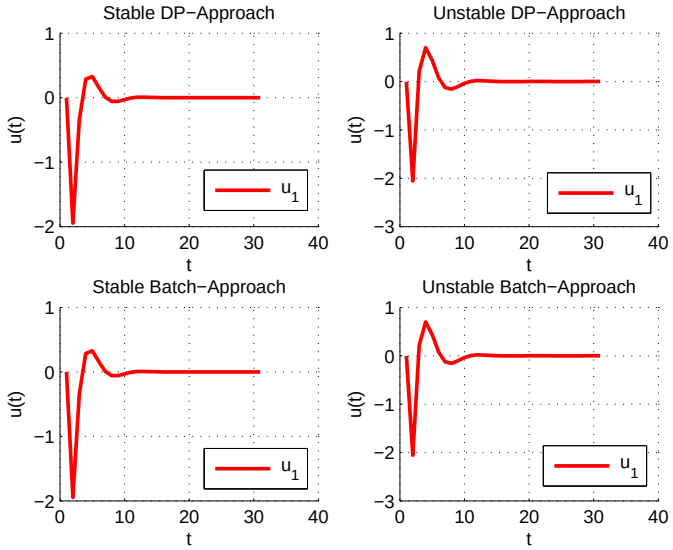

Figure 2.4: Control Inputs calculated by Batch- and DP-Approach, for stable and unstable System

While giving exactly the same behavior, the ideas of batch approach and DP-approach are fundamentally different. For the batch approach the sequence of optimal inputs is calculated once and then applied to the system in a feed-forward style. By contrast the DP approach uses a state-feedback (see 2.47) that is given by the matrix recursion 2.48. Using DP in a control loop is more efficient than using the Batch Approach since the latter would recalculate the whole sequence of optimal

inputs at every step. This means that the inversion of the possibly large matrix in 2.13 would have be to solved over and over again. According to [6, p. 167] a remedy to this problem is the partial inversion of 2.13 which will not be addressed here.

2.2 Model-Predictive Control

2.2.1 The Idea of Model-Predictive Control

The term model predictive control does not describe a specific control algorithm it rather characterizes control strategies that have the following features in common:

- The control is based on a model of the system
- The optimal input is determined in a way to minimize an objective function within a moving horizon
- The optimal input trajectory is recalculated in every cycle. Only the first sample of input trajectory is applied to the plant.

Before explaining the idea of model-predictive control the concept of finite horizon control shall be introduced.

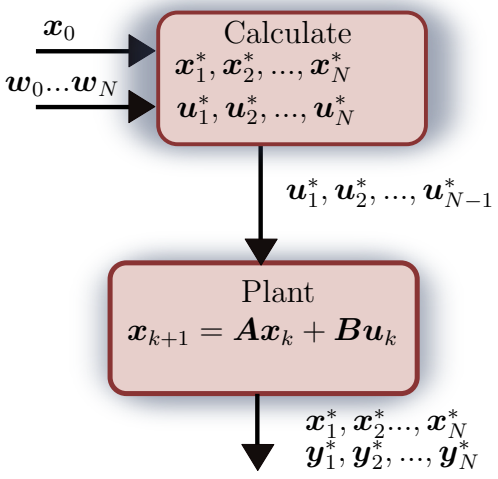

Figure 2.5: Finite Horizon Control

The concept of finite horizon control as depicted in figure 2.5 is to perform an optimization from instant $1...N$ using the initial state $\boldsymbol{x}_0$ and the reference trajectory $\boldsymbol{w}_0, ..., \boldsymbol{w}_N$. The optimal input sequence $\boldsymbol{u}_1^*, ..., \boldsymbol{u}_{N-1}^*$ is then applied to the plant in a feed-forward manner. However in practice finite horizon control requires large computational power for long task durations, especially when using fine prediction intervals.

Model predictive control is avoiding this problem by using a relatively short horizon that is constantly shifting with time. As shown in figure 2.6 the optimal input sequence $\boldsymbol{u}_k^*, \boldsymbol{u}_{k+1}^*, ..., \boldsymbol{u}_{k+N}^*$ and optimal state trajectory $\boldsymbol{x}_k^*, \boldsymbol{x}_{k+1}^*, ..., \boldsymbol{x}_{k+N}^*$ is computed in every cycle and only the first optimal input $\boldsymbol{u}_k^*$ is sent to plant. Model predictive control also establishes a state feedback since it uses the current state $\boldsymbol{x}_k$ for the starting point of each recurring optimization. By contrast to finite-horizon control where the horizon is as long as the task itself, for MPC much shorter horizons can be used while still achieving satisfactory results. However the trajectory of MPC is different from the global optimum, although with increasing horizon length MPC's behavior gets more and more similar to the result of finite horizon control.

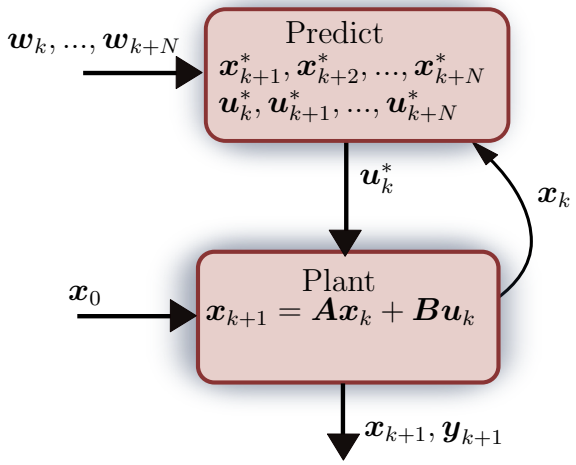

Figure 2.6: Model-Predictive Control

Model predictive control has become very popular in the industry due to the following reasons:

- Mulitvariable Control possible
- Constraints to state and control input can be implemented in a simple manner
- The control is guided by the future reference trajectory and not only by the current reference, the same holds for the constraints
- If the system has dead times, they can be eliminated

However MPC's major drawbacks are:

- It its difficult to formally proof stability in case of complex systems
- The computational burden rapidly increases with longer horizons or by adding constraints
- If the cost is non-quadratic or the system is nonlinear the optimal control law cannot by calculated analytically, thus numerical techniques have to be used

2.2.2 Generalized Predictive Control

Generalized predictive control (GPC) is an algorithm that can be used for linear systems and has been commercially successful in the process industry for applications like the control of distillation columns. In this section GPC shall be implemented in order to understand its characteristic behavior. The following introduction to GPC is based on Camacho [7].

GPC uses a discrete time model called controlled autoregressive integrated moving average model (CARIMA). For SISO-systems this is:

$$A(z^{-1})y(t) = z^{-d}B(z^{-1})u(t-1) + C(z^{-1})\frac{e(t)}{\triangle} \tag{2.56}$$

with delay d and $\triangle = 1 - z^{-1}$

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + ... + a_{na} z^{-na} \tag{2.57}$$
$$B(z^{-1}) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + ... + b_{nb} z^{-nb} \tag{2.58}$$
$$C(z^{-1}) = 1 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3} + ... + c_{nc} z^{-nc} \tag{2.59}$$

The model with output $y(t)$ and input $u(t)$ is disturbed by the white noise $e(t)$. The model is defined by A, B, C which are polynomials in z^{-1}. The polynomial C can be chosen to color the white noise, however for simplicity it is assumed to equal 1. The cost J in GPC is given by the following equation:

$$J = \sum_{j=0}^{N} \left\{ [\hat{y}(t+j) - w(t+j)]^2 + \triangle u(t+j-1)^2 \right\} \tag{2.60}$$

The reference trajectory is given by $w(t)$ for a horizon length N, the control input increment is denoted by $\triangle u(t)$. $\hat{y}(t+j|t)$ are the predicted outputs for instant $t+j$ when applying the computed optimal input.
The goal of GPC is to determine the sequence of future inputs $u(t), ..., u(t+N-1)$ in order to minimize the cost function 2.60. To achieve this $\hat{y}(t), ..., \hat{y}(t+N)$ has to be expressed as a function of the future inputs explicitly. This is accomplished by splitting up y into a free response $y_{free}(t)$ that only depends on the past inputs $u_{free}(t)$ and a forced response $y_{forced}(t)$ that only depends on the future input changes $u_{forced}(t)$ (see figure 2.7). $y_{free}(t)$ and $y_{forced}(t)$ are obtained by splitting up u into a signal $u_{free}(t)$ that equals $u(t)$ until t remaining constant afterwards and another signal, $u_{forced}(t))$ that is zero until t and equals the future inputs increments after t.

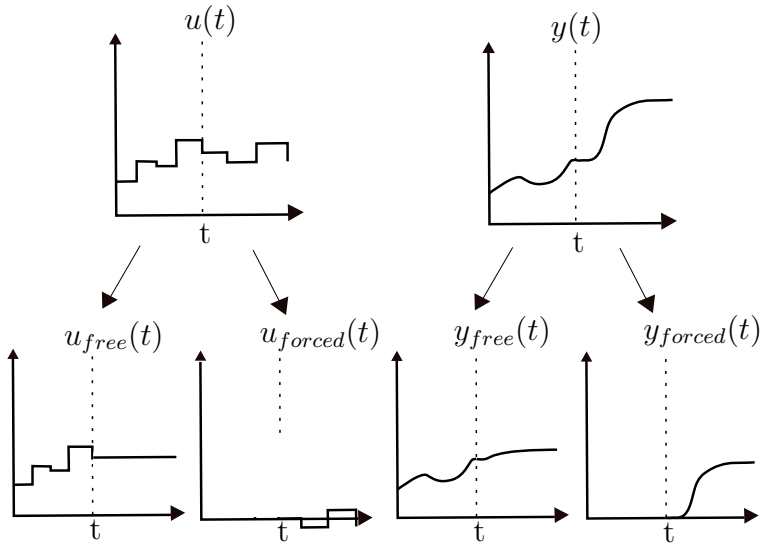

Figure 2.7: Dividing Control Input to obtain Free and Forced Response

In order to extract the forced and the free response from the model a diophantine equation is introduced:

$$1 = E_j(z^{-1}) \underbrace{\triangle A(z^{-1})}_{\tilde{A}(z^{-1})} + z^{-j} F_j(z^{-1}) \tag{2.61}$$

Here j is an index running from 1 to N, indicating future time steps and $\triangle = 1 - z^{-1}$ is an integrator. Both $E_j(z^{-1})$ and $F_j(z^{-1})$ are polynomials with orders of $j-1$ and na respectively. $E_j(z^{-1})$ is obtained by dividing 1 by $\tilde{A}$ until the remainder can be factorized as $z^{-1}F_j(z^{-1})$. For the simulation a first-order model is chosen where $A(z^{-1})$ is $1 - 0.8z^{-1}$. Then $\tilde{A}$ is:

$$\triangle\, A(z^{-1}) = \tilde{A}(z^{-1}) = 1 - 1.8z^{-1} + 0.8z^{-2} \tag{2.62}$$

Next $1/\tilde{A}$ is calculated in a polynomial long division:

$$\begin{array}{rlll}
 & 1 & & \quad : (1 - 1.8z^{-1} + 0.8z^{-2}) \\
- & 1 \quad -1.8z^{-1} & +0.8z^{-2} & \quad = 1 + 1.8z^{-1} + 2.44z^{-2} \\
\hline
 & \quad 1.8z^{-1} & -0.8z^{-2} & \\
- & \quad 1.8z^{-1} & -3.24z^{-2} & +1.44z^{-3} \\
\hline
 & & \quad 2.44z^{-2} & -1.44z^{-3} \\
- & & \quad 2.44z^{-2} & +4.392z^{-3} \quad -1.952z^{-4} \\
\hline
 & & & \quad 2.952z^{-3} \quad -1.952z^{-4}
\end{array}$$

From this polynomial long division $E_j(z^{-1})$ is obtained by taking the quotient after the j-th division and $F_j(z^{-1})$ is the remainder of the j-th division.

$$\begin{array}{l|l}
E_1 = 1 & F_1 = 1.8 - 0.8z^{-1} \\
E_2 = 1 + 1.8z^{-1} & F_2 = 2.44z^{-2} - 1.44z^{-3} \\
E_3 = 1 + 1.8z^{-1} + 2.44z^{-2} & F_3 = 2.952z^{-3} - 1.952z^{-4}
\end{array}$$

By multiplying the model equation 2.56 with $\triangle E_j(z^{-1})$ it can be written as:

$$\tilde{A}(z^{-1})E_j(z^{-1})y(t+j) = E_j(z^{-1})B(z^{-1}) \triangle u(t+j-d-1) + E_j(z^{-1})e(t+j) \quad (2.63)$$

Using equation 2.61 to replace $\tilde{A}(z^{-1})E_j(z^{-1})$ in the equation above, this can be written as:

$$(1-z^{-j}F_j(z^{-1}))y(t+j) = E_j(z^{-1})B(z^{-1}) \triangle u(t+j-d-1) + E_j(z^{-1})e(t+j) \quad (2.64)$$

which can be rewritten as:

$$y(t+j) = F_j(z^{-1})y(t) + E_j(z^{-1})B(z^{-1}) \triangle u(t+j-d-1) + E_j(z^{-1})e(t+j) \quad (2.65)$$

The last term $E_j(z^{-1})e(t+j)$ in the equation above is referring to unknown noise values in the future since the order of E_j is $j-1$. The prediction $\hat{y}(t+j|t)$ will be made assuming that the mean of the noise is zero and will therefore omit this term:

$$\hat{y}(t+j) = F_j(z^{-1})y(t) + \underbrace{E_j(z^{-1})B(z^{-1})}_{G_j(z^{-1})}(z^{-1}) \triangle u(t+j-d-1) + \underbrace{E_j(z^{-1})e(t+j)}_{=0} \quad (2.66)$$

Using equation 2.66 the j-step prediction can be written as:

$$\hat{y}(t+d+1) = G_{d+1} \triangle u(t) + F_{d+1}y(t) \quad (2.67)$$
$$\hat{y}(t+d+2) = G_{d+2} \triangle u(t+1) + F_{d+2}y(t) \quad (2.68)$$
$$\vdots$$
$$\hat{y}(t+d+N) = G_{d+N} \triangle u(t+N-1) + F_{d+N}y(t) \quad (2.69)$$

In vector-matrix form this can be written as:

$$\mathbf{y} = \mathbf{G}\mathbf{u} + \mathbf{F}(z^{-1}) + \mathbf{G}'(z^{-1}) \triangle u(t-1) \quad (2.70)$$

where:

$$\mathbf{y} = \begin{pmatrix} \hat{y}(t+d+1) \\ \hat{y}(t+d+2) \\ \vdots \\ \hat{y}(t+d+N) \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} \triangle u(t) \\ \triangle u(t+1) \\ \vdots \\ \triangle u(t+N-1) \end{pmatrix} \quad (2.71)$$

$$\mathbf{G}'(z^{-1}) = \begin{pmatrix} (G_{d+1}(z^{-1}) - g_0)z \\ (G_{d+2}(z^{-1}) - g_0 - g_1 z^{-1})z^2 \\ \vdots \\ (G_{d+N}(z^{-1}) - g_0 - g_1 z^{-1} - ... - g_{N-1}z^{-(N-1)})z^N \end{pmatrix} \quad (2.72)$$

$$\mathbf{G} = \begin{pmatrix} g_0 & 0 & \dots & 0 \\ g_1 & g_0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ g_{N-1} & g_{N-2} & \dots & g_0 \end{pmatrix}, \quad \mathbf{F}(z^{-1}) = \begin{pmatrix} F_{d+1}(z^{-1}) \\ F_{d+2}(z^{-1}) \\ \vdots \\ F_{d+N}(z^{-1}) \end{pmatrix} \quad (2.73)$$

With equation 2.70 the prediction $\mathbf{y}$ has been separated into a first part $\mathbf{Gu}$ that only depends on future inputs increments $\mathbf{u}$ and thus can be identified as the forced response. The last two terms that only depend on the past give the free response $\boldsymbol{f}$.

$$\mathbf{y} = \mathbf{Gu} + \mathbf{f} \tag{2.74}$$

The free-forced response description is inserted into the cost-function 2.60, yielding and an expression where J explicitly depends on the future control input increments u.

$$J = (\mathbf{Gu} + \mathbf{f} - \mathbf{w})^{\mathbf{T}}(\mathbf{Gu} + \mathbf{f} - \mathbf{w}) + \lambda \mathbf{u}^{\mathbf{T}}\mathbf{u} \tag{2.75}$$

Exactly in the same way as in the batch approach the cost function is derived with respect to the vector of inputs $\mathbf{u}$:

$$\frac{\partial}{\partial \mathbf{u}} J = 0 \tag{2.76}$$

consequently we obtain the sequence of optimal input increments:

$$\mathbf{u} = (\mathbf{G}^{\mathbf{T}}\mathbf{G} + \lambda \mathbf{I})^{-\mathbf{1}}\mathbf{G}^{\mathbf{T}}(\mathbf{w} - \mathbf{f}) \tag{2.77}$$

Only the first input of this sequence is sent to the plant therefore the control law can be written as:

$$\triangle u(t) = \mathbf{K}(\mathbf{w} - \mathbf{f}) \tag{2.78}$$

With $\mathbf{K}$ being the the first row vector of $\mathbf{u} = (\mathbf{G}^{\mathbf{T}}\mathbf{G} + \lambda \mathbf{I})^{-\mathbf{1}}\mathbf{G}^{\mathbf{T}}(\mathbf{w} - \mathbf{f})$.

