ISSN 0280-5316
ISRN LUTFD2/TFRT--5842--SE

Optimal Control and Path Following for Industrial Robots

Martin Hast

Department of Automatic Control
Lund University
June 2009

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
June 2009

Document Number
ISRN LUTFD2/TFRT--5842--SE

Author(s)
Martin Hast

Supervisor
Anders Robertsson Automatic Control, Lund
Johan Åkesson Automatic Control, Lund
Rolf Johansson Automatic Control, Lund (Examiner)

Sponsoring organization

Title and subtitle
Optimal Control and Path Following for Industrial Robots. (Optimal reglering och trajektorieföljning för industrirobotar)

Abstract
When using industrial robots in production lines both speed and accuracy is of great importance. This thesis investigates how off-line optimization can be used to create references to a control structure with the aim of traversing a given path in as little time as possible,
under given input constraints, without deviating from the path. In this thesis Modelica and Optimica is used to formulate and solve minimum time optimization problems. For the purpose of optimization, a model of an ABB IRB140B industrial robot was identified. A control structure known as a Path Velocity Controller has been implemented in Simulink with the objective to control the IRB140B. The implemented controller was then evaluated in simulations.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
88

Recipient's notes

Security classification

http://www.control.lth.se/publications/

Acknowledgements

First I would like to thank my supervisors Anders Robertsson and Johan Åkesson who have been very supporting during this master thesis. Always there to answer questions regarding everything from optimization, robotics, LaTeX to automatic control issues in general and this report in particular. Thank you. I would also like to thank Linnea Andersson for reading the manuscript for this thesis over and over again while taunting me for my repeated linguistical mistakes. Thank you. You have really facilitated this work.

Secondly I would like to thank everyone at the Department of Automatic Control for a pleasant and interesting working environment. In particular I would like to thank Leif who has been of tremendous help when it comes to typesetting this thesis in LaTeX and computer issues in general. Without his help the work of writing the thesis would have taken forever.

At last I would like to thank the other master thesis students that I have worked in parallel with and who had to endure my, sometimes, endless talking.

Contents

Acknowledgements 5

1. Introduction 9
- 1.1 Background 9
- 1.2 Problem Definition 9
- 1.3 Method 9

2. Optimization 12
- 2.1 Defining the Optimization Problem 12
- 2.2 Reduction of the State Dimension 14
- 2.3 Optimization over a Fixed Interval 15
- 2.4 Solving the Optimization Problem 18

3. Path Velocity Controller 22
- 3.1 Control Structure 22
- 3.2 Path Velocity Controller 23

4. Case Study 27
- 4.1 Model Identification 27
- 4.2 Path Recording 30
- 4.3 Optimization 33
- 4.4 PVC in Simulink® 38
- 4.5 Controller Parameters 42
- 4.6 Simulation 42
- 4.7 Practical issues with the PVC algorithm 53
- 4.8 Using the PVC with the Robot 55
- 4.9 Experiment 56

5. Conclusions and Future Work 57

6. Bibliography 58

A. Model Identification 59

B. Optimization Results 72

C. Simulink® models 79

D. Modelica and Optimica Code 84
- D.1 Modelica Code 84
- D.2 Optimica Code 85
- D.3 MATLAB® code 86

1. Introduction

1.1 Background

Industrial robots are used for a large number of tasks in industry. In several applications, such as glueing, painting and arc welding, not only the end points but also the path as such and the speed of traversal are strongly connected to the quality and efficiency. In order to maximize plant efficiency it is crucial to maximize robot work speed. Unfortunately, the cost and length of robot programming is an obstacle for companies producing small series. This thesis provides a method for making industrial robots follow predefined paths as quickly as possible. These paths can be defined by mathematical expressions or recorded using lead-through programming.

1.2 Problem Definition

The aim of this thesis is to use Modelica [6] and Optimica [5] to solve minimum time optimization problems. The results from the optimization should be used in the Simulink® implementation of a path velocity controller, PVC. The PVC should be used to control an ABB IRB140B robot.

1.3 Method

The project consist of a number of different tasks, briefly described below:

- Identify a model of the ABB IRB140B robot.
- Use lead-through to record a path.
- Formulate an optimization problem using Modelica and Optimica.
- Solve the optimization problem.
- Implement a Path Velocity Controller in Simulink®.
- Find a suitable controller to include in the Path Velocity Controller.
- Tune the controllers to obtain a desired system response.
- Evaluate the controller in simulations.
- Finally, use the Path Velocity Controller on the robot.

These tasks require a number of different tools as described below.

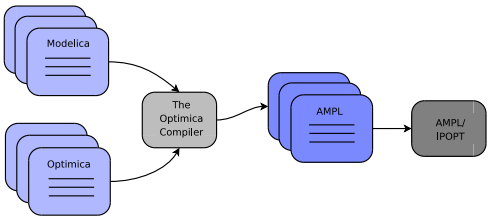

Figure 1.1 Work flow when doing optimization with Modelica and Optimica. Picture from [5].

Tools

MATLAB®, Simulink® and Real Time Workshop MATLAB® and Simulink® are used extensively throughout this thessi. MATLAB® is used for general computations and plotting of results. The System Identification Toolbox is used to identify models for the robot axis. Simulink® is used for the implementation and the simulation of the PVC. In order to run the controller on the IRB140B, Real-Time Workshop is used. Real-Time Workshop translates the Simulink® models into C-code to be run on the robot.

Modelica and Dymola Modelica is a free object-oriented modeling language developed by the members of the Modelica Association. Modelica is developed for multi-domain modelling, meaning that models can contain for instance electrical systems interacting with mechanical, hydraulic and control systems [12]. The Modelica language offers possibilities to model small parts of a system, or to use the available libraries, and to put the parts together to form large and complex models [6]. Although this thesis includes neither large nor complex systems, the Modelica language provides a convenient way of making models. As opposed to Simulink, Modelica is beneficial because models are stated using differential algebraic equations (DAE) which makes modelling much easier than using ordinary differential equations.

Dymola, "Dynamic Modeling Laboratory", is a modeling and simulation environment, developed by Dynasim, for the Modelica language. Dymola provides graphical features that enhance the use of Modelica as well as the possibility to simulate models. The simulation results can be used, e.g., to evaluate the models or to serve as an initial guess for the Optimica compiler as described below.

Optimization Tools In this thesis, Modelica together with Optimica, are used to formulate optimization problems. Modelica is used for making dynamical models, i.e., stating dynamical constraints, but lacks support for formulating the other parts of the optimization problem. Optimica is an extension to the Modelica language, allowing to express cost functions, boundaries and constraints on states and inputs [4], [5][1]. Once the problem is formulated in Modelica and Optimica code it is translated by the Optimica compiler into AMPL,"A Mathematical Programming Language",

[1]The version of Optimica used in this thesis is a prototype which differs in syntax from the one presented in [5]

see figure 1.1. AMPL is an algebraic modelling language suitable for the expression of a variety of optimization problems [11]. AMPL does not solve the optimization problem but calls for an external solver, in this thesis IPOPT[2], to do so. IPOPT solves the optimization problem and, if a feasible solution is found, produces a result file. Dymola is then used to evaluate the optimization results using plots and simulations. The chance of finding an optimal solution is substantially increased if an initial guess is given to the Optimica compiler.

Figure 1.2 ABB IRB140B [3]

Robot System

The robot that will be used in this thesis is an ABB IRB140B robot. It is ABB's smallest industrial robot with a 6 kg pay load and a 0.81 m reach [1]. The IRB140B is a serial industrial robot with six axes providing 6 degrees of freedom. Using only axis one, two and three it is possible to reach any point within the robot's workspace. Axis four, five and six allow reaching a given position from an arbitrary direction. Serial six-axis robots are therefore in general versatile and used for many applications as mentioned above. The IRB140B is controlled by an ABB IRC5 control cabinet and powered by a corresponding ABB drive module. The robot's control panel, the FlexPendant, can be used to jog and program the robot. The FlexPendant can also be used for numerous other task [2] and the ABB homepage [3]. The conventional way to program an ABB IRB140B robot is to use the robot programming language RAPID. This will however not be done in this thesis, but instead Simulink® will be used together with Real-Time Workshop to generate C-code. A program that logs signals, specified by the user, from the robot was available. For a description on how to run the robot and how to use Real-Time Workshop together with the robot available at the department see [10].

[2]IPOPT is described in [16]

2. Optimization

This chapter discusses the different parts of the time minimum optimization problem formulation. A general minimum time problem is presented in the first section. The problem is then reformulated to reduce the number of states. The reformulated problem is then reformulated again in order to obtain an optimization problem defined over a fixed interval. The reformulations are done according to the method in [8] Chapter 3, Sections 3.2 and 3.5. The goal of the optimization is to find input sequences that drive the model states from a start point to an end point along a predefined path. To do so requires a model, a predefined path for the model states to follow and a formulation of the model's constraints. The prerequisites are discussed and defined in Section 2.1.

2.1 Defining the Optimization Problem

The Model

A dynamic model of order p describes the system states q and their derivatives $\dot{q}, \ddot{q}, \ldots, q^{(p)}$ in relation to the system inputs τ. The model is assumed to be given on the form

$$\tau = g(q, \dot{q}, \ldots, q^{(p)}) \tag{2.1}$$

Eq. (2.1) constitutes the dynamical constraints of the optimization problem. We further assume that there are n inputs with the lower and upper bounds

$$\tau_i^{min} \leq \tau_i \leq \tau_i^{max}, \quad 1 \leq i \leq n \tag{2.2}$$

The Path

The predefined path is a description of how the model states should evolve. The path is defined as

$$f(s) = \begin{bmatrix} f_1(s) \\ \vdots \\ f_n(s) \end{bmatrix} \tag{2.3}$$

The path $f(s)$ is parametrized by the *path parameter* $s(t)$. We assume that the path parameter $s(t)$ is a real, piecewise twice differentiable function defined in the interval $t_0 \leq t \leq t_f$. The path's starting and ending points are defined as $s(t_0) = s_0$ and $s(t_f) = s_f$ respectively. The first derivative of s with respect to time, $\dot{s}(t) = \frac{ds}{dt}$, is called the *path velocity*. The second derivative of $s(t)$ with respect to time, $\ddot{s}(t)$, is called the *path acceleration*. If motion is assumed only to take place in the forward direction, equivalent to $\dot{s}(t) > 0$, it is possible to express the path velocity and acceleration as functions of the path parameter. We call these functions the velocity profile

$$v_1(s(t)) = \dot{s}(t) \tag{2.4}$$

and the acceleration profile

$$v_2(s(t)) = \ddot{s}(t) \tag{2.5}$$

with notation as in [8]. Eq. (2.4) and Eq. (2.5) are vital to the path velocity controller described in Chapter 3.