Simulation Results A simulation of the GPC algorithm was carried out with a model using the following parameters:

$$A(z^{-1}) = 1 - 0.8z^{-1} \tag{2.79}$$
$$B(z^{-1}) = 0.4 + 0.6z^{-1} \tag{2.80}$$
$$C(z^{-1}) = 1 \tag{2.81}$$

Figure 2.8 shows the system output y when performing tracking control with different horizon lengths N. Using short horizons a larger overshoot is observed than for long horizons. Short horizons cause higher costs and are less optimal. It turned out, that the cost integrated over the whole time $0...t_{end}$ quickly converges with increasing horizon lengths (see figure 2.9). It appears that with a horizon of 4 to 5 steps the cost does not decrease further, being already very close to the optimal. Besides the number of prediction steps the time-interval between the predictions also has great influence on the control's behavior. A more detailed discussion about the importance of the prediction intervals and the prediction horizon will be given in the end of the next subsection.

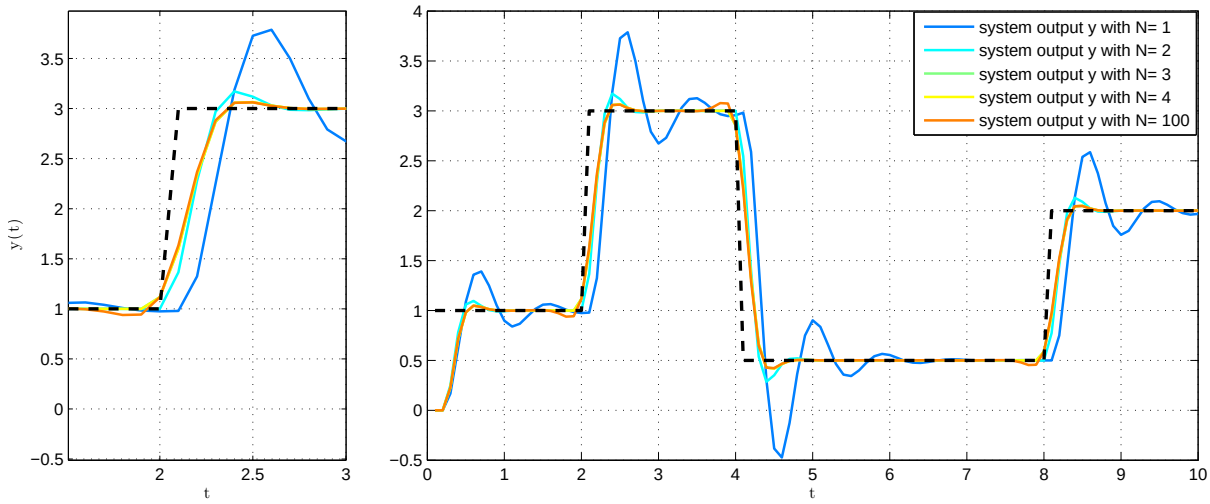

Figure 2.8: System Output y over time when using GPC with different horizons N

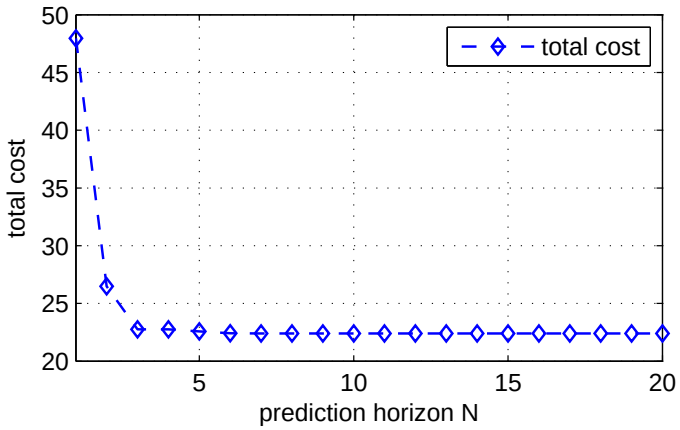

Figure 2.9: Total cost of control with different horizons; The optimum is already reached with short horizons

2.2.3 Constrained Model Predictive Control using Batch Approach

In real world control applications both the state and an the control input are bounded by physical limitations. It is possible to incorporate equality and inequality constraints into the optimization process performed in MPC. However using constraints the optimal input cannot be derived analytically anymore and numerical techniques have to be used. In case of a linear model and a quadratic cost the optimal control problem takes the shape of a quadratic program thus a standard QP-solver can be used. The constrained MPC introduced here is based on the batch approach for the tracking problem from section 2.1.1.

In the context of optimal control problem 2.1 constraints on the state, the final state and the input are introduced as follows:

$$\begin{aligned} \boldsymbol{A}_x \boldsymbol{x}_i &\leq \boldsymbol{b}_x \quad \text{for all } i = 1, ..., N-1 \\ \boldsymbol{A}_f \boldsymbol{x}_N &\leq \boldsymbol{b}_f \\ \boldsymbol{A}_u \boldsymbol{u}_i &\leq \boldsymbol{b}_u \quad \text{for all } i = 1, ..., N \end{aligned} \tag{2.82}$$

For example constraining each state-variable of $\boldsymbol{x}$: $-10 \leq x_i \leq 10$ can be written as:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{A_x} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x_i} \leq \underbrace{\begin{bmatrix} 10 \\ 10 \\ 10 \\ -10 \\ -10 \\ -10 \end{bmatrix}}_{b_x}$$

All inequality constraints are arranged into a batch format using the following vectors and matrices:

$$\boldsymbol{G}\boldsymbol{U}_0 - \boldsymbol{E}\boldsymbol{x}_0 \leq \boldsymbol{w} \tag{2.83}$$

where $\boldsymbol{U}_0$ is the sequence of control inputs, defined in 2.1.1 and $\boldsymbol{x}_0$ is the initial state.

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{A}_u & 0 & \dots & 0 \\ 0 & \boldsymbol{A}_u & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & A_u \\ 0 & 0 & \dots & 0 \\ \boldsymbol{A}_x\boldsymbol{B} & 0 & \dots & 0 \\ \boldsymbol{A}_x\boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}_x\boldsymbol{B} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{A}_f\boldsymbol{A}^{N-1}\boldsymbol{B} & \boldsymbol{A}_f\boldsymbol{A}^{N-2}\boldsymbol{B} & \dots & \boldsymbol{A}_f\boldsymbol{B} \end{bmatrix}, \boldsymbol{E} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -\boldsymbol{A}_x \\ -\boldsymbol{A}_x\boldsymbol{A} \\ -\boldsymbol{A}_x\boldsymbol{A}^2 \\ \vdots \\ -\boldsymbol{A}_f\boldsymbol{A}^N \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} \boldsymbol{b}_u \\ \boldsymbol{b}_u \\ \vdots \\ \boldsymbol{b}_u \\ \boldsymbol{b}_x \\ \boldsymbol{b}_x \\ \boldsymbol{b}_x \\ \vdots \\ \boldsymbol{b}_f \end{bmatrix}$$

In section 2.1.1,page 9 the cost for the tracking-controlled system was transferred into the following expression:

$$J_0 = \boldsymbol{U}_0'\boldsymbol{H}\boldsymbol{U}_0 + 2(\boldsymbol{x}'(0)\boldsymbol{F} - \boldsymbol{k})\boldsymbol{U}_0 + L \tag{2.84}$$

To compute the control law a standard QP-Solver, the Matlab™'s internal "quadprog"-function is used. This function requires the quadratic program to be of the form:

$$\min_{\boldsymbol{x}} \left\{ \frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Gamma}\boldsymbol{x} + \boldsymbol{f}^T\boldsymbol{x} \right\} \tag{2.85}$$

subject to:

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{2.86}$$

where $\boldsymbol{x}$ is a vector in $\mathbb{R}^n$, $\boldsymbol{\Gamma}$ is a positive definite, symmetric matrix, $\boldsymbol{f}^T$ is a vector in $\mathbb{R}^n$, $\boldsymbol{A}$ is matrix in $\mathbb{R}^{m\times n}$ and $\boldsymbol{b}$ is a vector in $\mathbb{R}^m$.
The following identifications can be made, where the left side is the notation of the QP-Solver and the right side the notation used here for the optimal control problem:

$$\boldsymbol{x} \mathrel{\widehat{=}} \boldsymbol{U}_0' \tag{2.87}$$

$$\boldsymbol{\Gamma} \mathrel{\widehat{=}} \frac{1}{2}\boldsymbol{H} \tag{2.88}$$

$$\boldsymbol{f} \mathrel{\widehat{=}} \boldsymbol{x}'(0)\boldsymbol{F} - \boldsymbol{k} \tag{2.89}$$

$$\boldsymbol{A} \mathrel{\widehat{=}} \boldsymbol{G} \tag{2.90}$$

$$\boldsymbol{b} \mathrel{\widehat{=}} \boldsymbol{w} + \boldsymbol{E}\boldsymbol{x}_0 \tag{2.91}$$

Simulation of Constrained MPC using the Batch Approach
A linear unstable continuous State Space Model is chosen with the following parameters:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -c/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} u \tag{2.92}$$

where $k = -2[N/m],\ c = 1[Ns/m],\ m = 1[kg]$. The state-space model is describing an unstable mass-spring-damper system with a negative spring-stiffness. Variable x_1 is the position and x_2 the velocity of the mass. The Model is discretized with time intervals of $0.05s$. The matrices of the cost function are chosen as follows:

$$\boldsymbol{P} = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \ \boldsymbol{Q} = \begin{bmatrix} 100 & 0 \\ 0 & 10 \end{bmatrix},\ R = 1/2 \tag{2.93}$$

The constraints are chosen to be:

$$-2 \leq x_1 \leq 2, \quad -40 \leq u \leq 40 \tag{2.94}$$

In a simulation MPC is tasked with controlling the mass to track a sinusoidal position profile, i.e. the position x_1 is required to follow the sinusoidal trajectory indicated in figure 2.10 with black dashed lines. At the same time position x_1 is chosen to be constrained by the interval [-2,2] and the input by the interval of [-40,40]. A time-step of $0.05s$ is used between the predictions and the same interval is used for simulation. Figure 2.10 and Figure 2.11 show the position- and velocity-trajectories generated by MPC's with different horizon lengths.

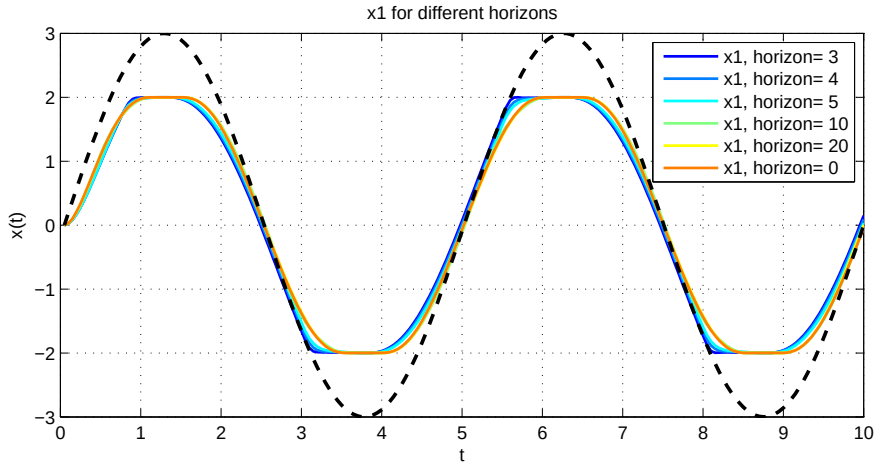

Figure 2.10: The position, state x_1 is required to follow the reference (dashed black line) while being constrained within $[-2, 2]$; Different horizons lengths are used, 0 indicates finite horizon control

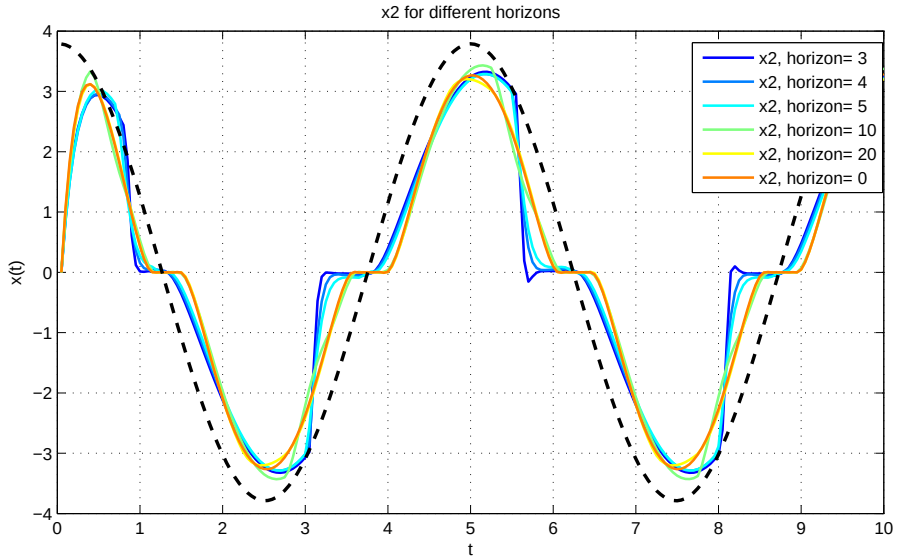

Figure 2.11: Evolution of state x_2, while state x_1 is constrained; different horizons lengths

Figure 2.10 shows that the position x_1 stops at the constraint although the reference trajectory line exceeds the constraint boundary. Therefore the controller must decelerate the mass when approaching the constraint.

2.2.4 Discussion of Results

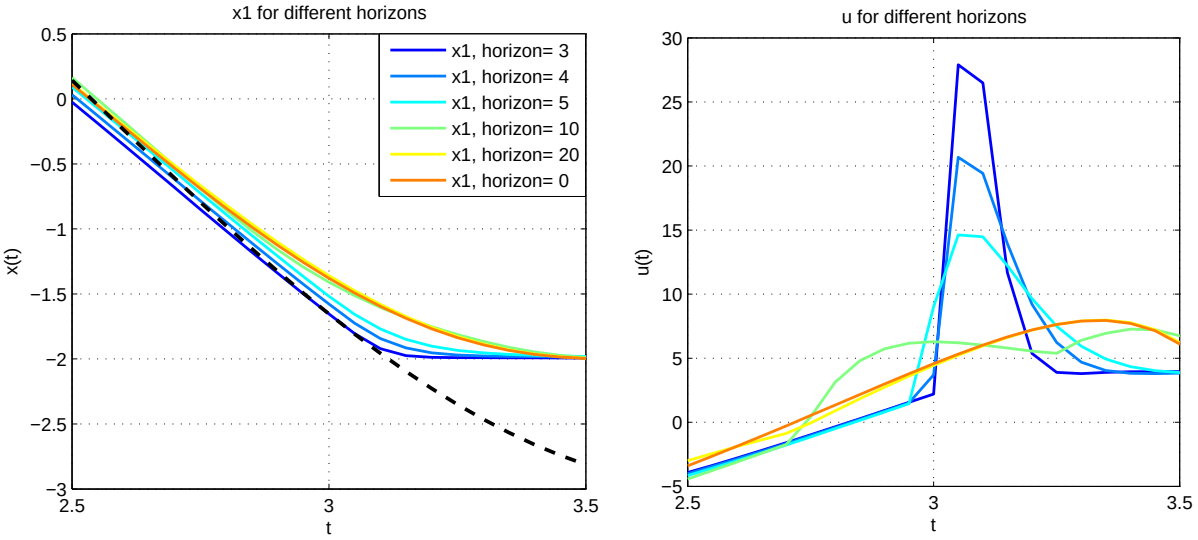

Figure 2.12: Left: Approaching the constraint at $x_1 = -2$ with different horizons, Right: Control input u with different horizons

By looking closely at the differences between horizons while approaching the constraint at $x_1 = -2$ (see figure 2.12), the following observation can be made: With shorter horizons (e.g. the dark blue line) x_1 is decelerating later in front of the constraint demanding a higher control spike in order not to break the constraint than with longer horizons. For example the finite horizon controller (the orange line) decelerates earlier thus requiring less control effort and enabling a smoother behavior. It can be concluded that longer horizon generate more foresighted behavior, whereas short horizons cause myopic behavior.

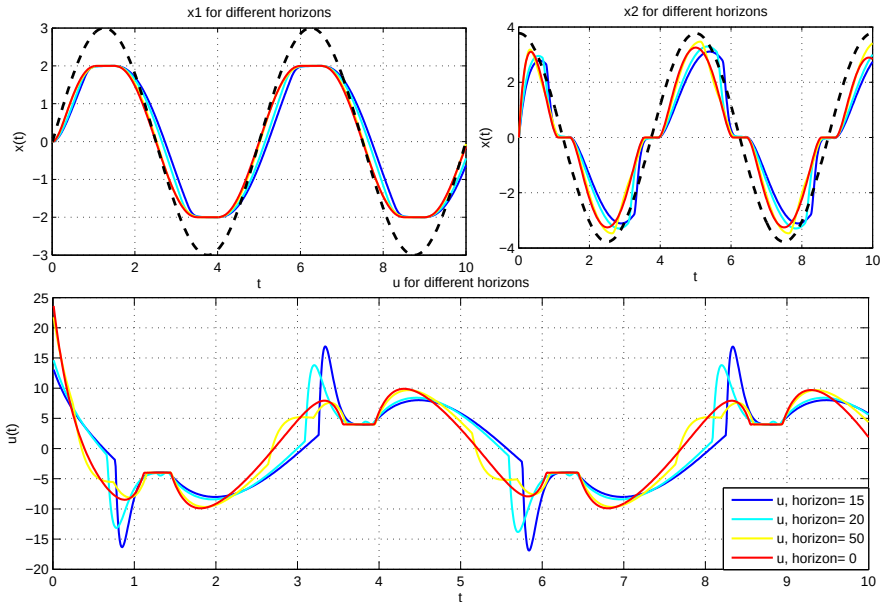

Figure 2.13: Smoother control behavior with smaller prediction and simulation steps

The influence of decreasing the prediction and simulation interval from $0.05s$ to $0.01s$ is shown in figure 2.13. When approaching the constraint the spikes in the control input (2.13 bottom figure) get smoother and smaller than in the previous simulation.
The key factors determining the behavior can be summarized as the number of prediction samples N, the prediction interval Δt_{pred} and the real-time horizon, $t_{real} = N \cdot \Delta t_{pred}$ (see figure 2.14). Longer real-time horizons enable more a foresightful behavior almost eliminating the spike in the control input. However the same real-time horizon can be achieved with different prediction intervals. For choosing a proper prediction interval the relation between the time constant of the system and the length of the interval is crucial. If the interval not small enough system dynamics cannot be captured anymore, if the interval is too small the algorithm becomes inefficient.

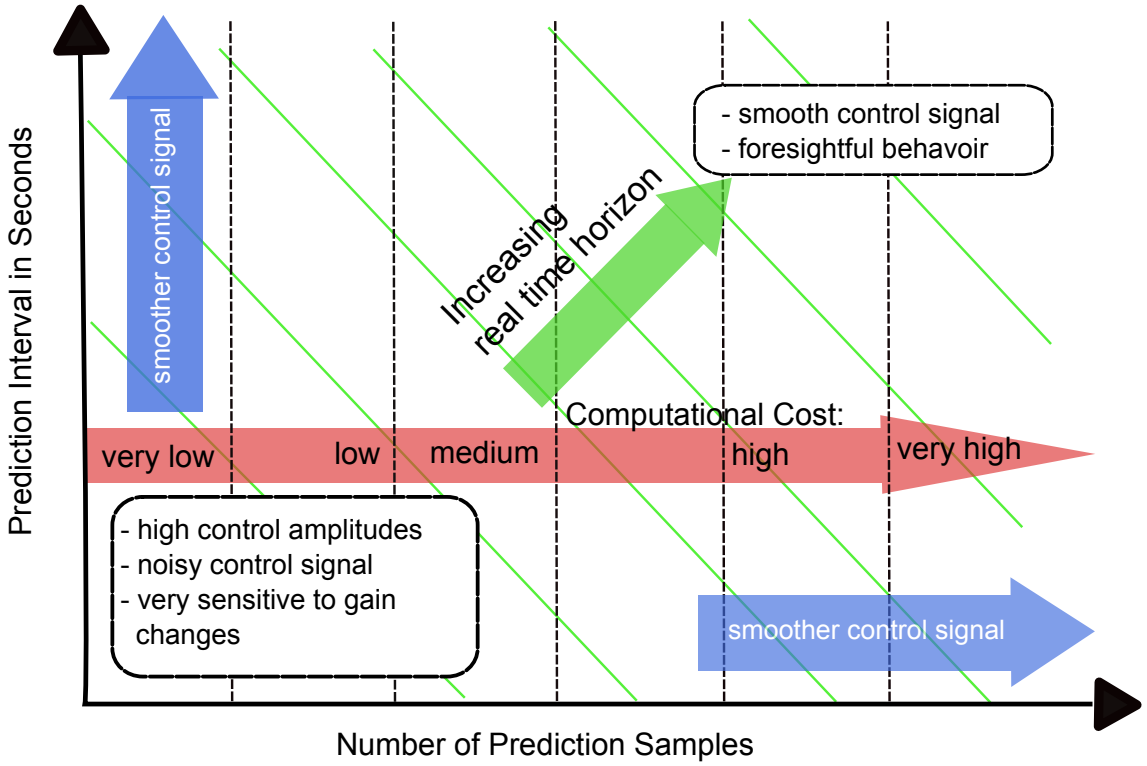

Figure 2.14: Qualitative effects of prediction horizon interval and number of prediction steps

Chapter 3

Robot Kinematic Control

In robotics tasks are commonly specified on position or velocity level in Cartesian space, e.g. a manipulator's endeffector is required to follow a given trajectory in (x, y, z) of the robot's base coordiante-system. Robot kinematic chains are controlled by assigning angles, angular velocities or angular accelerations to the joints.
In the example of figure 3.1 the kinematic controller compares the reference, a desired endeffector trajectory $\boldsymbol{r}_1, \dot{\boldsymbol{r}}_1$ with the real endeffector trajectory, $\boldsymbol{r}_{c1}, \dot{\boldsymbol{r}}_{c1}$ and computes a control input, the joint velocities $\dot{\boldsymbol{\theta}}$ in way to minimize the error between $\boldsymbol{r}_1, \dot{\boldsymbol{r}}_1$ and $\boldsymbol{r}_{1c}, \dot{\boldsymbol{r}}_{1c}$.

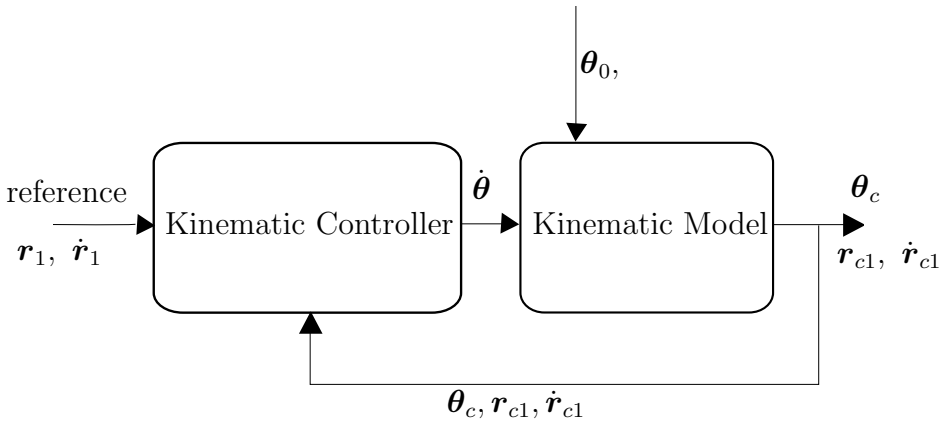

Figure 3.1: Robot Kinematic Control Scheme

The joint speeds $\dot{\boldsymbol{\theta}}$ are passed on to the kinematic model which integrates the velocities to obtain the joint angles 3.1 and computes the current endeffector position and velocity using forward kinematics 3.2:

$$\boldsymbol{\theta}_c = \int_{t_0}^{t_1} \dot{\boldsymbol{\theta}}(t)dt \tag{3.1}$$

$$\boldsymbol{r}_{1c} = \boldsymbol{f}(\boldsymbol{\theta}_c), \quad \dot{\boldsymbol{r}}_{1c} = \boldsymbol{J}_1(\boldsymbol{\theta}_c)\dot{\boldsymbol{\theta}}_c \tag{3.2}$$

Some kinematic controllers like finite horizon kinematic control do not use a feedback but rather compute a feedforward based on the inital configuration and the task trajectory. The problem that the controller solves, i.e. finding the proper joint velocities for a given endeffector velocity is called (velocity-level) inverse kinematics.

3.1 Instantaneous Robot Kinematic Control

There are numerous ways to realize a kinematic controller. The main focus of this thesis is model-predictive kinematic control. In order to rate the performance of the MPC-based controllers another more commonly-used method, the "inverse kinematics considering the order of priority" is implemented and will be described in the following section. They have been introduced by Nakamura et. al. in [8, p. 128]. Here this method shall also be referred to as instantaneous or local kinematic control since it only uses information from the current time-step by contrast to finite horizon and model-predictive control which predict the behavior to find the optimal control input within a time-interval in the future.
Nakamura's "inverse kinematics considering the order of priority" ensure the exact execution of the top priority task and use those degrees of freedom left to fullfill the lower priority task. Although any number of tasks are possible, here only the simple case of two tasks shall be addressed. The first and second priority task are specified on position and velocity level as follows:

$$\boldsymbol{r}_i = \boldsymbol{f}_i(\boldsymbol{\theta}), \quad (i = 1, 2) \tag{3.3}$$

$$\dot{\boldsymbol{r}}_i = \boldsymbol{J}_i(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}, \quad (i = 1, 2) \tag{3.4}$$

Here $\boldsymbol{\theta} \in \mathbb{R}^n$ is the joint configuration, $\boldsymbol{r}_i \in \mathbb{R}^{m_i}$ is called the manipulation variable of the i-th task. For example $\boldsymbol{r}_1$ could be the desired position and $\dot{\boldsymbol{r}}_1$ the desired velocity of the endeffector. $\boldsymbol{J}_i(\boldsymbol{\theta}) = \partial \boldsymbol{f}_i / \partial \boldsymbol{\theta} \in \mathbb{R}^{m_i \times n}$ is the Jacobian matrix of the i-th manipulation variable.
Computing the inverse kinematics on position level is much more difficult than on velocity level since it would require solving the nonlinear equation $\boldsymbol{f}_i$ in 3.3, whereas on velocity level the problem is defined only by the linear equation 3.4. In order to solve 3.4 for $\dot{\boldsymbol{\theta}}$ the Jacobian $\boldsymbol{J}_i(\boldsymbol{\theta})$ needs to be inverted. In case of a redundant manipulator i.e. $m < n$, the Jacobian becomes non-square thus the general least-squares solution has to be used:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_1^\sharp(\boldsymbol{\theta})\dot{\boldsymbol{r}}_1 + [\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp(\boldsymbol{\theta})\boldsymbol{J}_1(\theta)]\boldsymbol{y} \tag{3.5}$$

$$\text{where:} \quad \boldsymbol{J}_1^\sharp = \boldsymbol{J}_1^T(\boldsymbol{J}_1\boldsymbol{J}_1^T)^{-1} \tag{3.6}$$

Equation 3.5 yields all possible solutions $\dot{\boldsymbol{\theta}}$ satisfying 3.4, where $\boldsymbol{y}$ is an arbitrary vector in $\mathbb{R}^n$. The Moore-Penrose Pseudo-Inverse is denoted by $\boldsymbol{J}_1^\sharp$. The matrix $[\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp(\boldsymbol{\theta})\boldsymbol{J}_1(\theta)]$ is called the nullspace projection of $\boldsymbol{J}_1^\sharp$, i.e . it projects the velocity $\boldsymbol{y}$ in a way that it does not have any influence on the first manipulation variable $\dot{\boldsymbol{r}}_1$.

The vector $\boldsymbol{y}$ has to be calculated in such a way that also the second task is fulfilled. Therefore $\dot{\boldsymbol{\theta}}$ of equation 3.5 is inserted into 3.4 for $i = 2$.

$$\boldsymbol{J}_2(\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp \boldsymbol{J}_1)\boldsymbol{y} = \dot{\boldsymbol{r}}_2 - \boldsymbol{J}_2\boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1 \tag{3.7}$$

Equation 3.7 can be solved for $\boldsymbol{y}$ by applying the least square solution again:

$$\boldsymbol{y} = \hat{\boldsymbol{J}}_2^\sharp(\dot{\boldsymbol{r}}_2 - \boldsymbol{J}_2\boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1) + (\boldsymbol{E}_n - \hat{\boldsymbol{J}}_2^\sharp \hat{\boldsymbol{J}}_2)\boldsymbol{z} \tag{3.8}$$

$$\text{with: } \hat{\boldsymbol{J}}_2 := \boldsymbol{J}_2(\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp \boldsymbol{J}_1) \tag{3.9}$$

By inserting $\boldsymbol{y}$ into 3.5 $\dot{\boldsymbol{\theta}}$ can be obtained as:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1 + \hat{\boldsymbol{J}}_2^\sharp(\dot{\boldsymbol{r}}_2 - \boldsymbol{J}_2\boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1) + (\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp \boldsymbol{J}_1)(\boldsymbol{E}_n - \hat{\boldsymbol{J}}_2^\sharp \hat{\boldsymbol{J}}_2)\boldsymbol{z} \tag{3.10}$$

Here $\boldsymbol{z}$ is another arbitrary vector that represents the joint velocities that neither affect the manipulation variable of the 2. nor 3. task. If no third task is specified it can be assumed $\boldsymbol{z} = 0$. By defining the nullspace projection $\boldsymbol{N}_1 = \boldsymbol{E}_n - \boldsymbol{J}_1^\sharp \boldsymbol{J}_1$ equation 3.10 can be simplified to the following term:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1 + (\boldsymbol{J}_2\boldsymbol{N}_1)^\sharp(\dot{\boldsymbol{r}}_2 - \boldsymbol{J}_2\boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1) \tag{3.11}$$

3.1.1 The method of repulsive velocity

The prioritized inverse kinematics as introduced by Nakamura and refined by Maciejewski in [9] shall be applied on a redundant manipulator. The task is defined as tracking a path with its endeffector while at the same time avoiding an obstacle. The manipulator of figure 3.2 that shall be simulated is of planar type and has 3 degress of freedom. The top priority task is position- and velocity-control of the endeffector which are specified by the vectors $\boldsymbol{r}_1, \dot{\boldsymbol{r}}_1$. The second task accounts for collision avoidance by assigning a repulsive velocity $\dot{\boldsymbol{r}}_2$ pointing away from the obstacle. This repulsive velocity is always assigned to the point $\boldsymbol{r}_2$ on the manipulator which is closest to the obstacle.
On velocity level the second task would be defined as:

$$\dot{\boldsymbol{r}}_2 = \boldsymbol{J}_2\dot{\boldsymbol{\theta}} \tag{3.12}$$

The task specification 3.12 defines two separate conditions for the movement of point $\boldsymbol{r}_2$ if $\boldsymbol{J}_2$ has a rank equal 2. However in this example only one degree of redundancy is left for collision avoidance thus making it impossible to compute the pseudo-inverse $(\boldsymbol{J}_2\boldsymbol{N}_1)^\sharp$ needed in 3.11. To resolve this problem of over-determinateness, the task can be modified so that not the direction but only the speed of moving away from the obstacle is specified thus creating only one constraint. This can be done by projecting the velocity $\dot{\boldsymbol{r}}_2$ in the direction of the unit vector $\boldsymbol{n}_0$ pointing towards the obstacle.

$$\dot{r}_2 = \boldsymbol{n}_0^T \dot{\boldsymbol{r}}_2 \tag{3.13}$$

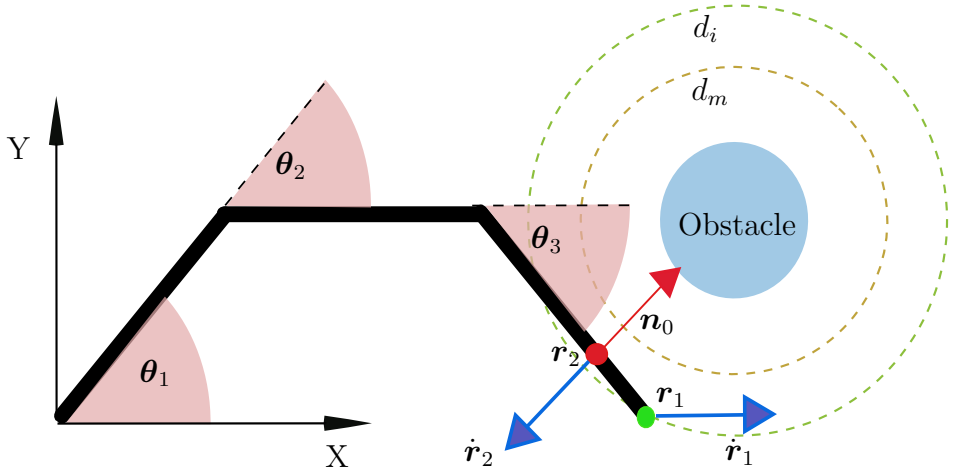

Figure 3.2: Obstacle avoidance using the method of repulsive velocity

The new Jacobian $\bar{\boldsymbol{J}}_2$ is defined by multiplying equation 3.12 with $\boldsymbol{n}_0$:

$$\dot{r}_2 = \bar{\boldsymbol{J}}_2 \dot{\boldsymbol{\theta}} \tag{3.14}$$

$$\text{with:} \quad \bar{\boldsymbol{J}}_2 = \boldsymbol{n}_0^T \boldsymbol{J}_2 \tag{3.15}$$