The Optimization Problem

The objective of the optimization is to minimize the traversal time t_f along the path, expressed as the following cost function

$$\min_{\tau} t_f = \min_{\tau} \int_{t_0}^{t_f} 1 \, \mathrm{d}t \tag{2.6}$$

Eq. (2.6) together with the constraints and boundary conditions stated below statute the optimization problem. The model, or the dynamical constraints, is defined by Eq. (2.1). The path is an equality constraint meaning that the states should always follow the path i.e., $f(s) = q$. The input limits are inequality constraints defined by Eq. (2.2). Finally, the boundary conditions are defined by the initial and final states

$$\begin{aligned} q(t_0), \dots q^{(p-1)}(t_0) \\ q(t_f), \dots q^{(p-1)}(t_f) \end{aligned} \tag{2.7}$$

This optimization problem has pn states and is generally hard to solve. Consequently, a reduction of the number of states is desirable.

An Example

In order to illustrate the procedure, an example of how an optimization problem is formulated will be given. The example will be continued in the following sections, each of them emphasizing different aspects of the problem. We assume that a process model is known and expressed as

$$M\ddot{q} + D\dot{q} = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{2.8}$$

where τ is the model inputs and q the states that should follow a path. The inputs are limited by lower and upper bounds

$$-1 \leq \tau_i \leq 1, \quad i = 1, 2 \tag{2.9}$$

The path is a straight line defined as

$$f(s) = \begin{bmatrix} 4 \\ 1 \end{bmatrix} s \tag{2.10}$$

The motion starts from rest at time $t_0 = 0$ with $q = (0, 0)$. The motion stops at rest at time t_f in position $q = (4, 1)$, giving the boundary conditions

$$\begin{aligned} q(0) &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \dot{q}(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ q(t_f) &= \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \quad \dot{q}(t_f) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned} \tag{2.11}$$

The cost function for the minimum time problem is

$$\min_{\tau} t_f = \min_{\tau} \int_0^{t_f} 1 \, \mathrm{d}t \tag{2.12}$$

Eq. (2.8), Eq. (2.9), Eq. (2.10), Eq. (2.11), and Eq. (2.12) describe all parts of the optimization problem. The optimization problem has $p = 2$ and $n = 2$, giving the total of four states. Section 2.2 presents a method of how to reduce the number of states from four to two.

2.2 Reduction of the State Dimension

Using the path constraints, the dynamics can be rewritten to reduce the number of states in the optimization problem from pn to p. The path is given by the vector Eq. $q = f(s)$. Using the chain rule the time derivatives of q can be computed. Since the model is of order p, only the derivatives of q to the order of p has to be calculated. Using the chain rule

$$\frac{df}{dt} = \frac{df}{ds}\frac{ds}{dt} \tag{2.13}$$

gives the following results

$$\begin{aligned} q &= f(s) \\ \dot{q} &= f'(s)\dot{s} \\ \ddot{q} &= f''(s)\dot{s} + f'(s)\ddot{s} \\ &\vdots \\ q^{(p)} &= \frac{dq^{(p-1)}}{dt} = \frac{dq^{(p-1)}}{ds}\frac{ds}{dt} \end{aligned} \tag{2.14}$$

Since $f(s)$ is given, and its derivative with respect to s can be calculated if putting the Eq. s (2.14) into the system dynamics (2.1), we have

$$\tau = g_s(s, \dot{s}, \ldots, s^{(p)}) \tag{2.15}$$

Eq. (2.15) serves as constraints in the rewritten optimization problem. The limits on τ are still described by Eq. (2.2) while the boundary conditions are expressed as

$$\begin{aligned} s(t_0), \ldots, s^{(p-1)}(t_0) \\ s(t_f), \ldots, s^{(p-1)}(t_f) \end{aligned} \tag{2.16}$$

The dynamics of the optimization problem are replaced by a p-order integrator

$$\begin{aligned} \frac{ds}{dt} &= \dot{s} \\ \frac{d\dot{s}}{dt} &= \ddot{s} \\ &\vdots \\ \frac{ds^{(p-1)}}{dt} &= s^p \end{aligned} \tag{2.17}$$

The cost function for the minimum time problem becomes

$$\min_{s^{(p)}} t_f = \min_{s^{(p)}} \int_{t_0}^{t_f} 1 \, \mathrm{d}t \tag{2.18}$$

where minimization is no longer done by using τ as input. Instead the p-th time derivative of s, $s^{(p)}$, serves as the input.

An Example - Reduced

Setting $q = f(s)$ with $f(s)$ as defined in (2.10) and calculating the first two time derivatives using the chain rule according to (2.13) gives

$$\begin{aligned} q &= \begin{bmatrix} 4 \\ 1 \end{bmatrix} s \\ \dot{q} &= \begin{bmatrix} 4 \\ 1 \end{bmatrix} \dot{s} \\ \ddot{q} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \dot{s} + \begin{bmatrix} 4 \\ 1 \end{bmatrix} \ddot{s} \end{aligned} \tag{2.19}$$

Inserting Eq. (2.19) into the system dynamics Eq. (2.8) gives

$$\begin{bmatrix} 4m_1 \\ m_2 \end{bmatrix} \ddot{s} + \begin{bmatrix} 4d_1 \\ d_2 \end{bmatrix} \dot{s} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{2.20}$$

The dynamics of the optimization problem is the set of integrators as defined by Eq. (2.14). Since $p = 2$ the dynamics are

$$\begin{aligned} \frac{ds}{dt} &= \dot{s} \\ \frac{d\dot{s}}{dt} &= \ddot{s} \end{aligned} \tag{2.21}$$

The boundary conditions of these dynamics are

$$\begin{aligned} s_0 = 0, \quad \dot{s}_0 = 0 \\ s_f = 1, \quad \dot{s}_f = 0 \end{aligned} \tag{2.22}$$

and the inputs' limits are still described by Eq. (2.9). According to Eq. (2.18) the cost function is

$$\min_{\ddot{s}(t)} \int_0^{t_f} 1 \, \mathrm{d}t \tag{2.23}$$

The reduced optimization problem has only two states and is therefore easier to solve than the previously formulated example.

2.3 Optimization over a Fixed Interval

Although the number of states in the optimization problem are reduced to p the problem remains hard to solve due to the open final time. The optimization problem in Section 2.2 can be reformulated to carry out the optimization over a fixed interval. Fixed time problems are generally easier

to solve than open time problems. The reformulation is done by interpreting the path parameter s as the time variable and by maximizing the speed $\dot{s}$ in the interval $s_0 \leq s \leq s_f$. Following the reformulation, the dynamic system is further reduced to order $p-1$ but the main advantage is that the optimization problem is formulated as a fixed time problem.

To reformulate the problem, new state variables, $x_1, \ldots, x_{p-1}$, are introduced where

$$x_1 = \frac{\dot{s}^p}{p} \tag{2.24}$$

ů Then the path velocity is expressed by the function

$$\dot{s} = g(x_1) = (px_1)^{\frac{1}{p}} \tag{2.25}$$

The following states are defined by

$$x_k = \frac{dx_{k-1}}{ds}, \quad k = 2, \ldots, p-1 \tag{2.26}$$

After the reformulation, the dynamical constraints of the optimization problem are written as

$$\begin{aligned} \frac{x_1}{ds} &= x_2 \\ \frac{x_2}{ds} &= x_3 \\ &\vdots \\ \frac{x_{p-2}}{ds} &= x_{p-1} \\ \frac{x_{p-1}}{ds} &= u - F_p(x_1, \ldots, x_{p-1}) \end{aligned} \tag{2.27}$$

where $u = s^{(p)}(s)$ is the input to the system. F_p is calculated by differentiating Eq. (2.25) with respect to time, p-times. According to Lemma 3.2 in [8] the p:th derivative of s can be expressed as

$$s^{(p)} = g'(x_1)g(x_1)^{p-1}\frac{dx_{p-1}}{ds} + F_p(x_1, \ldots, x_{p-1}) \tag{2.28}$$

It is also shown in the proof of Lemma 3.2 in [8] that th choice of g as in Eq. (2.25) will give a constant order of the dynamical system.

The equations presented in Section 2.2 will be expressed as functions of s and $x_1, \ldots, x_{p-1}$. The constraints, as described by Eq. (2.15), is therefore

$$g_x(s, x_1, \ldots, x_{p-1}, u) \tag{2.29}$$

and the boundary conditions in Eq. (2.16) will be

$$\begin{aligned} x_1(s_0), \ldots, x_{p-1}(s_0) \\ x_1(s_f), \ldots, x_{p-1}(s_f) \end{aligned} \tag{2.30}$$

The dynamic system is also reformulated to be interpreted as functions of s. The input to the reformulated dynamical system is the p:th derivative of s, $s^{(p)}$.

Reformulating the problem from a problem defined over time to a problem defined over the fixed path s implies a change in the cost function. Minimizing the traversal time t_f is the same as minimizing the inverse of the path velocity, $\frac{1}{\dot{s}}$, which is the same as minimizing the inverse of $g(x_1)$

$$\min_{\tau(t)} \int_{t_0}^{t_f} 1 \, \mathrm{d}t = \min_{s^{(p)}(t)} \int_{s_0}^{s_f} \frac{1}{\dot{s}} \, \mathrm{d}s = \min_{u(s)} \int_{s_0}^{s_f} \frac{1}{\sqrt[p]{(px_1)}} \, \mathrm{d}s \tag{2.31}$$

In order for the numerical solver, IPOPT, to find an optimal solution the occurrence of an initial guess is crucial. This is especially the case if the optimization problem has many dynamical state. Finding an optimal solution for the general minimum time problem requires an initial guess close to the optimal solution. Generating good initial guesses is hard and reformulating the optimization problem to a fixed interval is therefore preferable.

An Example - Fixed in Time

With $p = 2$ we start by rewriting the cost function (2.23) according to Eq. (2.31)

$$\min_{u(s)} \int_{s_0}^{s_f} \frac{1}{\sqrt[2]{(2x_1)}} \, \mathrm{d}s \tag{2.32}$$

x_1 and $g(s)$ are defined according to Eq. (2.24) and Eq. (2.25) as

$$x_1 = \frac{\dot{s}^p}{p} \tag{2.33}$$

and

$$\dot{s} = g(x_1) = (2x_1)^{\frac{1}{2}} \tag{2.34}$$

As p is equal to two $u = \ddot{s}$. To calculate F_2 we differentiate Eq. (2.34)

$$\ddot{s} = \frac{1}{2}(2x_1)^{\frac{1}{2}} 2 \underbrace{\frac{dx_1}{dt}}_{\frac{dx_1}{ds}\frac{ds}{dt}} = (2x_1)^{\frac{1}{2}} \frac{dx_1}{ds}\dot{s} = (2x_1)^{\frac{1}{2}} \frac{dx_1}{ds}(2x_1)^{\frac{1}{2}} = \frac{dx_1}{ds} \tag{2.35}$$

Thus, F_2 is equal to zero and the dynamics of the optimization problem is

$$\frac{dx_1}{ds} = u \tag{2.36}$$

The constraints from Eq. (2.20) are now written as functions of x_1 and u i.e.

$$\begin{bmatrix} 4m_1 \\ m_2 \end{bmatrix} u + \begin{bmatrix} 4d_1 \\ d_2 \end{bmatrix} (2x_1)^{\frac{1}{2}} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{2.37}$$

subject to (2.9). The reformulated boundary conditions are

$$x_1(s_0) = 0, \quad x_1(s_f) = 0 \tag{2.38}$$

Eq. (2.32), Eq. (2.36), Eq. (2.37) and Eq. (2.38) then constitutes an optimization problem of order 1 defined over the interval $s_0 \leq s \leq s_f$.