It is desirable that the robot moves away from obstacle faster if the distance to the obstacle becomes smaller. This is realized by multiplying the scalar velocity $\dot{r}_2$ by α_v, a scalar which becomes infinite if the distance gets zero. If the distance d is greater than the radius d_m (see figure 3.2) α_v is chosen to be zero.
Another Factor α_h is used to smoothly switch on collision avoidance if d is smaller than d_i and switch off when d is greater than d_i. The functions used for both factors are shown in figure 3.3. The factors have the following influences on the calculation of $\dot{\boldsymbol{\theta}}$:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1 + \alpha_h (\bar{\boldsymbol{J}}_2 \boldsymbol{N})^\sharp (\alpha_v \dot{r}_2 - \bar{\boldsymbol{J}}_2 \boldsymbol{J}_1^\sharp \dot{\boldsymbol{r}}_1) \tag{3.16}$$

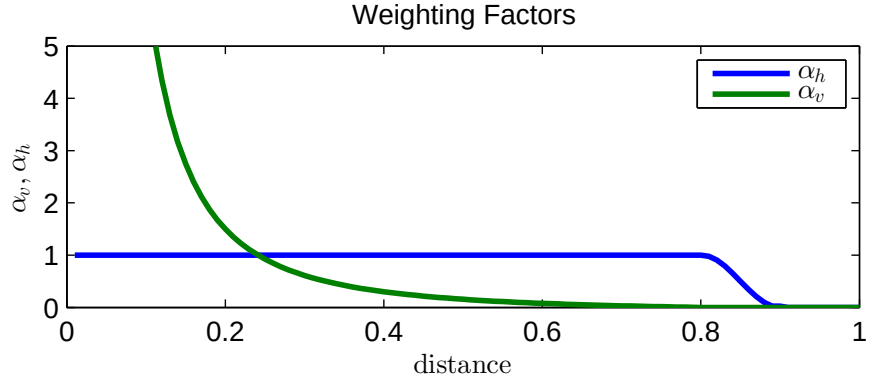

Figure 3.3: obstacle avoidance gain α_v and homogenous term gain α_h

3.1.2 Simulation of Instantaneous Kinematic Control

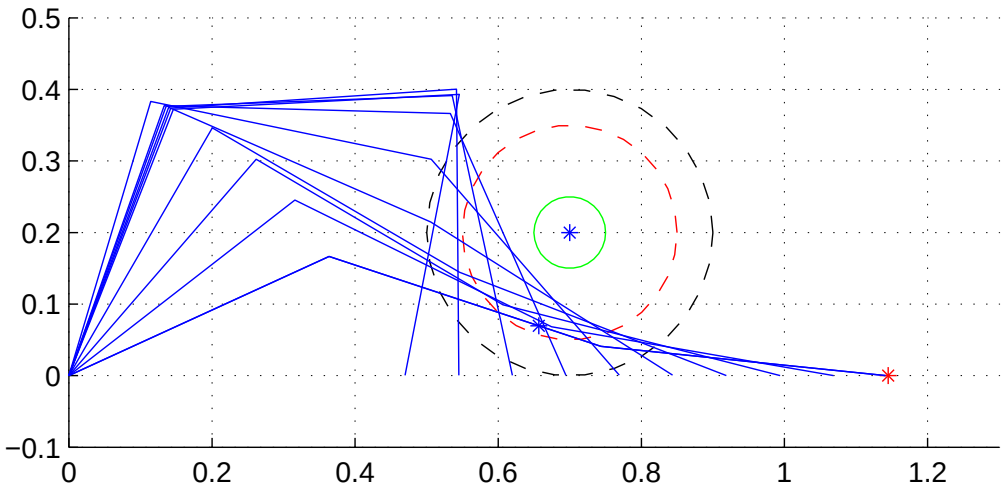

Figure 3.4: The nullspace movement is used to avoid the obstacle

While performing simulations it turned out that the algorithm gets unstable in the area of singularities, e.g. if the manipulator stretches far out. The reason is that the Jacobian of the endeffector $\boldsymbol{J}_1$ is becoming close to singular thus the entries of the pseudo-inverse $\boldsymbol{J}_1^\sharp$ reach very high values. A remedy for this problem is using a damped pseudo-inverse which computes as follows:

$$\boldsymbol{J}_1^\sharp = \boldsymbol{J}_1^T(\boldsymbol{J}_1\boldsymbol{J}_1^T + \lambda^2\boldsymbol{E}_n)^{-1} \tag{3.17}$$

Here λ is a scalar damping factor. The higher this factor, the less sensitive the pseudo-inverse is getting in the proximity of singularities. However using a higher damping of the pseudo-inverse also causes greater errors in the task execution.
Errors in the endeffector path-tracking can be reduced by feeding back the position error to the calculation of $\dot{\boldsymbol{\theta}}$. This is referred to as closed-loop inverse kinematics [10].

$$\dot{\boldsymbol{r}}_1 = \dot{\boldsymbol{r}}_{1d} + \boldsymbol{K}(\boldsymbol{r}_{1d} - \boldsymbol{r}_{c1}) \tag{3.18}$$

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_1^\sharp\dot{\boldsymbol{r}}_1 + \alpha_h(\bar{\boldsymbol{J}}_2\boldsymbol{N})^\sharp(\alpha_v\dot{r}_2 - \bar{\boldsymbol{J}}_2\boldsymbol{J}_1^\sharp\dot{\boldsymbol{r}}_1) \tag{3.19}$$

where $\dot{\boldsymbol{r}}_{1d}$ is the desired velocity $\boldsymbol{K}$ is a gain matrix and $\boldsymbol{r}_{c1}$ and $\boldsymbol{r}_{d1}$ are the current and desired endeffector positions respectively.
Recently, more intensive discussion about the use of the damped pseudo-inverse has been provided by An in 2014, [13] where an idea was proposed that separates the orthogonalization and inversion processes to eliminate errors in the task execution.

3.2 Finite Horizon Robot Kinematic Control

The instantaneous kinematic control described in the previous section is a heuristic that gives a desirable behavior at every time instant without caring about the future or past moves. However such a controller can not deal with requirements like minimizing the actuation energy on a whole trajectory.
Finite horizon kinematic control finds the optimal trajectory for the manipulator in a fixed time interval with respect to a cost function. By defining proper cost functions any desirable behavior, like minimizing actuation energy, avoiding obstacles and joint limits etc. can be specified. It is expected that predictive controller can handle more complex situations than instantaneous controllers.
This section will cover two methods of computing finite horizon kinematic control based on the calculus of variations and Pontryagin's Maximum (Minimum) Principle. The algorithms derived in this section find application in the succeeding section on MPC kinematic control.

3.2.1 Review of Variational Calculus

Both finite horizon and MPC kinematic control described in this thesis make use of the Pontraygin maximum (minimum) principle which is defined in the context of variational calculus. Apart from this other methods like dynamic programming or sequential qudratic programming (SQP) exist that would be adequate to solve nonlinear optimal control problems of this kind but will not be used in this work.

The goal of using variational calculus is to find the trajectory of inputs $\boldsymbol{u} \in \mathbb{R}^m$ for the nonlinear system[1]

$$\begin{aligned} \dot{x}_i &= f_i(\boldsymbol{x}, \boldsymbol{u}), \quad (i = 1, ..., n) \\ \boldsymbol{x} &= \mathrm{col}(x_1, x_2, ..., x_n) \in \mathbb{R}^n \end{aligned} \tag{3.20}$$

that yields a minimum value for the cost:

$$Q = \int_{t_1}^{t_2} f_0(\boldsymbol{x}, \boldsymbol{u}) dt \tag{3.21}$$

By contrast to classical variational methods which can only deal with unconstrained $\boldsymbol{u}$, Pontryagin's maximum principle can handle problems where $\boldsymbol{u} \in U \subset \mathrm{R}^m$.
The state vector $\boldsymbol{x}$ is augmented with the accumulated cost of 3.21 and called $\boldsymbol{x}_0$

$$\begin{aligned} \dot{x}_0 := f_0(\boldsymbol{x}, \boldsymbol{u}), \;\; x_0(t_1) = 0, \;\; x_0(t_2) = Q \\ \boldsymbol{x}_0 = \mathrm{col}(x_0, x_1, ..., x_n) \in \mathbb{R}^{n+1} \end{aligned} \tag{3.22}$$

[1] the following introduction to variational calculus and the Pontraygin Minimum (Maximum) Principle is based on Nakamura [8, p. 81-84]

The augmented dynamics equations are now given by:

$$\dot{\boldsymbol{x}}_0 = \boldsymbol{f}_0(\boldsymbol{x}_0, \boldsymbol{u}) \tag{3.23}$$

$$\boldsymbol{f}_0 = \text{col}(f_0, f_1, .., f_n) \quad \in \mathbb{R}^{n+1} \tag{3.24}$$

Next a variable called the adjoint vector or costate $\boldsymbol{\psi}_0 \in \mathbb{R}^{n+1}$ with the same dimension as $\boldsymbol{x}$ is introduced. Using $\boldsymbol{\psi}_0$ the Hamiltonian is defined as:

$$H_0(\boldsymbol{\psi}_0, \boldsymbol{x}_0, \boldsymbol{u}) = \boldsymbol{\psi}_0^T \boldsymbol{f}_0(\boldsymbol{x}_0, \boldsymbol{u}) \tag{3.25}$$

where the behaviors of $\boldsymbol{x}_0(t)$ and $\boldsymbol{\psi}_0(t)$ are given by:

$$\dot{\boldsymbol{x}}_0 = (\frac{\partial H_0}{\partial \boldsymbol{\psi}_0})^T \tag{3.26}$$

$$\dot{\boldsymbol{\psi}}_0 = -(\frac{\partial H_0}{\partial \boldsymbol{x}_0})^T \tag{3.27}$$

Pontryagin's Maximum Principle for Fixed-Time and Free End-States Problems [11]

> Let $\boldsymbol{u}$ be an admissible control in subset $U \in \mathbb{R}^n$ and t_1, t_2 and $\boldsymbol{x}_1$ are given, $\boldsymbol{x}_2$ is free. A necessary condition for $\boldsymbol{u}(t)$ to be optimal in the sense of 3.21 is that there exists a nonzero continuous $\boldsymbol{\psi}_0(t)$, $t_1 \leq t \leq t_2$, such that
>
> $$H_0(\boldsymbol{\psi}_0(t), \boldsymbol{x}_0(t), \boldsymbol{u}(t)) = \sup_{u \in U} H_0(\boldsymbol{\psi}_0(t), \boldsymbol{x}_0(t), \boldsymbol{u}) \tag{3.28}$$
>
> $$\boldsymbol{\psi}_0(t_2) = col(-1, 0, ..., 0) \tag{3.29}$$
>
> where ψ_0 is constantly equal to -1

3.2.2 Global Kinematic Control

Similarly to the instantaneous inverse kinematics in section 3.1, Nakamura's global kinematic control algorithm [8, p. 153] requires the definition of the two manipulation variables:

$$\boldsymbol{r}_1 = \boldsymbol{f}_1(\boldsymbol{\theta}), \quad \dot{\boldsymbol{r}}_1 = \boldsymbol{J}_1(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}, \quad \boldsymbol{r}_1 \in \mathbb{R}^{m1} \tag{3.30}$$

$$r_2 = \int_{t_0}^{t_1} p(\boldsymbol{\theta}, t)dt \quad \in \mathbb{R} \tag{3.31}$$

In the particular case covered here the equations 3.30 specify the endeffector velocity and/or orientation. The second task is the minimization of the cost integral 3.31. As in the instantaneous inverse kinematics the first task is ensured if $\dot{\boldsymbol{\theta}}$ is governed by the following equation:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_1^\sharp(\boldsymbol{\theta})\dot{\boldsymbol{r}}_1 + \boldsymbol{N}_1(\boldsymbol{\theta})\boldsymbol{y} \tag{3.32}$$

$$\stackrel{\Delta}{=} \boldsymbol{g}(\boldsymbol{\theta}, t, \boldsymbol{y}) \tag{3.33}$$

Equation 3.32 can be interpreted as a nonlinear time-varying dynamic system with $\boldsymbol{\theta}$ being the state and $\boldsymbol{y}$ being the input. The system shall be denoted by $\boldsymbol{g}(\boldsymbol{\theta}, t, \boldsymbol{y})$. The optimal control problem can now be defined as searching for the optimal input $\boldsymbol{y}$ so that the objective function 3.31 is minimized. The previously introduced PMP can be applied if the problem has a free end state. This is true because if $\boldsymbol{\theta}(t_0)$ satisfies equations $\boldsymbol{r}_0(t_0) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_0))$ then $\boldsymbol{r}_1(t_1) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_1))$ will be automatically fulfilled if $\boldsymbol{\theta}$ is governed by 3.32. Hence $\boldsymbol{\theta}(t_1)$ is free i.e. the problem has a free end-state. Using the definition 3.29 the Hamiltonian can be written as:

$$H(\boldsymbol{\psi}, \boldsymbol{\theta}, t, \boldsymbol{y}) = -p + \boldsymbol{\psi}^T \boldsymbol{g} \tag{3.34}$$

Note that H is now formulated with the normal state $\boldsymbol{\theta}$ and the costate $\boldsymbol{\psi}$ instead of the augmented $\boldsymbol{x}_0$ and $\boldsymbol{\psi}_0$. The state and adjoint vector are determined by the following set of ordinary differential equations:

$$\dot{\boldsymbol{\theta}} = (\frac{\partial H}{\partial \boldsymbol{\psi}})^T = \boldsymbol{g} \tag{3.35}$$

$$\dot{\boldsymbol{\psi}} = -(\frac{\partial H}{\partial \boldsymbol{\theta}})^T \tag{3.36}$$

Since the goal is to keep actuation effort low and avoid obstacles the objective function 3.37 is chosen to contain the joint velocities as well as a distance-cost function p_0. The distance-cost p_0 gives high values for small obstacle-manipulator distances and low values or zero for large distances.

$$r_2 = \int_{t_0}^{t_1} \underbrace{k p_0(\boldsymbol{\theta}) + \dot{\boldsymbol{\theta}}^T \dot{\boldsymbol{\theta}}}_{p} \, dt \tag{3.37}$$

Rewriting the Hamiltonian of equation 3.34 by replacing p with the integrand of 3.37 yields:

$$H = -k p_0 - \boldsymbol{g}^T \boldsymbol{g} + \boldsymbol{\psi}^T \boldsymbol{g} \tag{3.38}$$

After inserting the system equation $\boldsymbol{g}$ into the above equation the input $\boldsymbol{y}$ occurs. According to maximum principle the $\boldsymbol{y}$ that maximizes the Hamiltonian is a candidate for the optimal control input. It can be derived that such a $\boldsymbol{y}$ is given by (for details refer to Nakamura [8, p. 160]):

$$\boldsymbol{y} = \frac{1}{2}(\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp \boldsymbol{J}_1)\boldsymbol{\psi} \tag{3.39}$$

To obtain the ODE describing the behavoir of $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ the Hamiltonian of 3.38 simply has to be inserted into the differential equations 3.35 and 3.36 yielding:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{g} \tag{3.40}$$

$$\dot{\boldsymbol{\psi}} = (\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\theta}})^T (2\boldsymbol{g} - \boldsymbol{\psi}) + k(\frac{\partial p_0}{\partial \boldsymbol{\theta}})^T \tag{3.41}$$

$$\boldsymbol{g} = \boldsymbol{J}_1^\sharp \dot{r}_1 + \frac{1}{2}(\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp \boldsymbol{J}_1)\psi \tag{3.42}$$

The derivation of the gradient $\partial \boldsymbol{g}/\partial \boldsymbol{\theta}$ is given in the appendix in A.6 to A.18 and for $\partial p_0/\partial \boldsymbol{\theta}$ in equation A.3.
Since the ODE for $\dot{\boldsymbol{\theta}}$ and $\dot{\boldsymbol{\psi}}$ have n entires each it is necessary to find $2n$ boundary conditions. Depending on the problem definition there are numerous way of defining the boundary conditions having great influence on the computational complexity.

Type 1 Boundary Conditions The most straightforward way is to specify the complete initial configuration $\boldsymbol{\theta}(t_0)$. Since the problem is of fixed-time free-end-state type according to the PMP (equation 3.29) the adjoint vector at t_1 has to be zero. Thus the boundary conditions are separated into left and right hand conditions:

$$\text{L.E.:} \quad \boldsymbol{\theta}(t_0) \tag{3.43}$$
$$\text{R.E.:} \quad \boldsymbol{\psi}(t_1) = \boldsymbol{O} \tag{3.44}$$

Type 2 Boundary Conditions In [8, p. 158] Nakamura only defines position and/or orientation of the endeffector at the left-hand side, at t_0, leaving the redundant degrees of freedom open to the optimization. Therefore there are m_1 boundary conditions that can be retrieved from $\boldsymbol{r}_1(t_0) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_0))$. Again from the PMP follows the right hand boundary condition $\boldsymbol{\psi}(t_1) = \boldsymbol{O}$, giving n conditions. The remaining $n - m_1$ boundary conditions have to be derived from the transversality condition 3.45.[2] The left and right end boundary conditions can be summarized as follows:

$$\text{L.E.:} \quad \{\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp(\theta(t_0))\boldsymbol{J}_1(\boldsymbol{\theta}(t_0))\boldsymbol{\psi}(t_0)\} = \boldsymbol{O} \tag{3.45}$$
$$\boldsymbol{r}_1(t_0) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_0)) \tag{3.46}$$
$$\text{R.E.:} \quad \boldsymbol{\psi}(t_1) = \boldsymbol{O} \tag{3.47}$$

Since the differential equations 3.40 and 3.41 are coupled they need to be solved together. However they cannot be integrated as standard initial value problem since for type 1 and type 2 boundary conditions the values for $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ are defined on opposite ends. Such a problem is called a two-point boundary value problem (2-point BVP) and can be solved using the initial value adjusting method, also known as the shooting method.