2.4 Solving the Optimization Problem

As said above in the introduction, the Modelica language will be used to formulate the dynamical constraints while Optimica is used to formulate the rest of the optimization problem. Dymola offers help when formulating Modelica code and provides simulation possibilities. The simulation results have been used as an initial guess for the numerical solver. In this section the work of solving the optimization problem is presented.

Solving an Example - Fixed in Time

To illustrate this work the example above will be used. The optimization problem is defined as:

$$\min_{u(s)} \int_{s_0}^{s_f} \frac{1}{\sqrt[2]{(2x_1)}} \mathrm{d}s \tag{2.39}$$

subject to

$$\frac{dx_1}{ds} = u \tag{2.40}$$

$$x_1(s_0) = 0, \quad x_1(s_f) = 0 \tag{2.41}$$

$$\begin{bmatrix} 4m_1 \\ m_2 \end{bmatrix} u + \begin{bmatrix} 4d_1 \\ d_2 \end{bmatrix} (2x_1)^{\frac{1}{2}} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{2.42}$$

$$-1 \leq \tau_i \leq 1, \quad i = 1, 2 \tag{2.43}$$

The dynamical equations (2.40), the constraints (2.42) and the initial values in (2.41) are expressed in Modelica code as follows

```
model anExampleFixedInTime
  // Process
  processModel pm;
  // Dynamics - Define variables and starting values.
  Real sd(start=0);
  Real x1;
  Real sdd;
  // Path - Define variables for the first and
  // second derivatives of the path.
  Real[2,1] df;
  Real[2,1] ddf;
  // Optimization input - Define an input to the system
  Modelica.Blocks.Interfaces.RealInput u
equation
  // Equations defining the process
  pm.tau1  = pm.M1*ddf[1,1] + pm.D1*df[1,1];
  pm.tau2  = pm.M2*ddf[2,1] + pm.D2*df[2,1];
  // Dynamics
  x1=sd^2/2;
  der(x1) = u;
  sdd = u;
  // Path
  df = [4;1]*sd;
  ddf = [4;1]*sdd;
end anExampleFixedInTime;
```

The process model is defined by the Modelica class processModel

```
model processModel
  // Process definition
  Real tau1;
  Real tau2;
  parameter Real M1 = 1;
  parameter Real M2 = 1.5;
  parameter Real D1 = 0.5;
  parameter Real D2 = 0.3;
end processModel;
```

For an overview of the Modelica language see Fritzson[6]. A richer presentation can be found in [12] or in the language specifications [7]. Due to a bug in the Optimica compiler, `tau1` and `tau2` are defined as scalar variables and not as a vector.

The bounds on the inputs $\tau_{1,2}$, Eq. (2.43), the boundary conditions in (2.41) together with the cost function, (2.39) are expressed using Optimica. For a presentation and manual to Optimica see [4]. The code for the example is presented below

```
class fixedInTimeOpt
  pm.tau1(lowerBound=-1,upperBound=1);
  pm.tau2(lowerBound=-1,upperBound=1);
optimization
  grid(finalTime=fixedFinalTime(finalTime=1),nbrElements=200);
  minimize(lagrangeIntegrand=1/sqrt(2*x1+1e-15));
subject to
  terminal x1 = 0;
  terminal sd = 0;
  terminal sdd = 0;
  sd >= 0;
end fixedInTimeOpt;
```

In addition to this code for generating an initial guess by simulation were also written.

```
model initialGuess
    anExampleFixedInTime ex;
    parameter Real amplitude = 1;
equation
    ex.u = if time >= 0 and time <= 1/2 then amplitude
           else if time >= 1/2 and time <= 1 then -amplitude
           else 0;
end initialGuess;
```

When the Modelica and Optimica code is written and an initial guess has been generated it is time to compile the code. This is done as described in the method section in Chapter 1. Using this problem formulation gives the input sequences as functions of the path parameter s. This is convenient when the optimization results should be used in a PVC. The results from the numerical solver is readable in Dymola for inspection. They can also be imported into MATLAB®using m-functions distributed with Optimica.

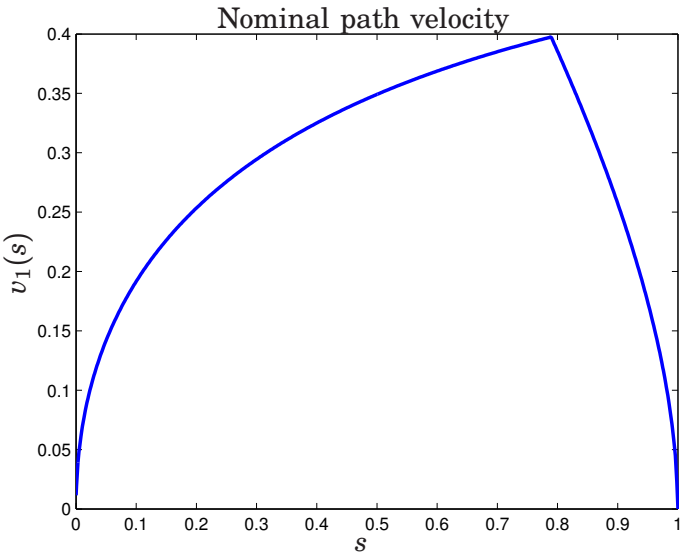

Figure 2.1 The nominal path velocity, v_1, as a function of the path parameter, s.

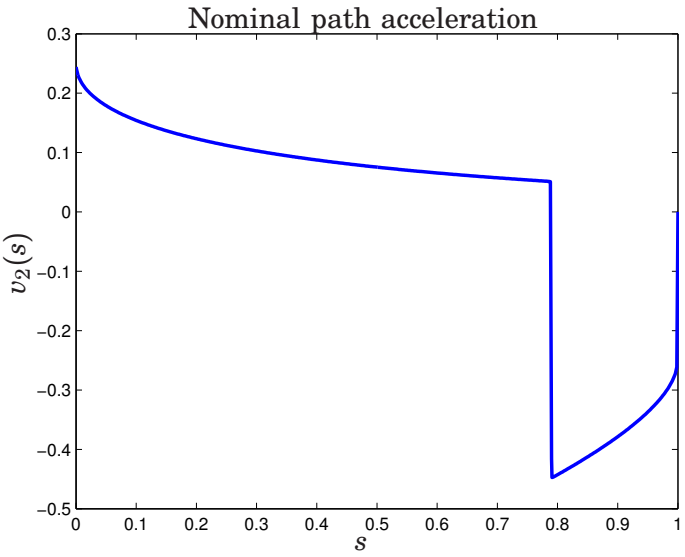

Figure 2.2 The nominal path acceleration, v_2, as a function of the path parameter s.

Using the initial guess generated when simulating the initialGuess model above in the optimization proved to be enough to find a optimal solution. The results are presented in Figure 2.1, 2.2 and 2.3.

It can be seen in Figure 2.3 that τ_1 is at limit at all points along the path and that the solution for this input has bang-bang-characteristics. This is xpected when moving along a straight line. Since, at least, one input is at limit at all parts of the path there does not exist other solutions that have larger acceleration profiles and therefore a shorter traversal time along the path. Any acceleration profile with larger accelerations would give inputs that breaks either the lower or upper boundaries.

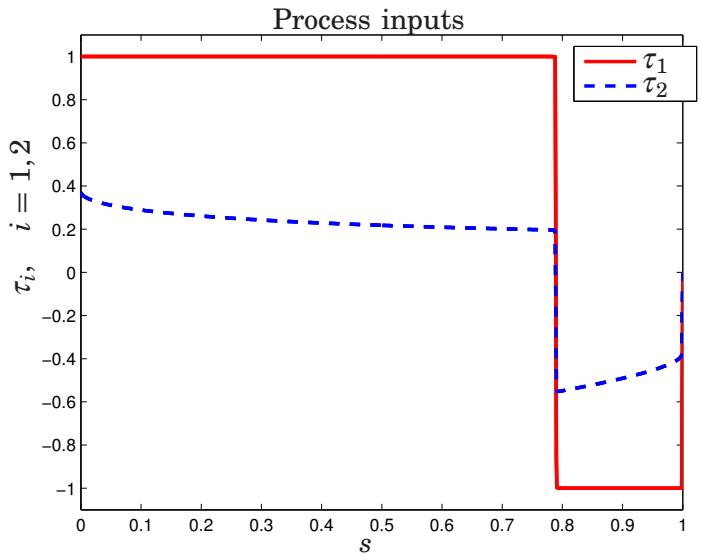

Figure 2.3 The process inputs, τ_1 and τ_2, as functions of the path parameter s.

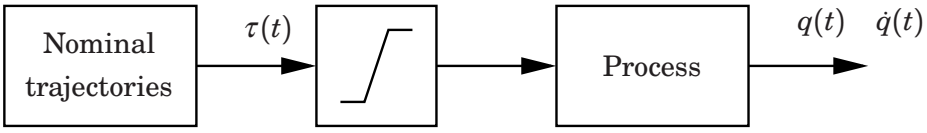

Figure 3.1 Open-loop control using the nominal input trajectories.

3. Path Velocity Controller

This chapter presents the algorithms of the Path Velocity Controller and describes the implementation of the PVC in Simulink®. The algorithms origin from the PhD thesis "Path Constrained Robot Control" by Ola Dahl [8]. The basic idea is to use feedback to modify the acceleration along the path. The algorithms ensure that the input constraints are not violated while still traversing the path in as short time as possible.

3.1 Control Structure

Using the input trajectories $\tau(t)$ obtained for the optimization as inputs to the process is a naïve way of controlling the process, see Figure 3.1. This open loop strategy gives no space to correct for model errors and disturbances. Instead, a strategy to include a controller, making it possible to reject disturbances, could be applied, see Figure 3.2.

There is however a major problem for both of these strategies. Since the inputs are functions of time, the reference will change regardless if the process can follow these reference trajectories or not. This will result in deviations from the path if the process inputs saturate.