An Example for the Distance-Cost Function The following function was chosen to calculate the distance-cost when using a circular obstacle:

$$p_0(\boldsymbol{\theta}) = \frac{s_{coll}}{3}\left(d_m - d_m\frac{d(\boldsymbol{\theta}) - d_b}{d_m - d_b}\right)^3 \tag{3.48}$$

Here the shortest distance between manipulator and obstacle $d(\boldsymbol{\theta})$ depends on the current configuration $\boldsymbol{\theta}$ and the location of the obstacle. d_b is the diameter of the

[2] As described in [8, p. 153] the transversality condition states that $\boldsymbol{\psi}(t_0)$ must orthogonally intersect with the mainfold $\boldsymbol{r}_1(t_0) = \boldsymbol{f}(\boldsymbol{\theta}(t_0))$. The normal vectors of this manifold are the columns of $(\partial \boldsymbol{f}_1/\partial \boldsymbol{\theta})^T$. Therefore there must exist a $\boldsymbol{q}$ so that $\boldsymbol{J}_1(\boldsymbol{\theta}(t_0))^T\boldsymbol{q} = \boldsymbol{\psi}(t_0)$. This can be equivalently written as 3.45.

circular obstacle and d_m denotes the threshold distance above which the distance-cost is set to zero. The higher the collision avoidance gain s_{coll} is, the more priority granted to collision avoidance. It is necessary that the cost is bounded to a maximum value and does not diverge when the obstacle is penetrated, i.e. $d(\boldsymbol{\theta}) < d_b$. The reason for this will be explained in section 3.3.

3.2.3 Using the Shooting Method to solve 2-point BVP

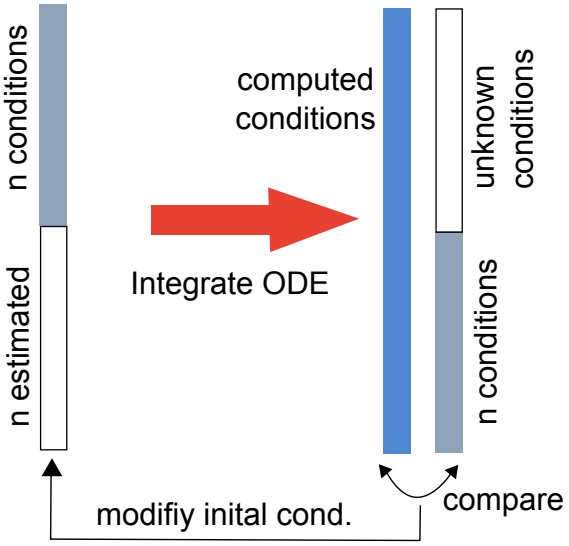

Figure 3.5: solving the 2-point boundary value problem in forward manner [8, p. 158], estimation and modification for n values

In this section the shooting method for solving the 2-point BVP is explained applying it to the boundary conditions of type 1. The scheme is depicted in figure 3.5 and works the following way: Since $\boldsymbol{\psi}(t_0)$ is unknown an initial guess is made and combining it with the known $\boldsymbol{\theta}(t_0)$ the ODE are integrated from the left to the right. If the value obtained from integration $\boldsymbol{\psi}_{int}(t_1)$ does not comply with the right hand boundary condition 3.44, i.e. $\boldsymbol{\psi}_{int}(t_1) \neq \boldsymbol{O}$, the initial guess of $\boldsymbol{\psi}(t_0)$ is adjusted and the process is repeated until it reaches $\boldsymbol{O}$ within a particular tolerance. More abstractly the problem can be understood as searching for the vector $\boldsymbol{\psi}(t_0)$ for which the nonlinear function

$$\boldsymbol{\psi}(t_1) = \boldsymbol{h}(\boldsymbol{\psi}(t_0), \boldsymbol{\theta}(t_0)) \tag{3.49}$$

being the forward integration of 3.40 and 3.41 yields $\boldsymbol{\psi}(t_1) = \boldsymbol{O}$. For simplicity the argument $\boldsymbol{\theta}(t_0)$ in function $\boldsymbol{h}$ will be omitted. The problem of finding zeros can be solved using the Newton-Raphson method. A detailed description of applying this method to solving 2-point BVPs can be found in [5, p. 343]. The initial guess is

denoted by $\boldsymbol{\psi}^{(0)}(t_0)$ and the value after integration, i.e. the output of function $\boldsymbol{h}$ is called $\boldsymbol{\psi}^{(0)}(t_1)$. Considering the current guess at step j, $\boldsymbol{\psi}^{(j)}(t_0)$, the next improved guess $\boldsymbol{\psi}^{(j+1)}(t_0)$ is computed as follows:

$$\boldsymbol{\psi}^{(j+1)}(t_0) = \boldsymbol{\psi}^{(j)}(t_0) - \left[\boldsymbol{J_h}(\boldsymbol{\psi}^{(j)}(t_0))\right]^{-1} \boldsymbol{\psi}^{(j)}(t_f) \tag{3.50}$$

$$\text{where} \quad \boldsymbol{J_h} = \frac{\partial \boldsymbol{h}(\boldsymbol{\psi}^{(j)}(t_0))}{\partial \boldsymbol{\psi}(t_0)} \tag{3.51}$$

In the equation above $\boldsymbol{J_h}$ is the Jacobian matrix obtained when deriving $\boldsymbol{h}$ at $\boldsymbol{\psi}^{(j)}(t_0)$. Since there is no analytic expression for $\boldsymbol{h}$ its Jacobian has to be determined numerically which can be done with the following procedure:

Algorithm 1 Compute Jacobian Numerically

$\boldsymbol{\psi}(t_1) \leftarrow \boldsymbol{h}(\boldsymbol{\psi}(t_0))$ //integrate

for $i = 1...n$ **do**

$\boldsymbol{\varepsilon} \leftarrow [0...0]$

$\boldsymbol{\varepsilon}_i \leftarrow \delta$ // assign small variation δ to i-th entry

$\boldsymbol{\psi}_{per}(t_0) \leftarrow \boldsymbol{\psi}(t_0) + \boldsymbol{\varepsilon}$ //perturb i-th variable of $\boldsymbol{\psi}(t_0)$ by δ

$\boldsymbol{\psi}_{per}(t_1) \leftarrow \boldsymbol{h}(\boldsymbol{\psi}_{per}(t_0))$ //integrate with perturbed value

$\frac{\partial \boldsymbol{\psi}(t_1)}{\partial \psi_i(t_0)} \leftarrow \frac{\boldsymbol{\psi}_{per}(t_1) - \boldsymbol{\psi}(t_1)}{\delta}$ //compute derivative with respect to $\psi_i(t_0)$

$(\boldsymbol{J_h})_{1...n,i} \leftarrow \frac{\partial \boldsymbol{\psi}(t_1)}{\partial \psi_i(t_0)}$ //assign the result to the i-th column of $\boldsymbol{J_h}$

end for

This algorithm computes the Jacobian by perturbing each of the entries of the input-vector $\boldsymbol{\psi}(t_0)$, computing the difference to the original value and dividing by the amount it has been perturbed.

Guessing an Appropriate $\boldsymbol{\psi}(t_0)$ Depending on the initial guess for $\boldsymbol{\psi}(t_0)$ the algorithm can either converge to a local minimum, a global minimum or not converge at all. Since $\boldsymbol{\psi}(t_1) = \boldsymbol{O}$ is a necessary but not a sufficient condition for a minimum it is possible to find starting values for $\boldsymbol{\psi}(t_0)$ where the algorithm converges to a solution which is not the optimal. The resulting trajectory yields significantly higher costs than the optimum and often collides with obstacle. A remedy to this problem could be to run the algorithm with multiple initial guesses in parallel and then choose the trajectory with the least cost.
By experimenting with different starting values $\boldsymbol{\psi}(t_0)$ in a collision avoidance scenario it was found that for $\boldsymbol{\psi}(t_0) = [0, -10, 10]$ the algorithm converges to the

global optimum. With this guess the algorithm determined the correct boundary condition to be approximately $\boldsymbol{\psi}(t_0) = [-18.5810, -45.7086, -8.0492]$. However simulation showed that the algorithm is very sensitive to the selection of $\boldsymbol{\psi}(t_0)$ and slight changes in this value can already lead to a local minimum. It turned out that the guess $\boldsymbol{\psi}(t_0) = [0, -10, 10]$ was working for most horizons and various obstacle avoidance situations.
Figure 3.6 shows the evolution of the norm of $\boldsymbol{\psi}(t_1)$ as the Newton-Raphson algorithm searches for the zero. It can be seen that after about 20 newton-steps the algorithm found a proper $\boldsymbol{\psi}(t_0)$ that complies with the boundary condition. For shorter horizons the number of necessary newton steps reduces significantly. For example when using a horizon of 0.5 seconds it converges after five newton steps.

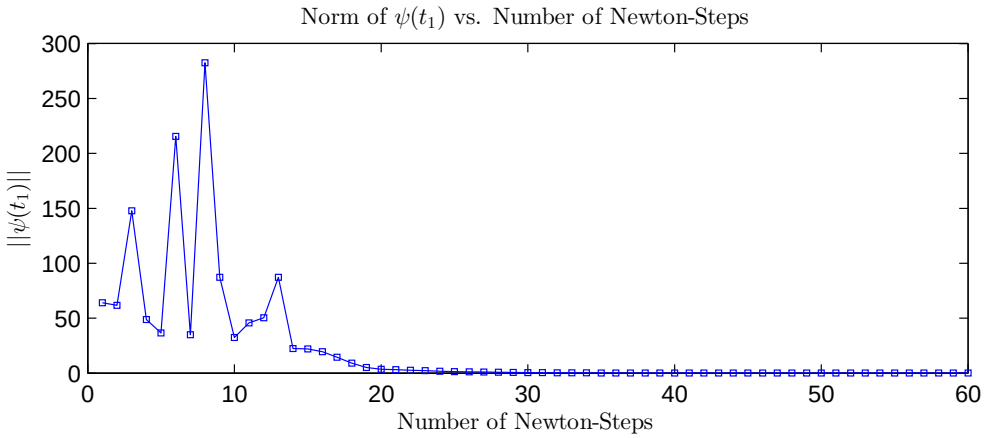

Figure 3.6: Newton Stepping in order to find the zero in $||\psi(t_1)||$

3.2.4 Nakamura's Method of Solving the BVP

Using the method described in the previous section one has to guess the whole vector $\boldsymbol{\psi}(t_0)$ with n entries. Since this can be computationally expensive Nakamura [8, p. 153] suggested to modify the boundary conditions of type 2, 3.45 to 3.47 so that instead of n only as many variables as the degree of redundancy, $n - m_1$ variables have to be guessed. This modification is achieved by considering that the left hand boundary condition $\boldsymbol{r}_1(t_0) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_0))$ will be automatically fulfilled on the right hand side, if $\boldsymbol{\theta}$ is governed by the equation 3.32. Conversely if the condition is moved to the right hand-side, i.e. $\boldsymbol{r}_1(t_1) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_1))$ the original condition $\boldsymbol{r}_1(t_0) = \boldsymbol{f}_1(\boldsymbol{\theta}(t_0))$ will also be fulfilled. Therefore the following equations are equivalent to the boundary conditions of type 2:

Type 3 Boundary Conditions

$$\text{L.E.:} \quad \{\boldsymbol{E}_n - \boldsymbol{J}_1^\sharp(\theta(t_0))\boldsymbol{J}_1(\boldsymbol{\theta}(t_0))\boldsymbol{\psi}(t_0)\} = \boldsymbol{O} \tag{3.52}$$

$$\text{R.E.:} \quad \boldsymbol{r}_1(t_1) = \boldsymbol{f}(\boldsymbol{\theta}(t_1)) \tag{3.53}$$

$$\boldsymbol{\psi}(t_1) = \boldsymbol{O} \tag{3.54}$$

Using this kind of boundary conditions the integration is carried out backwards in time while only $n - m_1$ boundary conditions have to be guessed (see figure 3.7).

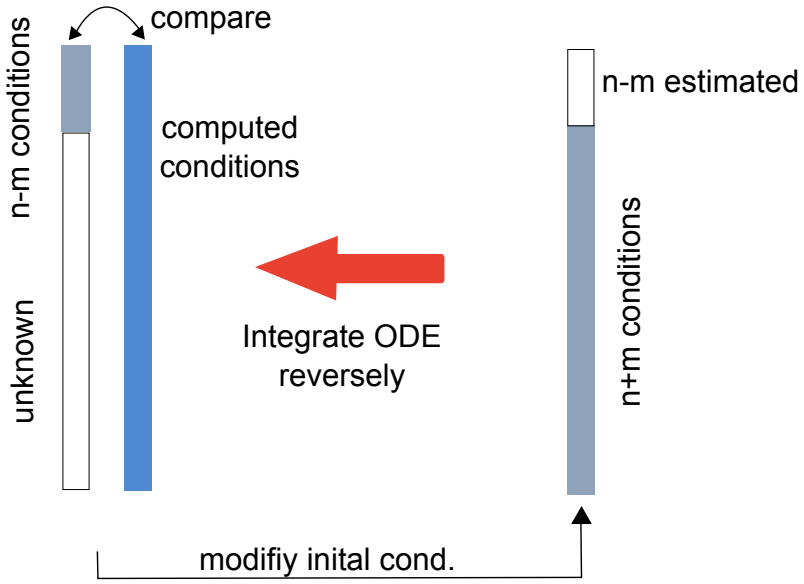

Figure 3.7: Integrating reversely with type 3 boundary conditions, the number estimated and modified values are reduce to $n - m_1$

Simulation using Type 3 Boundary Conditions A Simulation using type 3 boundary conditions is conducted for a 3-link planar manipulator where the endeffector's task is to track a horizontal line. The endeffector path as well as the endeffector locations at t_0 and t_1 are known. Considering boundary condition 3.53 all configurations where the endeffector is at position $\boldsymbol{r}_1(t_1)$ are valid initializations for the backward-integration. Figure 3.8 shows the two possible ways of exploration of the redundant space at time t_1. Both endeffector (right end) and base (left end) hold their locations while all configurations between the two extremes, marked in red are visited.

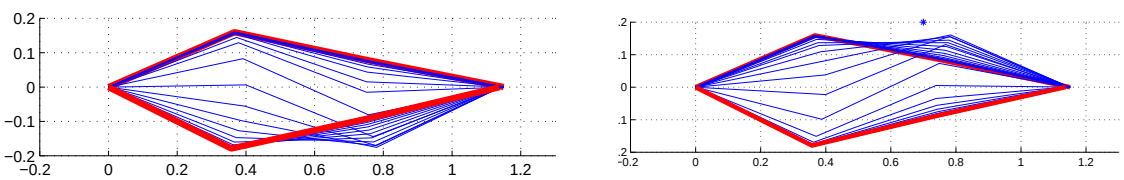

Figure 3.8: Two symmetric ways to explore the redundant space at the t_1 configuration

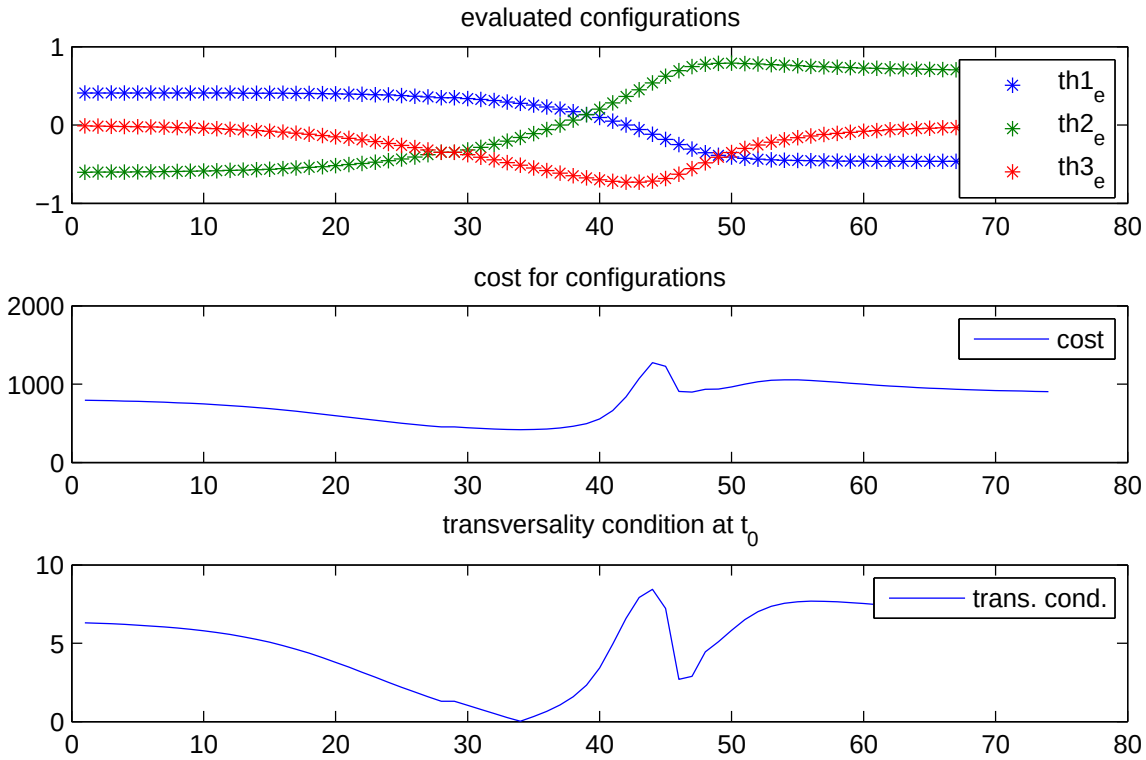

Figure 3.9: Reverse integration with type 3 boundary conditions, no collision avoidance

First the simulation is carried out without obstacle avoidance omitting the first term in the cost integral in 3.37. Both ways of exploring, shown in 3.8 yield symmetric results therefore only the results from the left one are shown in figure 3.9. The top diagram shows all the t_1-configurations that are found when exploring with some particular step-size. The middle diagram shows the accumulated cost for the trajectory of each configuration when integrating from t_1 to t_0. The bottom diagram shows the norm of the transversality condition 3.52 at time t_0. The transversality condition is a necessary condition for optimality meaning that a trajectory can only be optimal if it is satisfied, however not every zero of the transversality conditions indicates an optimal trajectory. From the diagrams in figure 3.9 it can be seen that the global optimum lies in configuration No. 34 since there is a minimum in the cost and the norm of the transversality conditions yields zero.

In the next simulation the distance cost is added to the cost integral. The results which are depicted in figure 3.10 and show that both the cost and the transversality condition become very noisy. A possible reason is that adding the distance cost changes the optimization problem in a way that a great number of local minimums are created. Therefore many point satisfy the transversality condition and show small values thus making it difficult to select a trajectory that shows obstacle avoidance.

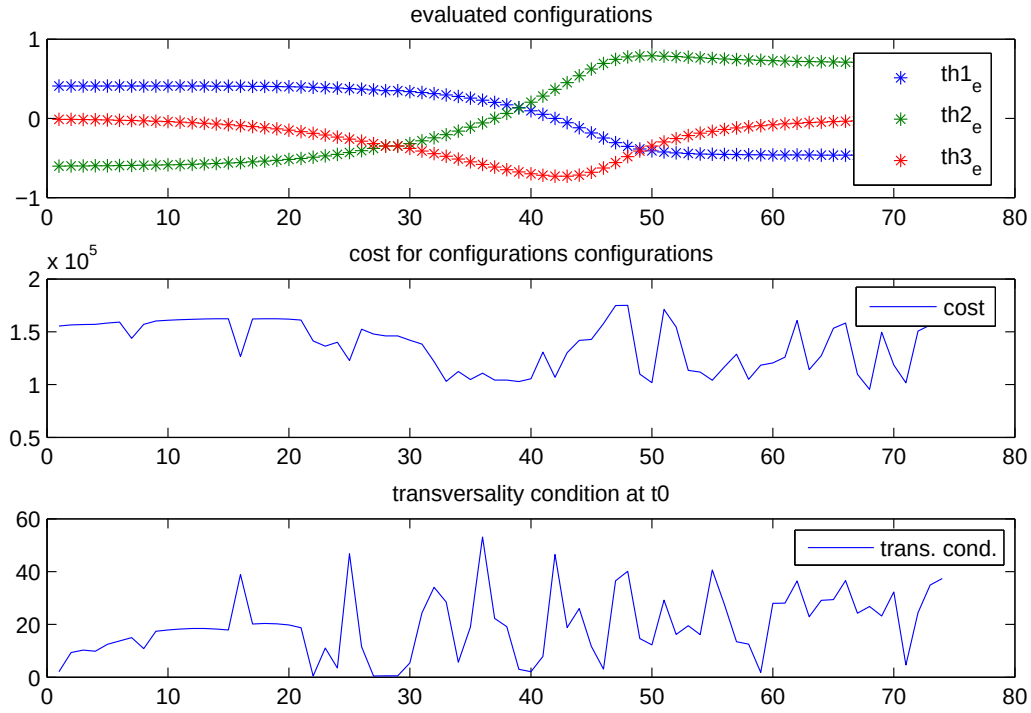

Figure 3.10: Adding Collision Avoidance, reverse integration with type 3 boundary conditions

Modification of Type 3 Boundary Conditions to enable Specification of $\boldsymbol{\theta}(t_0)$ The advantage of type 1 boundary conditions over type 2 and 3 is that the initial configuration can be specified and is not part of the solution as in type 2 and 3. Being able to specify the initial configuration is necessary for computing MPC, which will be dealt with in later sections. However the disadvantage of type 1 boundary conditions is the increased computational cost since it requires guessing and adjusting n boundary conditions instead of $n - m_1$.

The following suggestion is made in order to enable the specification of the initial configuration $\boldsymbol{\theta}(t_0)$ while reducing the number necessary guesses to $n - m_1$: Identically to type 3 boundary condition the redundant space is searched at time t_1 and the integration is carried out for every t_1 configuration. For every t_1-configuration integration yields a t_2-configuration called $\boldsymbol{\theta}_{int}$. The optimal trajectory is the one where $\boldsymbol{\theta}_{int}$ matches the original boundary condition $\boldsymbol{\theta}(t_0)$. Instead of integrating for each redundant configuration at t_1 the newton raphson-method or the gradient-descend method can be used to find configuration where the error norm $||\boldsymbol{\theta}_{int} - \boldsymbol{\theta}(t_0)|| = 0$ becomes zero. In the simulation example the bottom graph of figure 3.11 shows that this condition is fulfilled for configuration No. 34.

Type 4 Boundary Conditions

$$\text{L.E.:} \quad \boldsymbol{\theta}(t_0) \tag{3.55}$$
$$\text{R.E.:} \quad \boldsymbol{r}_1(t_1) = \boldsymbol{f}(\boldsymbol{\theta}(t_1)) \tag{3.56}$$
$$\boldsymbol{\psi}(t_1) = \boldsymbol{O} \tag{3.57}$$

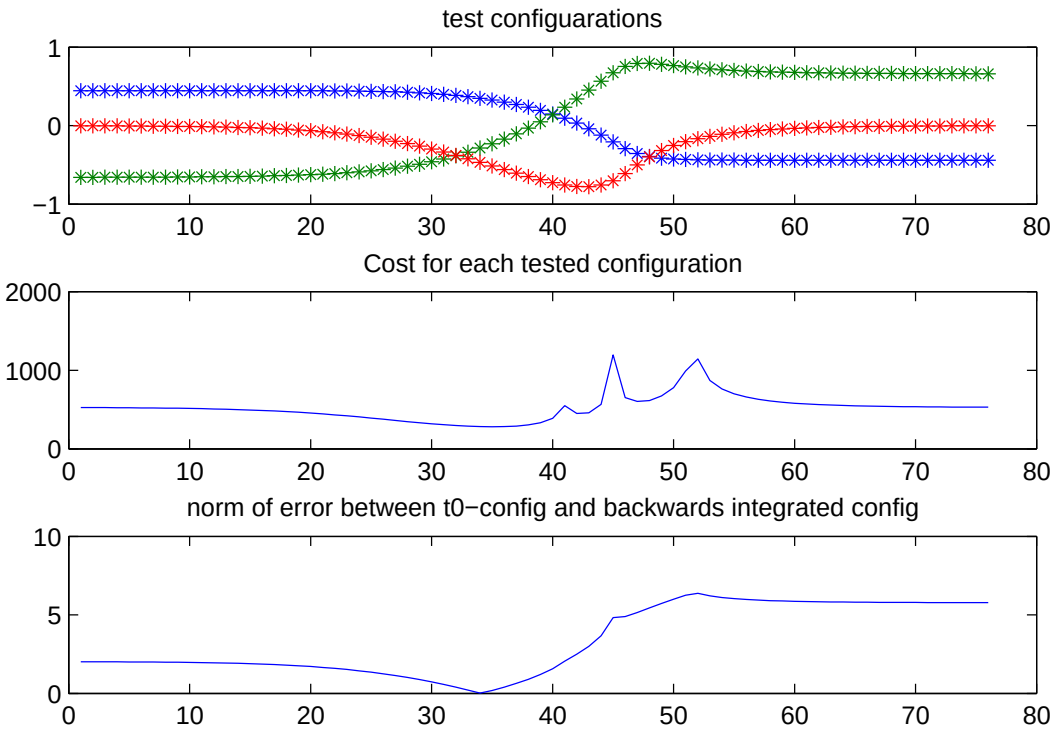

Figure 3.11: Reverse integration with type 4 boundary conditions, no collision avoidance

Regarding this section it can be summarized that using the boundary conditions of type 2 to 4 has been successfully implemented with a cost term only containing the joint velocities, i.e. only the second term of the integrand in 3.37. However when adding a distance-cost term they return very noisy results and do not provide an optimal trajectory. By contrast the approach using type 1 boundary conditions in combination with the Newton-Raphson method yields proper solutions (see figure 3.6), even when adding the obstacle avoidance term.

3.2.5 Schuetz' Method of Solving 2-point BVP

In [1] Schuetz suggested a different method to solve the 2-point boundary problem using the boundary conditions of type 1 namely:

$$\text{L.E.:}\ \ \boldsymbol{\theta}(t_0) \tag{3.58}$$

$$\text{R.E.:}\ \ \boldsymbol{\psi}(t_1) = \boldsymbol{O} \tag{3.59}$$

Starting from the formulation of the Hamiltonian in minimum principle form (p now carries a positive sign),

$$H(\boldsymbol{\psi}, \boldsymbol{\theta}, t, \boldsymbol{y}) = p + \boldsymbol{\psi}^T \boldsymbol{g} \tag{3.60}$$

Schuetz does not use the set of coupled ODE 3.40 and 3.41. Instead he uses the boundary conditions 3.58 and 3.59 to integrate the two following ODE independently, i.e. integrate $\boldsymbol{\theta}$ form left to right and $\boldsymbol{\psi}$ from right to left.

$$\left(\frac{\partial H}{\partial \boldsymbol{\psi}}\right)^T = \dot{\boldsymbol{\theta}} = \boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{y}, t) \tag{3.61}$$

$$-\left(\frac{\partial H}{\partial \boldsymbol{\theta}}\right)^T = \dot{\boldsymbol{\psi}} = -\left(\frac{\partial p}{\partial \boldsymbol{\theta}}\right)^T - \left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\theta}}\right)^T \boldsymbol{\psi} \tag{3.62}$$

The gradient $\partial H/\partial \boldsymbol{\theta}$ is given in in the appendix A.5. According to the Pontryagin minimum principle a necessary condition for the input to be optimal is that for every instant of the trajectory the Hamiltonian is minimal.

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmin}}\, H(\boldsymbol{\theta}^*, \boldsymbol{\psi}^*, \boldsymbol{y}) \tag{3.63}$$

Here * denotes the optimal trajectory. By contrast to the algorithms presented in the preceding sections, Schuetz' algorithm guesses an initial $\boldsymbol{y}$ trajectory thus being able to integrate 3.61 and 3.62 directly by using the boundary conditions 3.59 and 3.58. Therefore the problem can be reformulated as an unconstrained optimization problem, i.e. searching for the $\boldsymbol{y}$-trajectory that yields a minimum Hamiltonian at each time instant:

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmin}}\, H(\boldsymbol{\theta}(\boldsymbol{y}), \boldsymbol{\psi}(\boldsymbol{y}), \boldsymbol{y}) \tag{3.64}$$

This n-dimensional minimum-search problem can be solved in a standard way using the gradient descent method. The initial guess of $\boldsymbol{y}^0$ is iteratively improved by the following rule:

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k - \alpha \left(\frac{\partial H}{\partial \boldsymbol{y}}\right)^T \tag{3.65}$$

The gradient $\partial H/\partial \boldsymbol{y}$ is a also provided in the appendix A.4. To improve the speed of convergence Schuetz suggested to employ the conjugate gradient method which was introduced for optimal control by Mitter in [12].
The whole procedure of calculating the optimal input $\boldsymbol{y}$ using a conjugate gradient method is summarized in algorithm 2.

Algorithm 2 Schuetz' Method for Finite Horizon Kinematic Control

$j \leftarrow 0$
$\boldsymbol{\theta}^0(\tau) \leftarrow \int_\tau \dot{\boldsymbol{\theta}}(\boldsymbol{y}^0, t)dt$ //integrate with guessed $\boldsymbol{y}^0$-trajectory
repeat
  $\boldsymbol{\psi}^j(\tau) \leftarrow \int_\tau \dot{\boldsymbol{\psi}}(\boldsymbol{\theta}^j, \dot{\boldsymbol{\theta}}^j, \boldsymbol{y}, t)dt$
  $\boldsymbol{g}^j \leftarrow \frac{\partial H}{\partial \boldsymbol{y}}$ //calculate the gradients for each time-step
  if $j \neq 0$ **then**
    $\beta^j \leftarrow \frac{(\boldsymbol{g}^j, \boldsymbol{g}^j)}{(\boldsymbol{g}^{j-1}, \boldsymbol{g}^{j-1})}$
    where $(\boldsymbol{g}^j, \boldsymbol{g}^j) = \int_\tau (\boldsymbol{g}^j(t))^T \boldsymbol{g}^j(t)dt$
    $\boldsymbol{s}^j \leftarrow -\boldsymbol{g}^j + \beta^j \boldsymbol{s}^{j-1}$ //adjust stepping direction
  else
    $\boldsymbol{s}^j = -\boldsymbol{g}^j$ //initialize stepping direction
  end if
  $\boldsymbol{y}^{j+1} \leftarrow \boldsymbol{y}^j + \alpha \boldsymbol{s}^j$ //perform gradient descent
  $j \leftarrow j + 1$
  $\boldsymbol{\theta}^j(\tau) \leftarrow \int_\tau \dot{\boldsymbol{\theta}}(\boldsymbol{y}^j, t)dt$
  exit $\leftarrow (j < j_{max}) \vee \left(\frac{L^j - L^{j-1}}{L^{j-1}} < \epsilon\right) \vee (L^j > L^{j-1})$ //termination conditions
until exit = true
return $\boldsymbol{u}^j$

The conjugate gradient method improves the gradient descent by modifying the stepping direction $\boldsymbol{s}^j$ with a correction term $\beta^j \boldsymbol{s}^{j-1}$.
Schuetz suggested to use three termination conditions: Terminating after a maximum number of iterations j_{max}, if the relative cost decrease falls below a threshold ϵ and if the cost increases.
During simulations it turned out that the stepping length gain α^j has great influence on the results of the algorithm. If it chosen too small the algorithm will converge slowly being inefficient and might reach its maximum number of steps j_{max} without converging at all. If it is too large the algorithm escapes from the minimum, the cost increases and the algorithms terminates with a non-optimal result. The convergence of the cost in Schuetz algorithm, depicted in figure 3.12 is smooth by contrast to the convergence the norm of $\boldsymbol{\psi}(t_1)$ in the shooting method shown in figure 3.6.

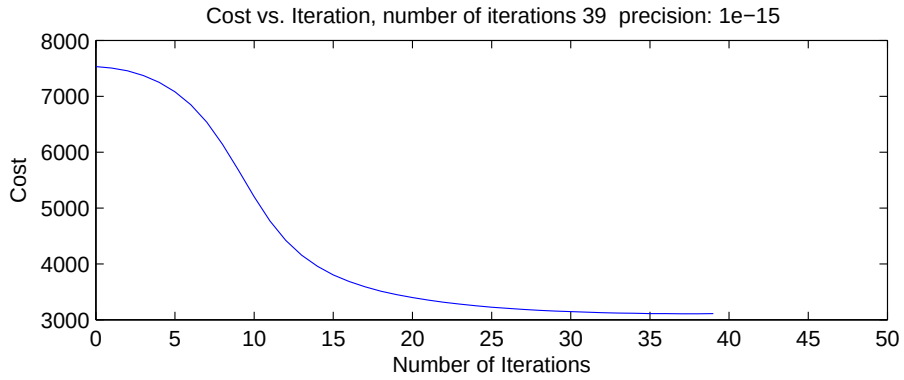

Figure 3.12: Convergence of the cost, Finite Horizon Length: 1.5 seconds

However an undesirable behavior has been observed when the termination condition is disabled and the gradient decent algorithm is continued despite increasing cost. As gradient descent always seeks the minimum, one would expect the algorithm to stay at the minimum or oscillate around the minimum. Instead as shown in figure 3.13 the cost increases with ongoing iterations, even showing an accelerated ascent. The ascent could already start before reaching to the optimal trajectory thus the algorithm would only provide an approximation of the optimal trajectory.