A solution to this problem is to use a control structure that modifies the reference trajectory. This control structure is refered to as a PVC and is thoroughly described in Section 3.2. A PVC is used to generate a reference signal to an external controller. The external controller should both consist of feed-forward and a feedback loop. The feed-forward part is used to obtain a fast response to the reference changes. The feedback loop is used to account for disturbances and to ensure robustness. The controller should be designed in a way that guarantees good tracking performance and disturbance rejection[8]. In order to use this approach with a PVC requires that the controller can be written on the form

$$\tau = \beta_1 \ddot{s} + \beta_2 \tag{3.1}$$

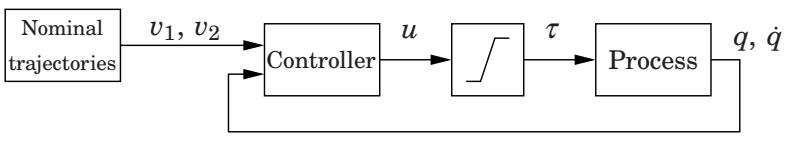

Figure 3.2 Closed loop control using the nominal input trajectories as references.

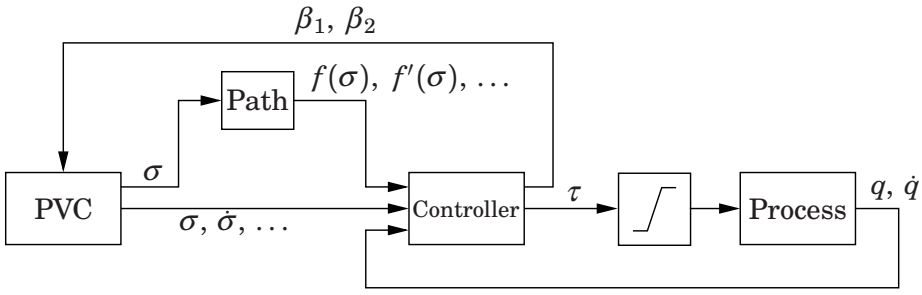

Figure 3.3 Control structure when using a PVC.

3.2 Path Velocity Controller

From the optimization a nominal acceleration profile $v_2(s)$ was acquired. The acceleration profile ensures, under ideal conditions, that all inputs will be within their limits. This profile will therefore be used as a reference acceleration trajectory. Due to model errors and disturbances, using the nominal acceleration will result in input signals, calculated by the controller, which violates the predefined limits (2.2). The path velocity controller limits the acceleration to ensure that the calculated input signals does not violate the limits (2.2). The control structure when including the PVC can be seen in Figure 3.3.

The Basic PVC Algorithm Based on a controller, a first prototype for a PVC algorithm is described in this section. Introduce a new path parameter σ. The first prototype limits the path acceleration, $\ddot{\sigma}$, with respect to the input limits. Writing the controller on the form

$$\tau = \beta_1(\sigma)\ddot{\sigma} + \beta_2(\sigma, \dot{\sigma}, q, \dot{q}) \tag{3.2}$$

, where τ is the input signals to the process, and inserting it in the input limitations (2.2) results in

$$\tau_i^{min} \leq \tau_i = \beta_{1_i}\ddot{\sigma} + \beta_{2_i} \leq \tau_i^{max}, \qquad 1 \leq i \leq n \tag{3.3}$$

where n is the number of inputs. Using Eq. (3.3) it is possible to calculate the lower and upper bounds for the path acceleration, $\ddot{\sigma}$, so that the input limits are not exceeded. For each degree of freedom upper and lower limits can be calculated using the equations below.

$$\begin{aligned}
\ddot{\sigma}^i_{max}(\beta_{1_i}\beta_{2_i}), = \begin{cases} \frac{\tau_i^{max} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} > 0 \\ \frac{\tau_i^{min} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} < 0 \\ \infty, & \beta_{1_i} = 0 \end{cases} \\
\ddot{\sigma}^i_{min}(\beta_{1_i}\beta_{2_i}), = \begin{cases} \frac{\tau_i^{min} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} > 0 \\ \frac{\tau_i^{max} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} < 0 \\ -\infty, & \beta_{1_i} = 0 \end{cases}
\end{aligned} \tag{3.4}$$

Given the upper and lower limit for each input the limits on the path acceleration can be calculated. By taking the smallest $\ddot{\sigma}^i_{max}$ as the upper

bound on the acceleration and the largest $\ddot{\sigma}^i_{min}$ as the lower bound we can be certain that all torques will be admissible.

$$\begin{aligned}\ddot{\sigma}_{min}(\beta_1,\beta_2) &= \max_i \ddot{\sigma}^i_{min}(\beta_{1_i}\beta_{2_i}) \\ \ddot{\sigma}_{max}(\beta_1,\beta_2) &= \min_i \ddot{\sigma}^i_{min}(\beta_{1_i}\beta_{2_i})\end{aligned} \tag{3.5}$$

These bounds can now be used in the first prototype algorithm for the PVC:

$$\begin{aligned}\frac{d\sigma}{dt} &= \dot{\sigma} \\ \frac{d\dot{\sigma}}{dt} &= \ddot{\sigma} \\ \ddot{\sigma} &= \text{sat}(v_2(\sigma), \ddot{\sigma}_{min}(\beta_1,\beta_2), \ddot{\sigma}_{max}(\beta_1,\beta_2))\end{aligned} \tag{3.6}$$

where v_2 is the function defined by Eq. (2.5). Using the algorithm in (3.6) the nominal velocity profile will be followed if $\ddot{\sigma}$ does not saturate. Once $\ddot{\sigma}$ saturates the acceleration will differ from the nominal acceleration given by v_2. If the path acceleration is at limit for a period of time the path velocity will differ from the nominal velocity profile v_1. Since (3.6) is an open-loop control algorithm the deviation from the nominal velocity profile will not be considered. After the period during which the path acceleration was at limit there is the possibility to try to ”catch up” by accelerating more than the nominal acceleration profile prescribes. This can be done by introducing feedback into the algorithm (3.6). It is shown in [8] that σ is a time delayed version of the s if a feedback, which makes $\dot{\sigma}$ converge to $v_1(\sigma)$, is introduced. The feedback used in this thesis is the same as the the one in [8]. Introducing feedback that makes the path velocity converge towards the nominal velocity profile renders the following control algorithm

$$\begin{aligned}\frac{d\sigma}{dt} &= \dot{\sigma} \\ \frac{d\dot{\sigma}}{dt} &= \ddot{\sigma} \\ u &= v_2(\sigma) + \frac{\alpha}{2}(v_1(\sigma)^2 - \dot{\sigma}^2) \\ \ddot{\sigma} &= \text{sat}(u, \ddot{\sigma}_{min}(\beta_1,\beta_2), \ddot{\sigma}_{max}(\beta_1,\beta_2))\end{aligned} \tag{3.7}$$

It can be shown[8] that, assuming $\dot{\sigma} > 0$, $\dot{\sigma} \to v_1(\sigma)$. The assumption that $\dot{\sigma} > 0$ corresponds to a forward motion along the path.

Velocity Profile Scaling The maximum velocity profile is defined by

$$v_{max} = \max_{\dot{s},\ddot{s}} \dot{s} \tag{3.8}$$

subject to the Eq. (2.2). Eq. (3.8) describes the point wise maximum velocity in each point along the path that is admissible according to the constraints in Eq. (2.2) combined with (2.15) [8]. Problems can occur if the velocity profile, $v_1(s)$, obtained from the optimization, at some point is equal to the maximum velocity profile. If the real process differs from the process model a situation where v_1 is larger than the maximum velocity profile for the real process could occur. This could result, e.g., in the velocity,

at one point, being so high that even if minimum acceleration is applied the velocity further along the path will be so high that the process will deviate from the path. The problem is addressed by applying scaling to the velocity profile v_1. Scaling of the velocity profile corresponds to time scaling of the path parameter. To show this a time scaled nominal path parameter $\overline{s} = s(\gamma t)$ is introduced as in [8]

$$\dot{\overline{s}} = \frac{d}{dt}s(\gamma t) = \gamma \dot{s}(\gamma t) = \gamma v_1(s(\gamma t)) = \gamma v_1(\overline{s}(t))$$

To show how the path acceleration is affected by the scaling $\dot{\overline{s}}$ is differentiated.

$$\ddot{\overline{s}} = \frac{d}{dt}\gamma \dot{s}(\gamma t) = \gamma^2 \ddot{s}(\gamma t) = \gamma^2 v_2(s(\gamma t)) = \gamma^2 v_2(\overline{s}(t))$$

Introducing the time scaling in algorithm (3.7) changes u to

$$u = \gamma^2 v_2(\sigma) + \frac{\alpha}{2}(\gamma^2 v_1(\sigma)^2 - \dot{\sigma}^2) \tag{3.9}$$

The implementation used in this thesis uses feedback to change the scaling factor in the same way as in [8]. γ is adjusted so that it goes towards the scaling factor, $\hat{\gamma}$, which results in $\dot{\sigma} = \hat{\gamma} v_1(\sigma)$. γ is updated according to

$$\frac{d\gamma}{d\sigma} = k(\hat{\gamma} - \gamma) \tag{3.10}$$

Combining the chain rule

$$\frac{d\gamma}{dt} = \frac{d\gamma}{d\sigma}\dot{\sigma} \tag{3.11}$$

and the relation

$$\dot{\sigma} = \hat{\gamma} v_1(\sigma) \tag{3.12}$$

the Eq. (3.10) is written as

$$\frac{d\gamma}{dt} = \dot{\sigma} k(\frac{\dot{\sigma}}{v_1(\sigma)} - \gamma) \tag{3.13}$$

To Eq. (3.13) additional logic is introduced. γ is only adjusted when the path acceleration, $\ddot{\sigma}$, is at limit and when the path velocity $\dot{\sigma}$ is lower than γv_1. Furthermore, the algorithm is not used when then velocity profile, $v_1(\sigma)$, is zero since this would render an infinite derivative on γ.

The Final PVC Algorithm The final algorithm including the velocity profile scaling for the PVC is presented below. Figure 3.4 shows the block diagram for the PVC algorithm.

$$\begin{aligned}
\frac{d\sigma}{dt} &= \dot{\sigma} \\
\frac{d\dot{\sigma}}{dt} &= \ddot{\sigma} \\
u &= \gamma^2 v_2(\sigma) + \frac{\alpha}{2}(\gamma^2 v_1(\sigma)^2 - \dot{\sigma}^2) \\
\ddot{\sigma} &= \text{sat}(u, \ddot{\sigma}_{min}(\beta_1, \beta_2), \ddot{\sigma}_{max}(\beta_1, \beta_2)) \\
\dot{\gamma} &= \begin{cases} \dot{\sigma} k(\frac{\dot{\sigma}}{v_1(\sigma)} - \gamma), & \gamma v_1(\sigma) \geq \dot{\sigma} \\ 0, & \gamma v_1(\sigma) < \dot{\sigma} \end{cases}
\end{aligned} \tag{3.14}$$

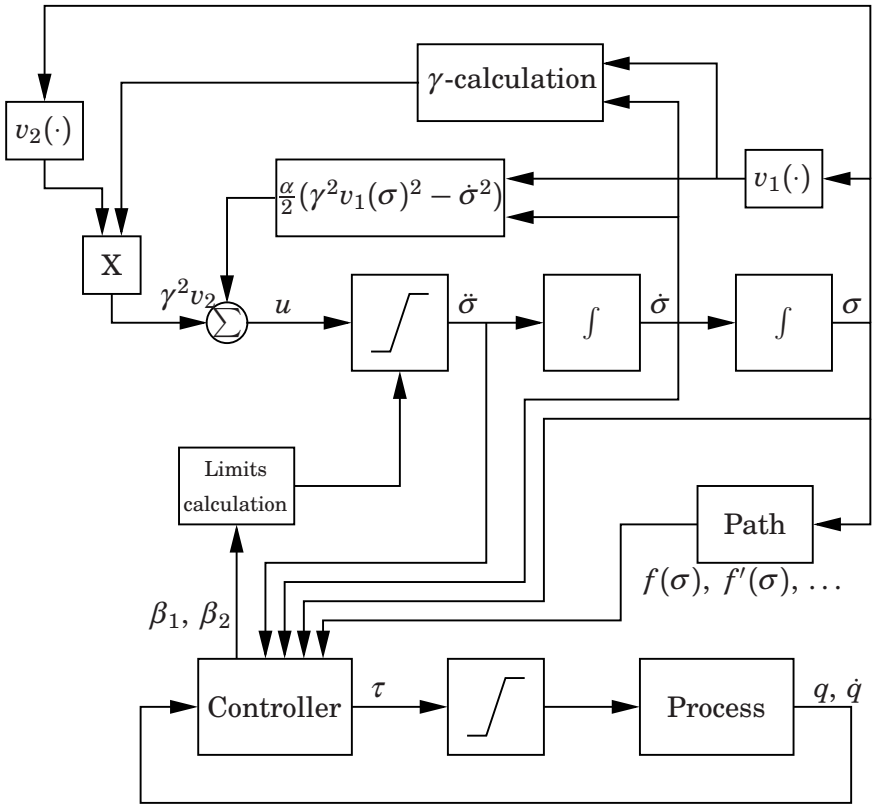

Figure 3.4 Block diagram for the final PVC algorithm presented in Section 3.2.

4. Case Study

In this chapter a PVC, according to Algorithm (3.14), is implemented and tested, both in simulations and on a real process, the ABB IRB140B. The chapter starts with a section regarding the system identification of the ABB robot. Following this section is a section about path recording using ABB's lead-through system. The time minimum optimization problem is then formulated in Modelica and Optimica. The problem is solved and the optimization results are used in the PVC, which is implemented in Simulink®. The PVC is tested in simulations before it is used on the real robot system.

4.1 Model Identification

Ideally, the control structure and the controller parameters of the ABB robot system should be known. If that was the case, a model from motor torque to angular position could have been identified and used together with the ABB control structure. Unfortunately, and understandably, this information is not shared by ABB. Therefore, the model and control structure used in this thesis is somewhat forced by the possibilities offered by the available robot interface.

It was decided that the ABB position control loop should be turned off and that the velocity reference is to be considered as the system input. Note that the ABB velocity control loop is left unchanged. The output is considered to be the motor angular position. During the system identification, the input was the velocity reference and the output was the velocity measurements. An integrator has been added to the identified model in order to obtain the angular position rather than the angular velocity.

Identification Procedure

The identification has been performed joint-wise and the model does not include any cross couplings between the joints. When including the velocity control loop the input-output behaviors are alike for all six joints. Deriving an extensive model for the ABB robot is not in the scope for this thesis, therefore finding simple linear models that describe the process "good enough" have been identified. As always when doing system identification the question of how to define "good enough" arises. Since the processes to identify already include a well tuned controller, the input-output relation is nice in the sense that it does appear to be linear.

The input signal used is a square wave that changes sign depending on the sign of a random sequence. How often the square wave could change sign was set by a parameter in the Simulink® model as well as the amplitude of the wave. The amplitude was chosen as large as possible for each joint. Consideration was taken when choosing amplitude so that the robot safety system did not execute an emergency stop and so that the physical angular limitations was not exceeded.

The sample time when using the robot interface is fixed at $h = 0.004\,\text{s}$. Measurements are retrieved using an available log program that logs specified signals. The measurement logs are converted to textual format which is

i	a_i	b_i
0	$1.41 \cdot 10^{10}$	$1.41 \cdot 10^{10}$
1	$4.96 \cdot 10^{5}$	$1.55 \cdot 10^{5}$
2	$1.11 \cdot 10^{8}$	$-9.73 \cdot 10^{7}$
3	532.6	-653.6
4	1	0

Table 4.1 ARMAX model parameters for joint 1

readable using the MATLAB®function `readlog`. The data was then divided into two equal parts, one used for identification, the other for validation.

Two different models have been identified, using the MATLAB® System Identification Toolbox, for each joint, a discrete-time ARMAX model and a continuous-time process model. These two models are described below.

ARMAX-model

ARMAX-models have a structure according to Eq. (4.1).

$$A(z^{-1})y_k = z^{-d}B(z^{-1})u_k + C(z^{-1})w_k \tag{4.1}$$

where d is a delay, A, B and C are polynomials according to (4.2)

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + \ldots + a_{n_A} z^{-n_A} \\ B(z^{-1}) &= b_0 + b_1 z^{-1} + \ldots + b_{n_B} z^{-n_B} \\ C(z^{-1}) &= 1 + c_1 z^{-1} + \ldots + c_{n_C} z^{-n_C} \end{aligned} \tag{4.2}$$

w_k is a white noise stochastic process with zero mean, $\mathcal{E}\{w_k\} = 0$, and some covariance $\mathcal{E}\{w_k w_i\} = \delta_{ij}\sigma_w^2$, [14].

The polynomial orders and delay, n_A, n_B and n_C and d, in Eq. (4.2) are determined by extensive testing. A great number of different models have been tried before the decided model structure was chosen. The goal was to find a model of relativly low order that still gave small residuals and a good model data fit when doing cross-validation. The discrete ARMAX models are converted to continuous models using the MATLAB® `d2c` command with the zero-order method [15].

The model structure that gave the best models for all six joints with a relativly low model order was $n_A = 4$, $n_B = 4$, $n_C = 4$ and $k = 2$. Hence, the model structure in continuous time is described by the transfer function (4.3).

$$G(s) = \frac{b_3 s^3 + b_2 s^2 + b_1 s + b_0}{s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0} \tag{4.3}$$

In Table 4.1, the identification result for joint 1 will be presented. Figure 4.1 and Figure 4.2 shows the model fit and residuals from the cross-validation for joint 1. For figures and data regarding joint two to six refer to Appendix A.

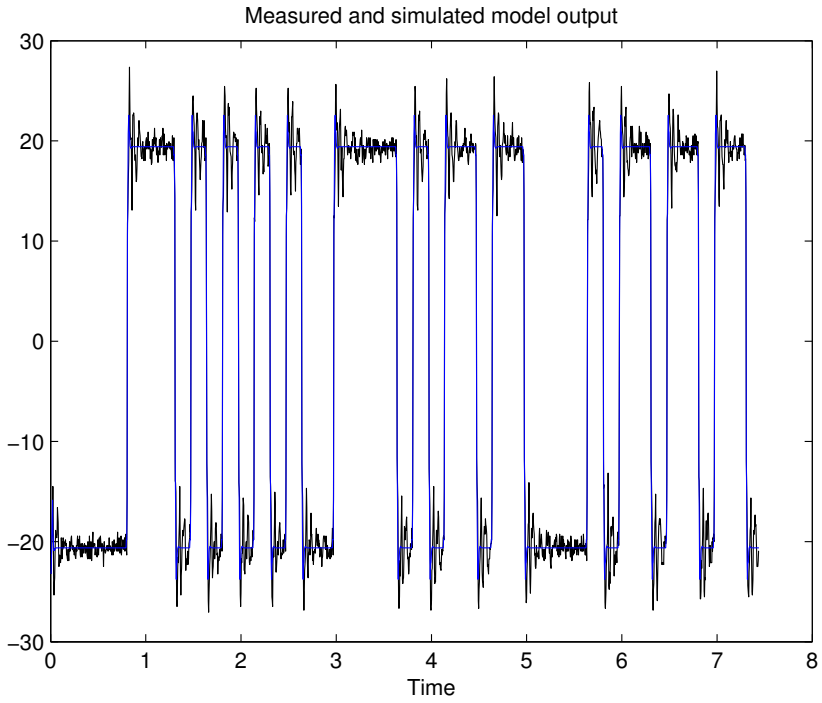

Figure 4.1 Fit between ARMAX-model and cross-validation data for joint 1

Process-model

The Simulink® Identification Toolbox has also been used to obtain continuous process models with a predefined structure. Tests with different model structures have been performed. The Identification Toolbox allows for continuous process models up to an order of three. It can also include an extra integrator, a zero and a time delay. All these different choices have been tested but the model described by (4.4) was finally chosen to represent this category of identified models.

$$G_i(s) = \frac{K_i}{T_i s + 1}, \qquad 1 \leq i \leq 6 \tag{4.4}$$

The identified parameters are seen in Table 4.1 and the model fit and residuals when doing cross validation are seen in Figure 4.3 and Figure 4.4. Both the residuals and the model fit indicates that there are more information in the data that the model does not utilize.

Model Used for Optimization

For optimization purposes the process model (4.4) together with an integrator will be used. The resulting transfer function is

$$G_i(s) = \frac{K_i}{T_i s + 1}\frac{1}{s}, \qquad 1 \leq i \leq 6 \tag{4.5}$$

with parameters K_i and T_i according to table 4.1. These models are chosen in favour for the higher-order ARMAX models due to the fact that only the two first derivatives of the path are required. The residuals for the

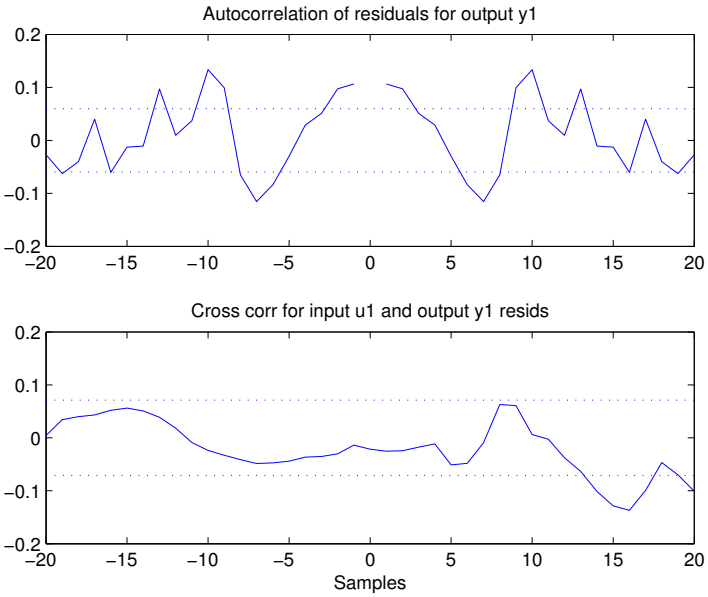

Figure 4.2 Auto correlation and cross-correlation for the residuals for the ARMAX-model

Joint	K_i	$T_i \times 10^2$
1	1.031	1.907
2	1.077	2.043
3	1.061	1.913
4	1.051	1.716
5	1.062	1.791
6	1.062	1.745

Table 4.2 Model parameters for the identified process models

ARMAX models are smaller and the model fit is better compared to the process models. Hence, using the ARMAX models would probably render better results since it better describes the dynamics, eg., resonances etc., in the robot joints. Since the ARMAX models are of higher order it will probably be harder to solve the optimization problems, using these models.