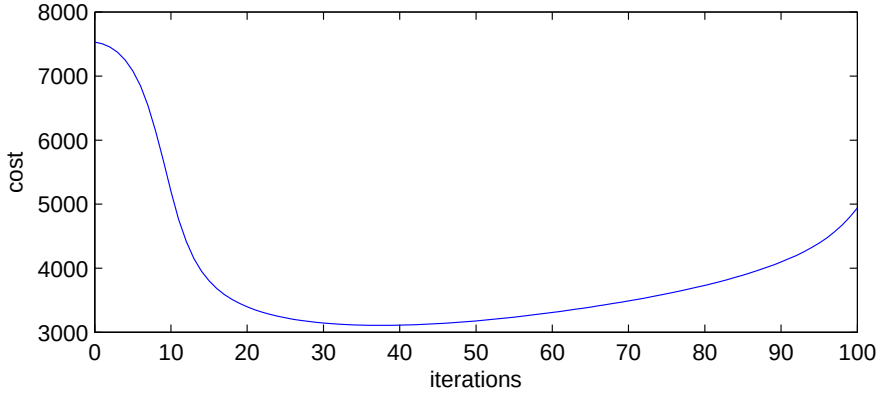

Figure 3.13: The cost goes up again, despite gradient descent thus revealing an instability in the algorithm

Another evidence that the algorithm gives an approximation of the optimal is related to the choice of the initial $\boldsymbol{y}$ trajectory. Simulation showed that using the normal termination conditions different initial $\boldsymbol{y}$ trajectories lead to different values for the accumulated cost. Possible reasons why this algorithm leads to an approximate result will be discussed in section 3.2.6.

3.2.6 Comparison of Finite Horizon Control Methods

The Shooting method, Schuetz' method and the instantaneous inverse kinematics have all shown the ability to perform collision avoidance with the planar 3-link manipulator (see figure 3.14).

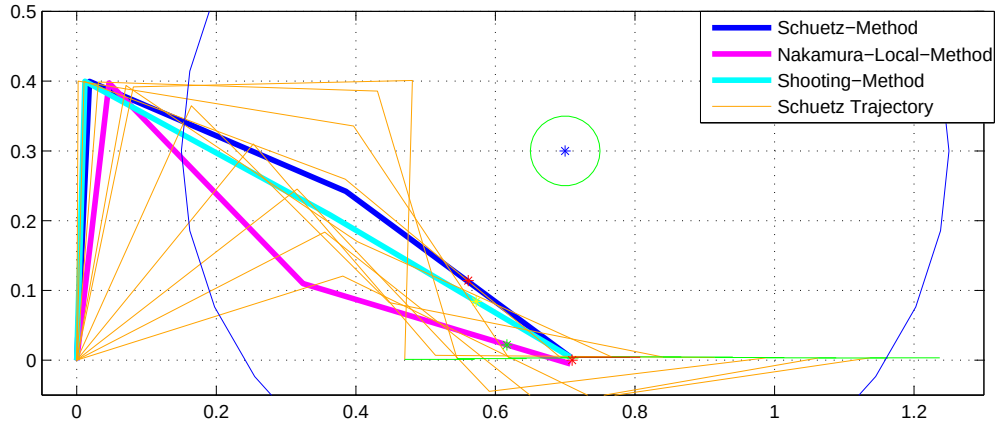

Figure 3.14: Simultaneous Simulation of Finite Horizon Kinematics Control with Schuetz' Method, the Shooting Method and Nakamura's Instantaneous Kinematic Control

Since the finite horizon control methods lay the basis for MPC, it is important to verify if the resulting trajectories are indeed optimal. Comparing the integrated costs between the three algorithms indicates which algorithm yields a trajectory that is closer the optimum. In order to guarantee an objective comparison the same initial configuration, obstacle, cost function and identical endeffector velocities are applied. The precision of the shooting method and Schuetz' method is set very high by reducing the step lengths and increasing the number of iterations. The following relations among the accumulated costs L are obtained from a simulation of horizontal tracking with obstacle avoidance on a finite horizon of 1.5 seconds:

$$\frac{L_{instantaneous} - L_{shooting}}{L_{shooting}} = 14.079\%$$
$$\frac{L_{Schuetz} - L_{shooting}}{L_{shooting}} = 6.5784\%$$

As expected, instantaneous control is far less optimal than both finite horizon control methods. An important result is that the accumulated cost for the trajectory calculated by Schuetz' algorithm is significantly higher than the cost obtained from the Shooting method. Figure 3.15 shows that this difference in the accumulated costs becomes larger with increasing horizon lengths.
The disparity in the cost values does not change for different endeffector trajectories and different obstacle locations. Therefore it is evident that Schuetz' Method only approximates the optimal trajectory.

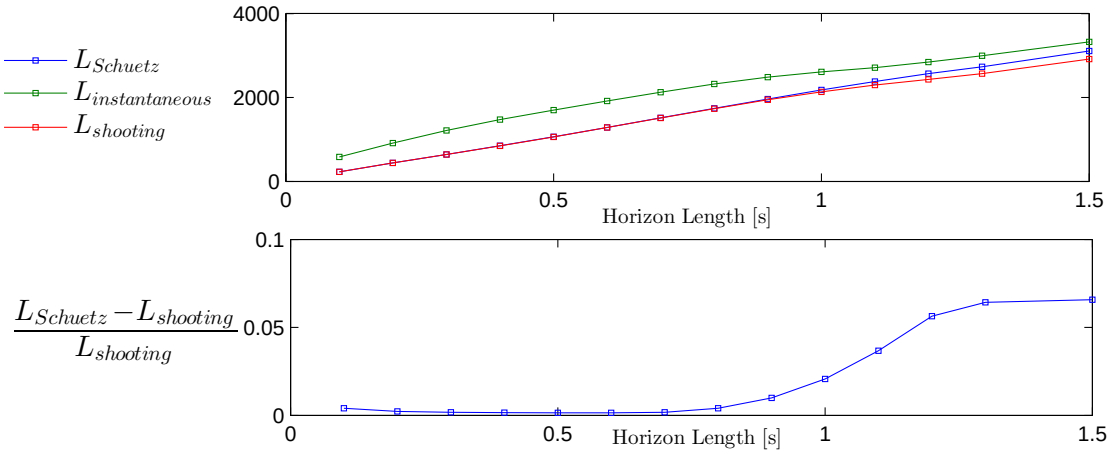

Figure 3.15: Comparison of accumulated costs for the Schuetz' Method and Shooting Method; Finite Horizon Control, with increasing horizon lengths

One possible reason why Schuetz' method only approximates the optimal solution could be the following: When Schuetz formulates the derivative of the Hamiltonian with respect to $\boldsymbol{u}$ in equation 3.62, he neglects the third term:

$$\frac{\partial H}{\partial \boldsymbol{u}} = \frac{\partial p}{\partial \boldsymbol{u}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} + \underbrace{\boldsymbol{g}^T \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{u}}}_{=0} \tag{3.66}$$

Neglecting this term causes an error in the gradient which is used as the direction for gradient descent. Furthermore Schuetz assumed the first term in the derivate of $\boldsymbol{g}$ to be zero:

$$\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} = \frac{\partial}{\partial \boldsymbol{u}} \left(\boldsymbol{J}_1^\sharp(\boldsymbol{q}(\boldsymbol{u}))\dot{\boldsymbol{r}}_1 + \boldsymbol{N}_1(\boldsymbol{q}(\boldsymbol{u}))\boldsymbol{u} \right) \tag{3.67}$$

$$= \dot{\boldsymbol{r}}_1^T \frac{\partial \boldsymbol{J}_1^\sharp}{\partial \boldsymbol{q}} \underbrace{\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{u}}}_{=0} + \boldsymbol{u}^T \frac{\partial \boldsymbol{N}_1}{\partial \boldsymbol{q}} \underbrace{\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{u}}}_{=0} + \boldsymbol{N}_1 \tag{3.68}$$

Neglecting these terms causes an error in the gradient which is used as the direction for gradient descent. This could explain why the cost increases again after reaching the bottom value (see figure 3.13)
Another possible reason is that Schuetz' algorithm converged to local minimum different from the minimum the shooting method converged to. Since the problem solved here is a non-convex optimization problem both algorithms can, depending on the guesses for the initial $\boldsymbol{y}^0$ or $\boldsymbol{\psi}(t_0)$ converge to either local or global optimums.

3.3 The Cost Function Dilemma

For both the shooting method and Schuetz' method a finitely bounded distance-cost 3.48 is used for collision avoidance (see also bottom right figure in 3.16). A finitely bounded cost does not guarantee that the obstacle is avoided. The reason is that in extreme cases, when obstacle avoidance requires very large velocities $\dot{\boldsymbol{\theta}}$ (the velocity adds quadratically to the cost) or the maximum value in the distance-cost is too low, penetrating the obstacle causes less cost than avoiding it, thus the trajectory does not evade the obstacle anymore.

An obvious approach to guarantee obstacle avoidance is to define a distance cost that diverges when the distance becomes zero (bottom left figure in 3.16).

However neither the shooting method nor Schuetz' method can run with an unbounded distance-cost function. Considering the shooting method the problem is that when integrating with the first initial guess for $\boldsymbol{\psi}(t_0)$ the trajectory can possibly cross the obstacle (top left figure in 3.16). When touching the obstacle, the distance-cost returns infinity thus causing the algorithm to fail. In case of the Schuetz' method, the initial guess $\boldsymbol{y}(t)$ can also lead to a trajectory passing through obstacles.

One method to counteract this problem would be to use a finite distance cost and iteratively increase the maximum value of the distance-cost if a collision is detected. There exist several methods trying to solve this problem like the penalty method and the augmented lagrangian method. However resolving this issue is not in the focus of this thesis.

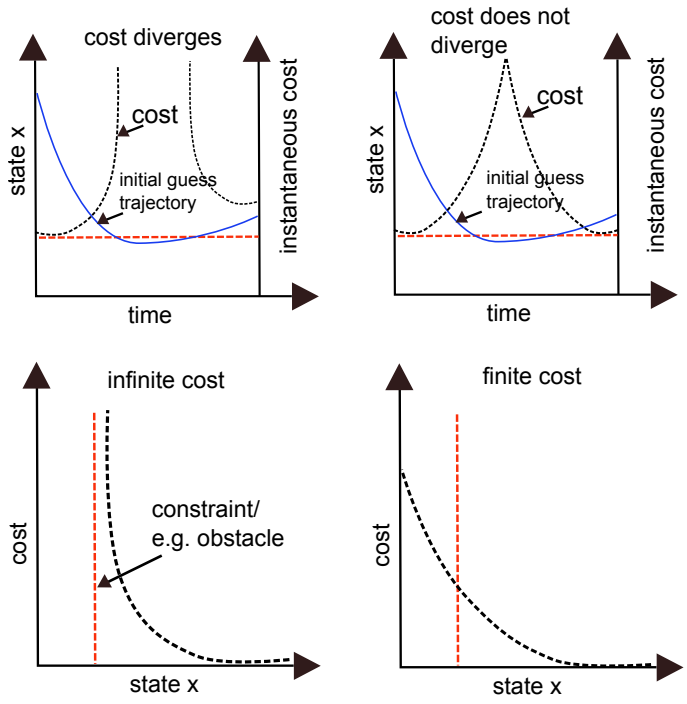

Figure 3.16: Different behavior for diverging and finite cost functions

3.4 MPC Robot Kinematic Control

The main drawback of finite horizon kinematic control is that the computational burden increases significantly with longer horizons and higher degrees of redundancy. A remedy is to compute the optimal trajectory not for the whole duration of a particular task but only for a shorter, predefined horizon. The optimal joint-trajectory is recomputed in every cycle and the first sample of this trajectory is applied on the robot. The optimization performed at every step is identical to the finite horizon kinematic control introduced in the previous section.

Algorithm 3 Model Predictive Kinematic Control

$T_{start} = t_0 = 0$
$n = T_{end}/\Delta t_{sim}$
$n_{plan} = \Delta T_{horizon}/\Delta t_{sim}$
for $k = 0$ *to* n **do**
  $\boldsymbol{\theta}_{plan,0} \leftarrow \boldsymbol{\theta}_{real}(t_k)$ //assign current configuration to $\boldsymbol{\theta}_{plan}(t_0)$
  $\dot{\boldsymbol{\theta}}_{plan} \leftarrow$**FiniHorControl**$(\boldsymbol{\theta}_{plan,0}, n_{plan}, \Delta t_{sim})$//Shoot.- or Schuetz Method
  $\dot{\boldsymbol{\theta}}_{real}(t_k) \leftarrow \dot{\boldsymbol{\theta}}_{plan}(t_k)$ //apply first sample of $\dot{\boldsymbol{\theta}}_{plan}$ to the robot
  $t_{k+1} \leftarrow t_k + \Delta t_{sim}$
end for

Algorithm 3 shows an MPC simulation for the time interval $t_k \in [T_{start}; T_{end}]$. The variable Δt_{sim} is the simulation time-step, n defines the number of simulations steps, $\Delta t_{horizon}$ denotes the prediction horizon and n_{plan} are the number of steps for the finite horizon calculation. In every cycle the current configuration is used as the initial configuration for the finite horizon control calculation. Function **FiniHorControl()** uses either Schuetz' method or the shooting method to compute the optimal trajectory of joint velocities $\dot{\boldsymbol{\theta}}_{plan}$ for n_{plan} steps with the interval of Δt_{sim}. The first sample $\dot{\boldsymbol{\theta}}_{plan}(t_k)$ is then applied to the robot.
Figure 3.17 shows the two MPC-algortihms and instantaneous kineamtic control running at the same time. The difference between the two MPC algorithms using a horizon of$\Delta t = 0.35$ is only marginal. The horizon of Schuetz' MPC algorithm is indicated by a series of yellow previews of the planned trajectory.

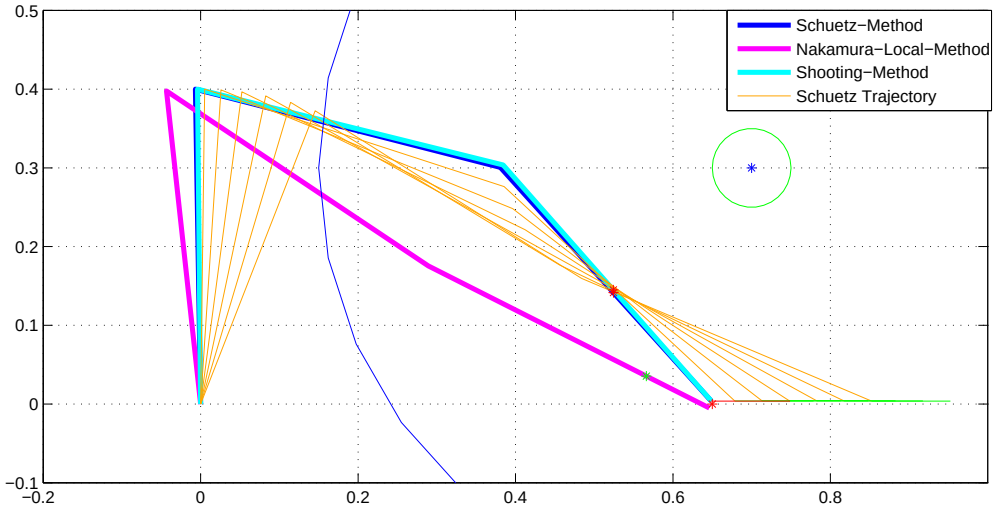

Figure 3.17: Collision avoidance, Comparison of MPC and Instantaneous Inverse Kinematics

3.5 Comparison of Kinematic Control Methods

In this section the MPC kinematic controllers based on Schuetz- and the shooting method shall be compared in terms of optimality and computational complexity.