4.2 Path Recording

The paths can, as in the examples in Chapter 2, be constructed using mathematical expressions. In the case of robot programming, lead-through is a more versatile way to construct paths. Lead-through is a force-control mode which allows the operator to freely move/lead the robot in the workspace by hold ing the end effector. The joint angles are recorded while an operator moves the robot around. Some manual manipulation of the recorded path can be required. This includes editing the length of the recorded track so

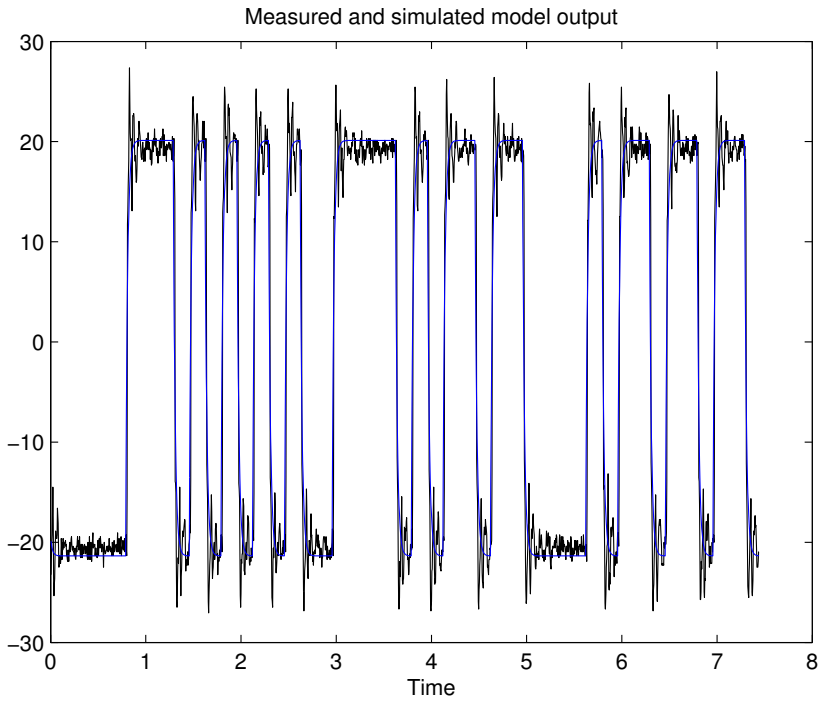

Figure 4.3 Fit between process model and cross-validation data

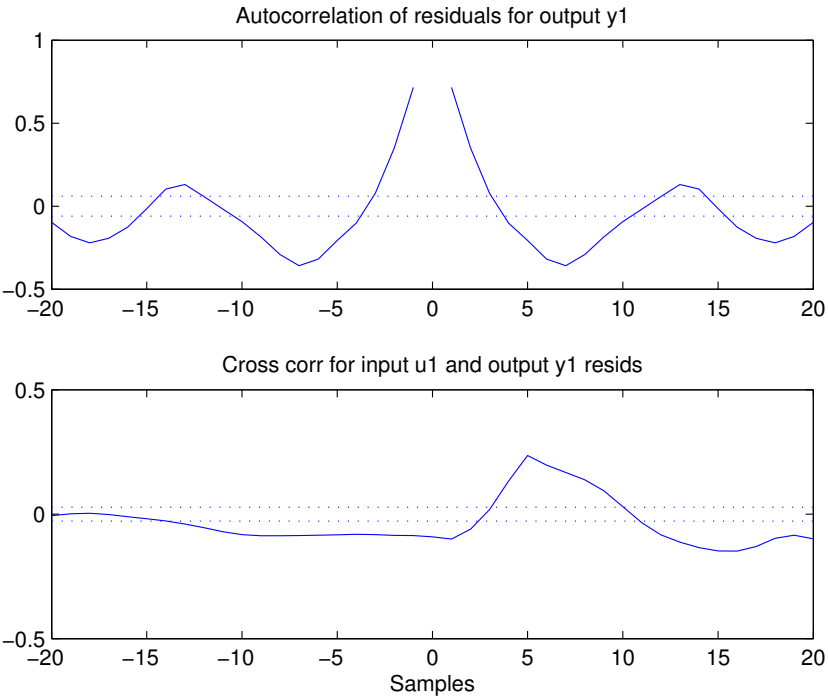

Figure 4.4 Auto correlation and cross-correlation for the residuals

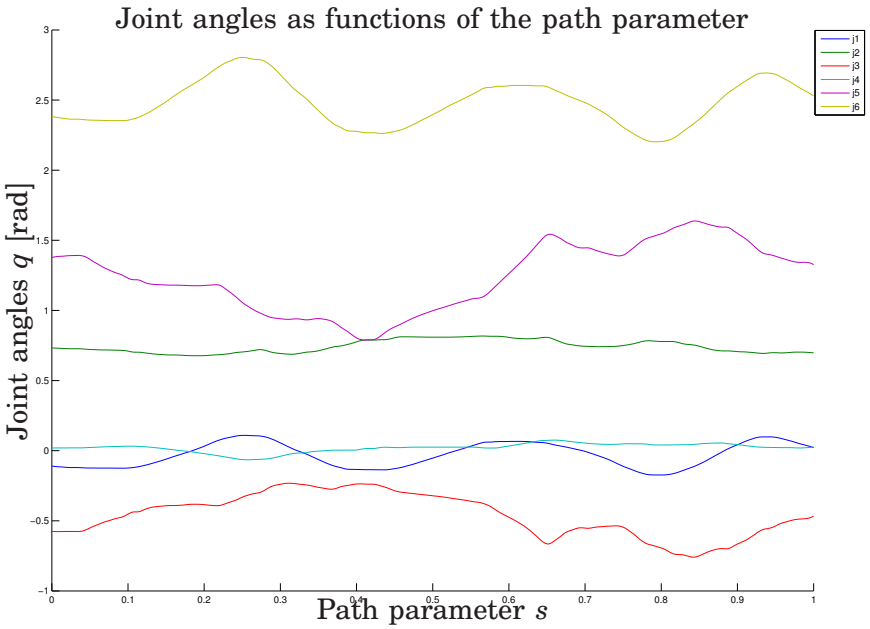

Figure 4.5 The path represented as joint angles

that it represent the desired motion. When the joint angles are available as vectors these are parametrized as function of a path parameter, s, defined on the interval $s_0 = 0 \leq s \leq s_f = 1$. The choice of s_0 and s_f are not that important as long as $s_0 < s_f$.

When the path is parametrized, Modelica splines can be created using the MATLAB® function `gen_modelica_spline` that is distributed along with Optimica 0.3. The derivatives of the path, $f'(s), f''(s), \ldots$ are calculated in the Modelica model using the `der()`-operator. The `der()`-operator differentiates a variable with respect to the Modelica variable `time`. Using the reformulated optimization problem formulation in Section 2.3, `time` will be equal to s and the `der()`-operator can be regarded as if it differentiates with respect to s.

Using all the points from the recording when generating the splines is unnecessary since it will create a large amount of splines that does not contribute considerably to a more accurate optimal solution. When creating splines with `gen_modelica_spline` an appropriate scale factor has to be chosen. The scale factors determines how fine or coarse the splines should be. The scale factor should be chosen small enough so that the misfit between the original path and the splines are small but large enough so that the resulting file containing the splines is not too large. Fine splines makes it harder to find a good optimal solution as well as it makes the optimization sloving take longer time. In the path used in this chapter the scaling factor was chosen to be 30. Figure 4.5 show the path represented as joint angles. Figure 4.6 shows the path in a Cartesian coordinate system with origin at the robot base.

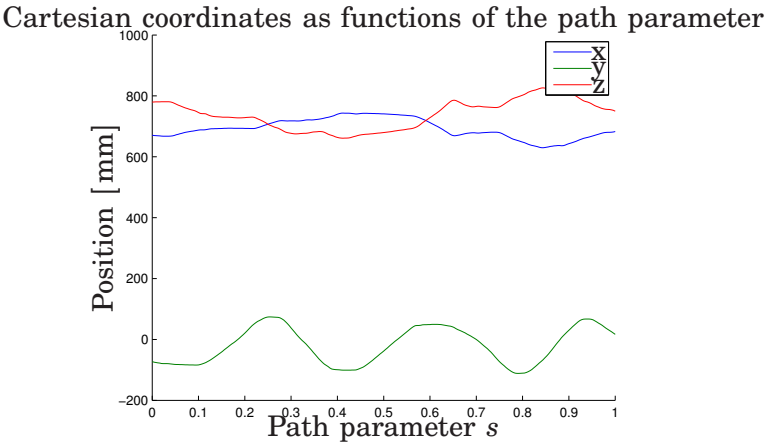

Figure 4.6 The resulting Cartesian coordinates for the tool center point along the path

4.3 Optimization

The time minimum optimization problem is formulated over a fixed time interval as in Section 2.3. When solving for the minimum time solution, the results are often a bit spiky. Therefore a second optimization is done in order to get smoother profiles.

In the first optimization, the time t_f, i.e., the time it takes to traverse the path is obtained. In the second optimization, a terminal constraint is set to make the traversal time one or two percent longer than the minimum time. The cost function is changed so that it should minimize the derivative of the input signals τ_i and the nominal path acceleration v_2. This makes the acceleration and velocity profiles smoother.

The optimization is formulated using the model from Section 4.1 and the path from Section 4.2. The time minimum optimization problem is formulated in Section 4.3 and 4.3.

The input boundaries are chosen to be five percent of the maximum velocities for each joint specified by ABB, [1]. This is due to the fact that the fixed sample time in the robot system is to large to accuratly perfom the integration needed in the PVC. By limiting the inputs, the traverasal time is increased and therefore the numerical errors from the integrations decreased.

Formulation of the Model in Modelica

The Modelica part of the problem formulation is stored in a Modelica package. This package includes two Modelica models and one Modelica class that holds different parts of the optimization problem. The Modelica files are used in both optimizations.

The main Modelica model is *Optimization*. It declares all the used variables, the robot model, the derivatives of the path and the system dynamics etc. Commented code for the model can be seen in Appendix D.1.

The Modelica class, *Splines* contains the splined path $f(s)$ implemented as if-clausules. Because of the implementation using if-clausules the class is large and it will therefore not be fully presented here. The structure of *Splines* can be seen in Appendix D.2.

The InitialGuess model is used, as is suggested by the name, to generate an initial guess for the Optimica compiler. The InitalGuess model contains an Optimization object and a predefined sequence connected to the Optimization object's input, u. This file, when run in Dymola's simulation environment, produces an initial guess in textual format that is readable to the Optimica compiler.

Formulating the Problem in Optimica

As mentioned above, the optimization is done in two steps and therefore two separate Optimica files has been written. Some things are alike in the two optimization problem formulations. For instance the input constraints, i.e., the upper and lower bounds on the inputs, are described as

```
tau1(lowerBound=-3.49*0.05, upperBound=3.49*0.05);
```

Here, as an example, only the bounds on joint one are displayed.

The grid over which the optimization is calculated is described by

```
grid(finalTime = fixedFinalTime(finalTime=1)
                 ,nbrElements=200);
```

The optimization is performed over the interval $0 \leq s \leq 1$ which is specified by setting `finalTime=1`. The parameter `nbrElements=200` specifies that the optimization is calculated using 200 numerical elements. The more elements used, the longer it takes to solve the optimization problem but it also offers the possibility for more exact solutions. It is hard to say how many elements to use when solving these kinds of problems. After a while you get a feeling for how many elements to use in order to obtain a optimal solution but until this feeling appear you are down to trial and error.

Both Optimica files also contains terminal constaints that are alike in both optimization problems. These are

```
terminal x1 = 0;
terminal sd = 0;
sd >= 0;
```

The first and second terminal constraints specifies that the velocity at the end of the path should be zero. The third and last constraint specifies that the velocity should be larger or equal to zero during the traversal of the path.

The two Optimica files has resemblances, as described above, but they do differ.

Finding the Minimum Traversal Time The Optimica file MinTimeOpt.op is used to find the time it takes to traverse that path, t_f. t_f is found by minimizing the cost function described in Eq. (2.31). The cost function is described in Optimica as

```
minimize(lagrangeIntegrand=1/sqrt(2*x1+1e-10));
```

Note that a small value has been added to `2*x1`. This is to avoid division by zero in the start and the end of the path were the speed along the path is zero.

This problem was solved in the way described in the Chapter 1. The file MinTimeOpt.op can be found in Appendix D.2. Even though the initial guess given to the Optimica compiler was not close to the optimal solution,

the problem could be solved without problems. The time it took to solve an optimization problem was approximately one minute.

The path acceleration and velocity profiles can be seen in Figure 4.7 and 4.8. In Figure 4.9, which shows all input trajectories normalized with respect to their limits, one can see that at least one input is at limit, i.e., -1 or 1, for the whole path.

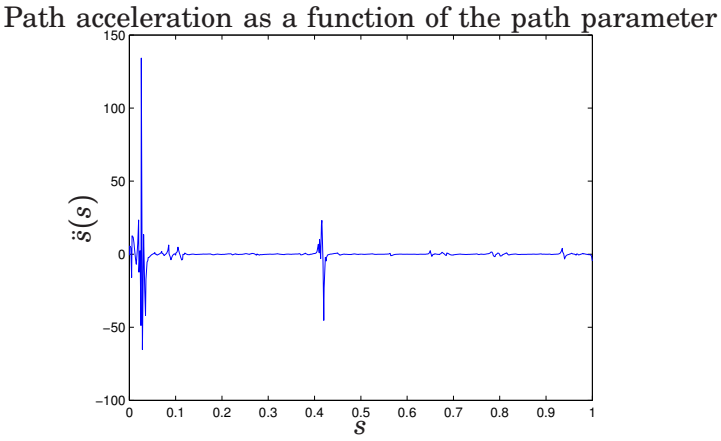

Figure 4.7 Nominal path acceleration profile obtain from the first optimization run.

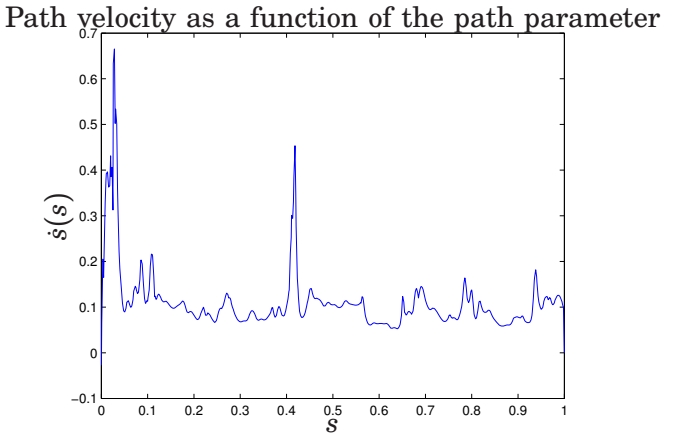

Figure 4.8 Nominal path velocity profile obtain from the first optimization run.

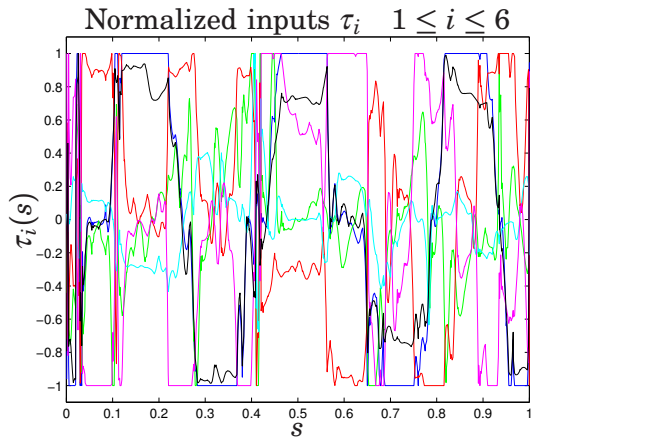

Figure 4.9 Optimal normalized input sequences τ from the first optimization run.

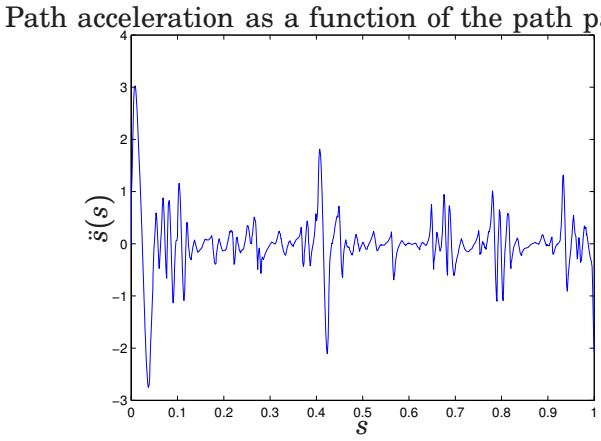

Figure 4.10 Nominal path acceleration profile obtain from the second optimization run.

Finding Nominal Trajectories To obtain smoother profiles a second optimization was done. The result from the first optimization was used as an initial guess. In this second optimization the cost function was changed to

```
minimize(lagrangeIntegrand=der(sdd)^2 + der(tau1)^2
          + der(tau2)^2 + der(tau3)^2 + der(tau4)^2
          + der(tau5)^2 + der(tau6)^2);
```

A new terminal constraint was also added to the problem formulation

```
terminal tf = 10.653*1.02;
```

If the time for traversal found in the first optimization was $t_f = 10.653$ the new time for traversal is now two percent more. This means that the path will not be traversed in minimum time, however the nominal acceleration and velocity profiles together with the input sequences are smoother. Hence, this is a trade-off between a minimum time traversal of the path and the wear on the motors. The Optimica code for the second optimization problem can be seen in Appendix D.2. Figure 4.10 and 4.11. In Figure 4.12 the normalized input sequences are displayed.

By comparing Figure 4.7 to Figure 4.10, a difference in the acceleration magnitude can be seen. This results in smoother path velocity profiles which can be seen in Figure 4.8 and Figure 4.11 The not so spiky and smoother profiles will work better in the PVC because of the numerical integration. Since the integrators have a fixed sample time of 0.004 s, the integration of a spiky nominal profile is more prone to numerical problems. Comparing Figure 4.9 and Figure 4.12 we can see that the input sequences are almost the same, which is to be expected since the traversal time is just two percent longer.

What to do if no Optimal Solution is found Besides making the problem easier to solve because of the fewer dynamical states, reformulating the optimization problem to a fixed interval was another advantage as well. If the solver fails to find an optimal solution the problem can be solved in steps.

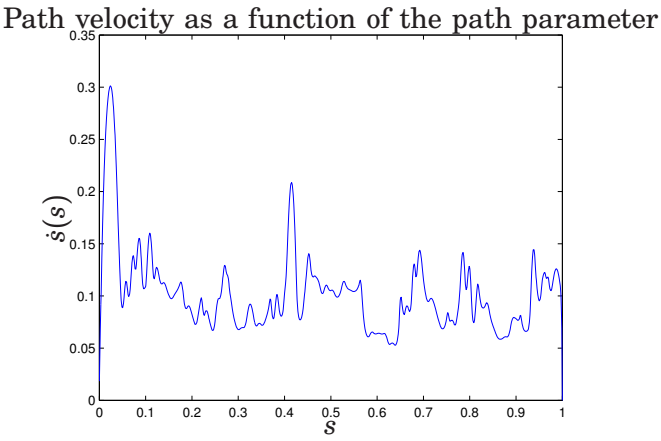

Figure 4.11 Nominal path velocity profile obtain from the second optimization run.

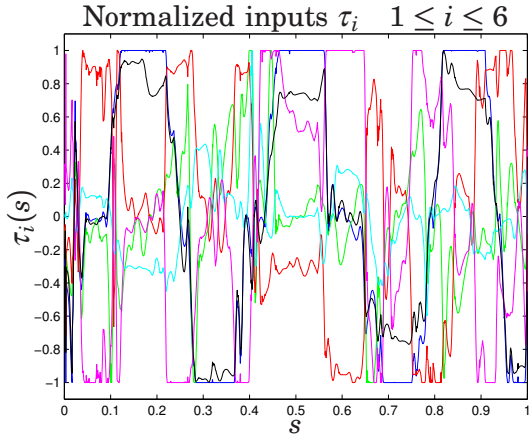

Figure 4.12 Optimal normalized input sequences τ from the second optimization run

By reducing the interval, i.e., changing s_f for $s_{temp} < s_f$ in the cost function (2.31), over which the optimization problem is solved, an optimal solution can be obtained for the interval $s_0 \leq s \leq s_{temp}$ [9]. The optimal solution obtained can then be used as an initial guess when solving the optimization problem with a larger s_{temp}. If an optimal solution is found, s_{temp} is increased and the procedure is repeated. If an optimal solution is not found s_{temp} has to be decreased and the optimization has to be run again. By repeating this procedure, and gradually increasing the optimization interval until $s_{temp} = s_f$, an optimal solution for the whole path can be found.