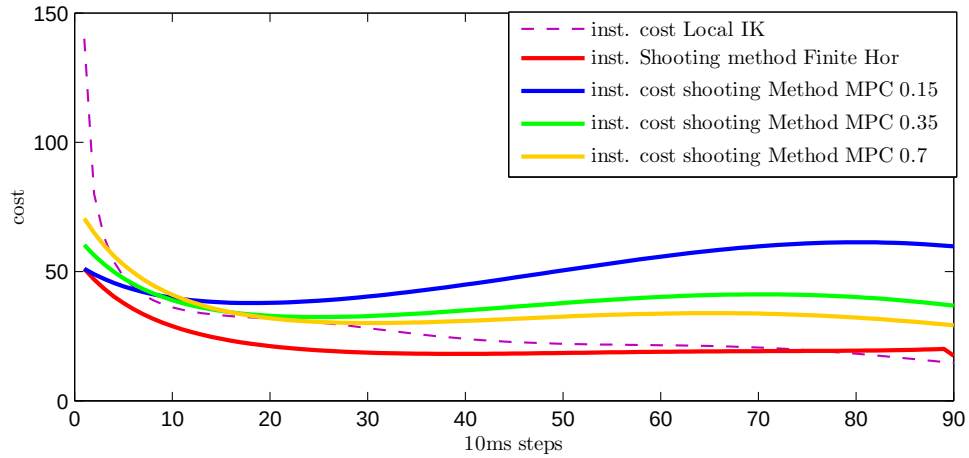

Figure 3.18: Collision avoidance, Comparison of MPC and Instantaneous Inverse Kinematics

Comparison of Optimality Figure 3.18 shows the instantaneous costs for different algorithms when using the same setup, i.e. the same cost function, the same endeffector trajectory etc. The instantaneous costs of finite horizon control (continuous red line) is the lowest of all algorithms for most of the time since finite horizon control produces the behavior closest to the optimum. The simulation shows that with increasing MPC-horizons the instantaneous costs get lower, approaching the

cost of finite horizon. However in general with increasing horizons the trajectories do not necessarily converge to the trajectories generated by finite horizon control. For example an MPC using a horizon of 0.9 seconds gives different results than a finite horizon control for the duration of 0.9 seconds. Since MPC uses a moving time window, the last control input would be optimized for the time frame of 0.9 to 1.8 seconds, whereas finite horizon solely considers the span from 0 to $0.9s$.
Figure 3.19 shows the integrated costs for different horizons lengths with a simulation duration of 0.8 seconds. It is important to mention that the instantaneous inverse kinematics produce lower integrated costs than both of the MPC-methods when the MPC horizons is less than $\approx$ 0.4 seconds (see figure 3.19). If optimality is considered as the only performance criteria MPC in this scenario becomes superior to instantaneous control when using a horizon of more than $\approx$ 0.4 seconds.
In section 3.2.6 it was pointed out that finite horizon control using the shooting method causes a significantly lower cost than using Schuetz' method. However this observation apparently does not apply on the MPC-case in general. According to figure 3.19 depending on the horizon length the integrated costs for MPC using the Shooting Method are higher for horizons of less than $\approx$ 0.55 seconds and lower for horizons longer than $\approx$ 0.55 seconds.

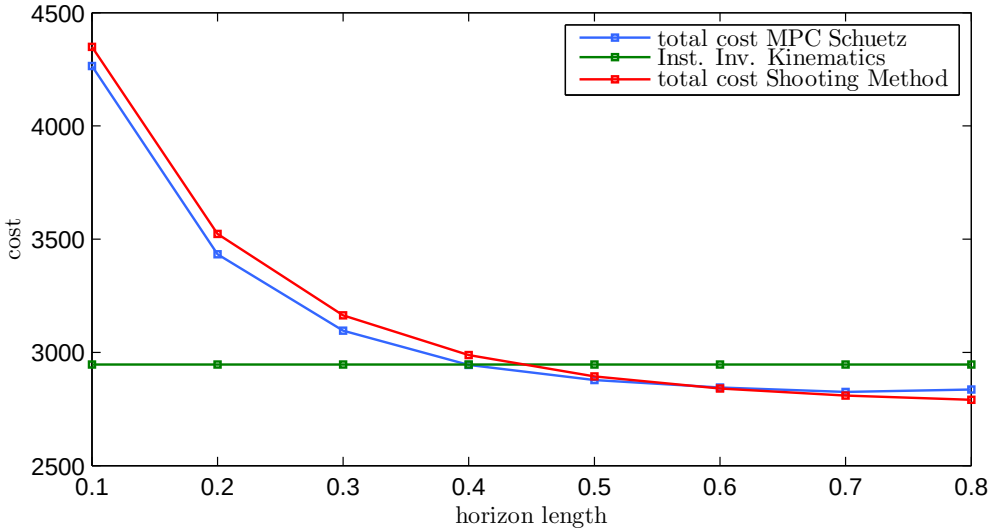

Figure 3.19: Total Costs for different MPC-horizons in a Simulation of 0.8 seconds

For this comparison the termination conditions for Schuetz'- and the shooting method are decisive. In most control steps Schuetz' algorithm terminated when the cost decrease fell below a predefined threshold (see figure 3.21). For increased precision this value was chosen to be 10^{-4}. The shooting method was terminated when the remaining $\boldsymbol{\psi}(t_f)$ was smaller than 10^{-4}. Figure 3.20 shows that for a specific simulation this was fulfilled at all times.

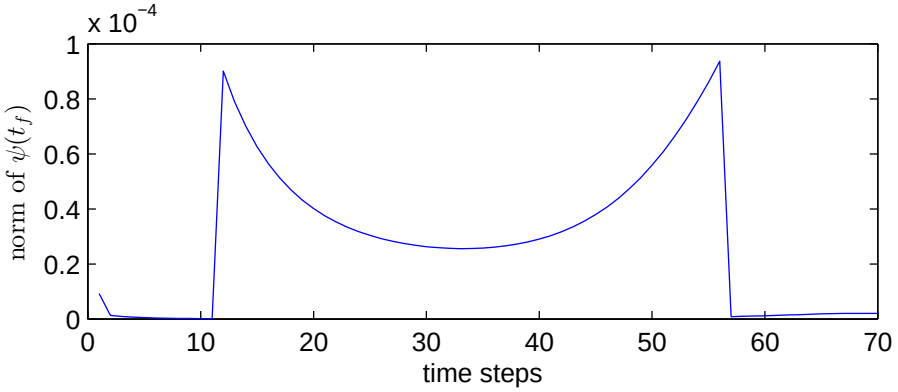

Figure 3.20: Boundary condition for Shooting Method

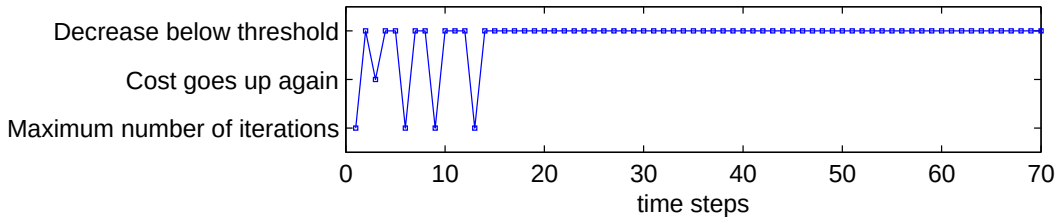

Figure 3.21: Occurence of termination conditions in Schuetz' algorithm

Comparison of Computational Complexity Besides optimality another important criterion for the performance of an algorithm is the computational cost. It depends on the time for one iteration and the number of iterations. Concerning the time for one iteration the shooting method needs to integrate over the trajectory $n+1$ times per iteration where n is the number of joints. The reason is that for one Newton-step a Jacobian matrix has to be calculated by perturbing each element of an n-vector and integrating after each perturbation (see section 3.2.3). However as Nakamura has shown this can reduced to degree of redundancy, i.e. $n - m_1$ if type 4 boundary conditions are used.
Schuetz' method performs an n-dimensional minimum value search using conjugate gradient descent. This is also computationally expensive.
The second factor for computational cost is the number of iterations needed to arrive at a satisfactory result. To compare it is necessary to define equal termination conditions for both algorithms. However Schuetz' algorithm in most cases terminates when the speed of cost decrease is less than some value , whereas the Shooting method terminates when the residuum $\boldsymbol{\psi}(t_1)$ falls below some threshold.
At the current stage Schuetz' algorithm has slightly shorter running times. However the shooting method's potential for reducing the computational load has not been fully exploited yet.

Chapter 4

Conclusion

In the first part of this thesis the concept of model-predictive control has been applied to linear systems. Subsequently MPC has been evaluated in the context of nonlinear kinematic control. Schuetz' method for computing MPC and finite horizon kinematic control and Nakamura's instantaneous inverse kinematics have been implemented. A finite horizon kinematic controller has been developed based on Nakamura's formulation of the optimal control problem by using the shooting method in conjunction with the Newton-Raphson method. For particular scenarios it has been shown that this method yields up to 6.5% less cost than Schuetz' algorithm. Based on these two finite horizon methods MPC kinematic control has been implemented and tested in a collision avoidance scenario. Both algorithms were compared with regard to the optimality of the output trajectories and computational complexity. Simulation showed that relatively long horizons are necessary for MPC to generate a more optimal behavior than well-tuned instantaneous inverse kinematics.

Future Work Future research can improve computation time and robustness of the presented methods. It would be interesting to test the MPC algorithms in more difficult situations and apply it to systems with higher degrees of freedom. Dealing with more complex obstacle shapes and taking rigid body dynamics or contacts into account is a working area with great potential.
Besides Pontryagin's Maximum (Minimum) Principle there also exist other methods to solve nonlinear control problems, like Dynamic Programming and Sequential Qudratic Programming (SQP).
Another important topic for future investigations is finding a solution to the cost function dilemma discussed in section 3.3. Possible approaches could use the penalty method or the augmented lagrangian method.

List of Figures

Bibliography

[1] C. Schuetz, T. Buschmann, J. Baur, J. Pfaff, and H. Ulbrich, "Predictive online inverse kinematics for redundant manipulators," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5056–5061, 2014.

[2] G. M. Poignet Ph., "Nonlinear Model Predictive Control of a Robot Manipulator," *Advanced Motion Control, 2000. Proceedings. 6th International Workshop on*, no. 2, 2000.

[3] Y. Tassa, *Theory and Implementation of Biomimetic Motor Controllers*. PhD thesis, Hebrew University of Jerusalem, 2011.

[4] P. S. Boyd, "Model predictive control, class ee364b lecture notes," *http://stanford.edu/class/ee364b/lectures.html*, 2013.

[5] D. Kirk, "Optimal control theory," *Prentice Hall Network Series*, pp. 10–11, 1970.

[6] F. Borrelli, A. Bemporad, and M. Morari, "Predictive control for linear and hybrid systems," *http://www.mpc.berkeley.edu/mpc-course-material*, p. 167, 2014.

[7] E. F. Camacho, "Model-predictive control," *Springer-Verlag London Limited*, pp. 48–57, 2007.

[8] Y. Nakamura, "Advanced robotics, redundancy and optimization," *Reading, Mass., Addison-Wesley*, p. 128, 1991.

[9] B. Nemec, "Kinematic Control Algorithms for On-Line Obstacle Avoidance for Redundant Manipulators," *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, no. October, pp. 0–5, 2002.

[10] B. Siciliano, "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy," *The International Journal of Robotics Research*, 1991.

[11] L. S. Pontryagin, "The mathematical theory of optimal processes," *New York [u.a.]: Interscience Publ.*, pp. 10–11, 1962.

[12] S. K. M. L. S. Ladsen and A. D. Waren, "The Conjugate Gradient Method for Optimal Control Problems," *IEEE Transactions on Automatic Control*, no. 2, p. 132, 1967.

[13] S.-i. An and D. Lee, "Prioritized inverse kinematics using qr and cholesky decompositions," vol. 2, pp. 5062–5069, IEEE International Conference on Robotics and Automation (ICRA), 2014.

[14] A. Maciejewski and C. Klein, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117, 1985.

Appendix A

Derivation of Gradients

Derivation of $\partial p_0/\partial \boldsymbol{\theta}$

$$p_0(\boldsymbol{\theta}) = \frac{s_{coll}}{3}\left(d_m - d_m\frac{d(\boldsymbol{\theta}) - d_b}{d_m - d_b}\right)^3 \tag{A.1}$$

$$d(\boldsymbol{\theta}) = ||\boldsymbol{r}_c(\boldsymbol{\theta}) - \boldsymbol{r}_{ob}|| \tag{A.2}$$

where $\boldsymbol{r}_c$ is the vector from the origin to the closest point between obstacle and robot and $\boldsymbol{r_o b}$ is the location of the obstacle.

$$\frac{\partial p_0}{\partial \boldsymbol{\theta}} = -\frac{s_{coll} d_a}{d_a - d_b}\left(d_a - d_a\left(\frac{d_k - d_b}{d_a - d_b}\right)\right)^2 \frac{2(\boldsymbol{r}_c^T - \boldsymbol{r}_{ob}^T)}{2||\boldsymbol{r}_c - \boldsymbol{r}_{ob}||}\frac{\partial \boldsymbol{r}_c}{\partial \boldsymbol{\theta}} \tag{A.3}$$

Derivation of $\partial H/\partial \boldsymbol{y}$

$$\frac{\partial H}{\partial \boldsymbol{y}} = 2(\boldsymbol{E}_n - \boldsymbol{J}^\sharp \boldsymbol{J})^T \dot{\boldsymbol{\theta}} + (\boldsymbol{E}_n - \boldsymbol{J}^\sharp \boldsymbol{J})^T \boldsymbol{\psi} \tag{A.4}$$

Derivation of $\partial H/\partial \boldsymbol{\theta}$

$$\dot{\boldsymbol{\psi}} = -\frac{\partial H}{\partial \boldsymbol{\theta}} = -2(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\theta}})^T \dot{\boldsymbol{\theta}} - (\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\theta}})^T \boldsymbol{\psi} - \frac{\partial p_0}{\partial \boldsymbol{\theta}} \tag{A.5}$$

Formulation of $\partial \boldsymbol{g}/\partial \boldsymbol{\theta}$:

$$\boldsymbol{J}_\theta := \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{\theta}} \tag{A.6}$$

$$\left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\theta}}\right)^T \boldsymbol{\psi} = \left[\boldsymbol{J}_\theta^\sharp \dot{\boldsymbol{r}}_1 - \boldsymbol{J}_\theta^\sharp \boldsymbol{J}\boldsymbol{y} - \boldsymbol{J}^\sharp \boldsymbol{J}_\theta \boldsymbol{y}\right]^T \boldsymbol{\psi} \tag{A.7}$$

$$= \underbrace{\left(\dot{\boldsymbol{r}}_1^T - \boldsymbol{y}^T \boldsymbol{J}^T\right)}_{\boldsymbol{s}^T} \underbrace{\boldsymbol{J}_\theta^{\sharp T}\boldsymbol{\psi}}_{\boldsymbol{P}[m\times n]} - \underbrace{\boldsymbol{y}^T \boldsymbol{J}_\theta^T \boldsymbol{J}^{\sharp T}}_{\boldsymbol{Q}[n\times n]}\boldsymbol{\psi} \tag{A.8}$$

$$\boldsymbol{P} = \left[(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J}_\theta - (\boldsymbol{J}J^T)^{-1}\boldsymbol{J}\boldsymbol{J}_\theta^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} - (\boldsymbol{J}J^T)^{-1}\boldsymbol{J}_\theta \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J}\right]\boldsymbol{\psi} \tag{A.9}$$

$$\boldsymbol{s}^T\boldsymbol{P} = \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\sum_{i=1}^{n}\left(\frac{\partial}{\partial\boldsymbol{\theta}_i}\boldsymbol{J}\psi_i\right) \tag{A.10}$$

$$-\boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J}\sum_{i=1}^{n}\left(\frac{\partial}{\partial\boldsymbol{\theta}_i}\boldsymbol{J}^T\psi_i\right)(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} \tag{A.11}$$

$$-\boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\sum_{i=1}^{n}\left(\frac{\partial}{\partial\boldsymbol{\theta}_i}\boldsymbol{J}\psi_i\right)\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} \tag{A.12}$$

$$\boldsymbol{s}^T\boldsymbol{P} = \underbrace{\begin{bmatrix} \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\left(\frac{\partial}{\partial\boldsymbol{\theta}_1}\boldsymbol{J}\right) \\ \vdots \\ \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\left(\frac{\partial}{\partial\boldsymbol{\theta}_n}\boldsymbol{J}\psi_i\right) \end{bmatrix}^T}_{A}\boldsymbol{\psi} \tag{A.13}$$

$$-\underbrace{\begin{bmatrix} \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\left(\frac{\partial}{\partial\boldsymbol{\theta}_1}\boldsymbol{J}^T\right)(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} \\ \vdots \\ \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\left(\frac{\partial}{\partial\boldsymbol{\theta}_n}\boldsymbol{J}\right)(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} \end{bmatrix}^T}_{B}\boldsymbol{\psi} \tag{A.14}$$

$$-\underbrace{\begin{bmatrix} \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\left(\frac{\partial}{\partial\boldsymbol{\theta}_1}\boldsymbol{J}\right)\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} \\ \vdots \\ \boldsymbol{s}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\left(\frac{\partial}{\partial\boldsymbol{\theta}_n}\boldsymbol{J}\right)\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}\boldsymbol{J} \end{bmatrix}^T}_{C}\boldsymbol{\psi} \tag{A.15}$$

$$\boldsymbol{Q}\boldsymbol{\psi} = \boldsymbol{y}^T\sum_{i=1}^{n}\left(\frac{\partial}{\partial\theta_i}\boldsymbol{J}^T\psi_i\right)\boldsymbol{J}^{\sharp T} \tag{A.16}$$

$$= \underbrace{\begin{bmatrix} \boldsymbol{y}^T\frac{\partial}{\partial\boldsymbol{\theta}_1}\boldsymbol{J}^T\boldsymbol{J}^{\sharp T} \\ \vdots \\ \boldsymbol{y}^T\frac{\partial}{\partial\boldsymbol{\theta}_n}\boldsymbol{J}^T\boldsymbol{J}^{\sharp T} \end{bmatrix}^T}_{D}\boldsymbol{\psi} \tag{A.17}$$

$$\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{\theta}} = A - B - C - D \tag{A.18}$$

License