4.4 PVC in Simulink®

In this section, the control structure described in Chapter 3 will be implemented using Simulink®. The PVC will be used for both simulations, Section 4.6, and for experiments on the ABB IRB140B, Section 4.9. The robot

interface and its connection to the PVC is described in Section 4.8. The implementation of the PVC algorithm is described in Section 4.4. Section 4.4 also describes the controller used with the PVC and its implementation.

The control structure is the same as discussed in Section 3.1, Figure 3.3. Below, the implementation of the Controller, the PVC and the Path blocks shown in Figure 3.3, is described.

Controller

The model (4.5) consists of six, $n = 6$, second order systems, $p = 2$. The feed-forward part was chosen as

$$\tau = K^{-1}T\ddot{q}_{ref} + K^{-1}\dot{q}_{ref} \tag{4.6}$$

with K and T as diagonal matrices. The diagonal elements are chosen as in Table 4.1 to match the system dynamics so that good reference trajectory tracking is obtained. The feedback part consists of six parallel PID controllers on the form

$$v_i = K_i^P(q_i^{ref} - q_i) + \frac{1}{K_i^I}\int(q_i^{ref} - q_i) + K_i^V(\dot{q}_i^{ref} - \dot{q}_i) \tag{4.7}$$

Due to the fact that all six models are very alike, the parameters K_i^P, K_i^I and K_i^V have been chosen as scalars, i.e., the value of each parameter is the same for all six joints. Combining Eq. (4.6) and Eq. (4.7) gives the controller

$$\tau = K^{-1}T\ddot{q}_{ref} + K^{-1}\dot{q}_{ref} + K_i^P(q_i^{ref} - q_i) + \frac{1}{K_i^I}\int(q_i^{ref} - q_i + K_i^V(\dot{q}_i^{ref} - \dot{q}_i)) \tag{4.8}$$

Setting $q = f(s)$, differentiating this expression according to the chain rule (2.13) and writing the result on the form

$$\tau = \beta_1(\sigma)\ddot{\sigma} + \beta_2(\sigma, \dot{\sigma}, q, \dot{q}) \tag{4.9}$$

gives

$$\begin{aligned}\beta_1(\sigma) &= K^{-1}Tf'(\sigma)\\ \beta_2(\sigma, \dot{\sigma}, q, \dot{q}) &= K^{-1}(Tf''(\sigma)\dot{\sigma}^2 + f'(\sigma)\dot{\sigma}) + \\ &\quad K^V(f(\sigma)\dot{\sigma} - \dot{q}) + K^I\int((f(\sigma) - q) + K^P(f(\sigma) - q))\end{aligned} \tag{4.10}$$

This controller is straightforward to implement in Simulink and the block diagram can be found in Appendix C, Figure C.3.

PVC

During the implementation of the PVC it has proved efficient to use *Embedded MATLAB function* blocks. Embedded MATLAB functions support a subset of MATLAB commands and is therefore a convenient way of implementing certain parts of the PVC.

The PVC Simulink® model can be found in Appendix C, Figure C.4. The core of the PVC is the look up-tables containing the nominal acceleration

and velocity profiles. These are called *v1* and *v2* respectively in Figure C.4. The model contains a parameter called *f_switch* that is used to switch the PVC on and off, where *f_switch* equal to one corresponds to on and *f_switch* equal to zero corresponds to off. The *v2* signal is multiplied by γ^2 and the result is added to the i_{fb} signal. i_{fb} is short for *Internal FeedBack* and it is implemented according to Eq. (3.9), i.e.

$$i_{fb} = \frac{\alpha}{2}(\gamma^2 v_1 - \dot{\sigma}^2)$$

The implementation in Simulink® is straightforward and is shown in Figure C.5. The signal u, calculated according to Eq. (3.9) is the input to the *Saturation* block. All the internal blocks in the PVC model, including the *Saturation* block, are described below. The output form the *Saturation* block goes through the *ON/OFF Logic* block, see the below section. The output from this first *ON/OFF Logic* block is the path acceleration *sigmadd*, $\ddot{\sigma}$. $\ddot{\sigma}$ is sent as an output from the PVC block as well as integrated by a resetable discrete integrator. The integrator, as well as all integrators used in this implementation, uses the forward Euler method for integration and has a sample time of $h = 0.004$ s. The result of the integration is *sigmad*, $\dot{\sigma}$. $\dot{\sigma}$ is also sent as an output from the PVC as well as integrated and sent trough the *Stopper* block. The output signal from the *Stopper* block is *sigma*, σ which drives the look up-tables *v1* and *v2*.

Throughout the implementation some blocks have constants as inputs due to the fact that global parameters cannot be set or changed inside an *Embedded MATLAB function*. In order to conveniently change the parameters between simulations and during the robotic experiments the solution to implement them as constant inputs to the concerned functions was chosen.

Saturation The *Saturation* block is implemented as an *Embedded MATLAB function* and has three inputs: an upper limit, a lower limit and u. The block limits u between the upper and lower bounds just like a regular saturation block in Simulink® but has some special features. If the lower bound is greater than the upper bound, the saturation block output is equal to the block input i.e. $y = u$. The block also sets the output *LimitsActive* to one if the lower bound is lesser than the upper bound and if u is not between the upper and lower bounds. Otherwise the *LimitsActive* is set to zero. This is used by the function that calculates γ. The MATLAB® code for the saturation block is found in Appendix D.6.

Limits calculation The limits that serves as inputs for *Saturation* are calculated by the *Embedded MATLAB functionpathAccLim*. Inputs to the function are the vectors *beta1*, *beta2*, *tauMax*, *tauMin*. *tauMax* and *tauMin* are constant inputs for the reason described above. *beta1* and *beta2* are values calculated by the controller, Section 4.4, according to Eq. (4.10). The block calculates lower and upper bounds for all axes according to Eq. (3.4). The block output is then chosen according to Eq. (3.5). To avoid problems when translating the Simulink® model to C-code in order to run it on the robot the infinity limits have been replaced by large numbers, in this case 10^6. The MATLAB® code for the *pathAccLim* block is presented in Appendix D.5.

Velocity Profile Scaling The block responsible for the velocity profile scaling, *Gamma Calculation*, can be seen in Appendix C, Figure C.6. The block consists of an *Embedded MATLAB function* and a resettable integrator. The time derivative of γ, $\dot{\gamma}$ is calculated according to Eq. (3.13) and the additional logic following Eq. (3.13). The calculation of $\dot{\gamma}$ is done in an *Embedded MATLAB function*. The code for this block is available in Appendix D.7. The *Embedded MATLAB function* is followed by an integrator in order to obtain γ. This integrator differs only from the other integrators by having the output starting value equal to one, $\gamma(t = 0) = 1$, corresponding to no scaling.

Stopper The *Stopper* block has been implemented to ensure that σ never gets larger than than the maximum s, s_f. When σ is close to s_f, the stopper value becomes larger than zero and the switch threshold is exceeded making the *switch* block switch input. When the treshold is exceeded σ is set to s_f. The stopper signal affects the *On / Off Logic*, see below. The *Stopper* block also includes a *Min Max Running Resettable* Simulink® block to ensure that σ never decreases.

On/Off logic The *On / Off Logic* block has two functions, turning the PVC on and off. The PVC is turned on when *f_switch* is equal to one. Turned on meaning that the input signal is equal to the output signal. The PVC is turned off when *f_switch* is zero and when *stopper* is larger than zero, i.e., σ is equal to s_f. The Simulink® model can be found in the Appendix C.

Path

The path and its two derivatives, with respect to the path parameter, are stored in look-up tables. These tables are interpolated between the specified points in the tables. If the table input is outside the specified interval the table output will hold the specified output value for interval. A Simulink® block diagram for the path can be found in Appendix C. In the Simulink implementation of the PVC, I have chosen to locate the ”Path block” outside the PVC block for convenience and readability.

Implementation Issues

During the implementation and testing of the PVC some problems arose. When σ got close to s_f it began to decrease instead of settling at $\sigma = s_f$. The path, which is recorded point wise, is represented using splines during the optimization and by interpolated values originating from look up tables in the PVC. This, together with the numerical integration of $\ddot{\sigma}$ and $\dot{\sigma}$, could be the reason for the numerical problems encountered. In order to handle this problem the *Min Max Running Resettable* Simulink® block was inserted as well as the parameter $s_{f\epsilon}$, *sfeps*. The *Min Max Running Resettable* Simulink® block prevents σ from decreasing and *sfeps* determines how close to s_f the path parameter σ should be before the *Stopper* block sets σ to s_f. Using the *Min Max Running Resettable* Simulink® block can be motivated by the assumption, $\dot{\sigma} \geq 0$, that motion only should take place in the forward direction.

4.5 Controller Parameters

The controller has been tuned manually to obtain good reference tracking. Tests were done both in simulations and on the real robot. Since the models for all six joints are alike the control parameters are also alike. Both $K^P = 20$ and $K^I = 1$ are chosen identical for all six joints. For the parameter K^V the value 0.5 was chosen at first. This value worked well for joint one, two and three whereas for joint four, five and six small oscillations occurred. New test were performed on joint four to six with $K^V = 0.3$ which was good in terms of tracking and did not lead to any oscillations.

4.6 Simulation

The implemented PVC has been used in simulations with the recorded path from Section 4.2 and the nominal trajectories obtained in Section 4.3. In this section the effect of the internal feedback and the velocity profile scaling will be investigated. Furthermore, a test including a model error will be performed. Finally, the effect of faster sampling will be investigated. There are of course an almost infinite number of different simulations that could be done and an equally large number of signals to plot. The results of the simulation displayed in this section are the ones that are believed to be of greatest importance.

Different Parameters

Three simulations were done using different parameters α and k, see Section 3.2. The sample time $h = 0.004$ was chosen as the sample time that is used in the ABB robot system.

Simulation 1 The first simulation was done with $\alpha = 0$, i.e., no internal feedback, and $k = 0$, i.e., no velocity profile scaling. Without the internal feedback there can be no recovery in speed if the system is disturbed. Not using the velocity profile scaling increases the risk of touching the maximum velocity profile if model errors are present. Velocity profile scaling is discussed in Section 3.2. The traversal time for this simulation was $t_f = 11.19\,\text{s}$ to be compared with the optimal time $t_{f_{opt}} = 10.87\,\text{s}$.

Small deviations from the nominal acceleration profile can be found in Figure 4.13. These deviations, together with numerical errors from the numerical integration, are the source of the deviation in the nominal velocity profile in Figure 4.14. Since no internal feedback is used the path velocity does not approach the nominal velocity. The fact that the deviations depend on the numerical integrations will be investigated in Simulation 5.

Figure 4.15 show the error in the x-, y- and z- directions for the tool center point, *TCP*, in millimeters. The error is never larger than 0.31 mm in each direction at any part of the path.

Figure 4.16 shows the input sequences for all six robot joints along with their limits. The inputs are not at limit during the the whole traversal of the path. This is due to the deviation from the nominal trajectories and the fact that there is no internal feedback that makes the path velocity approach the nominal velocity.

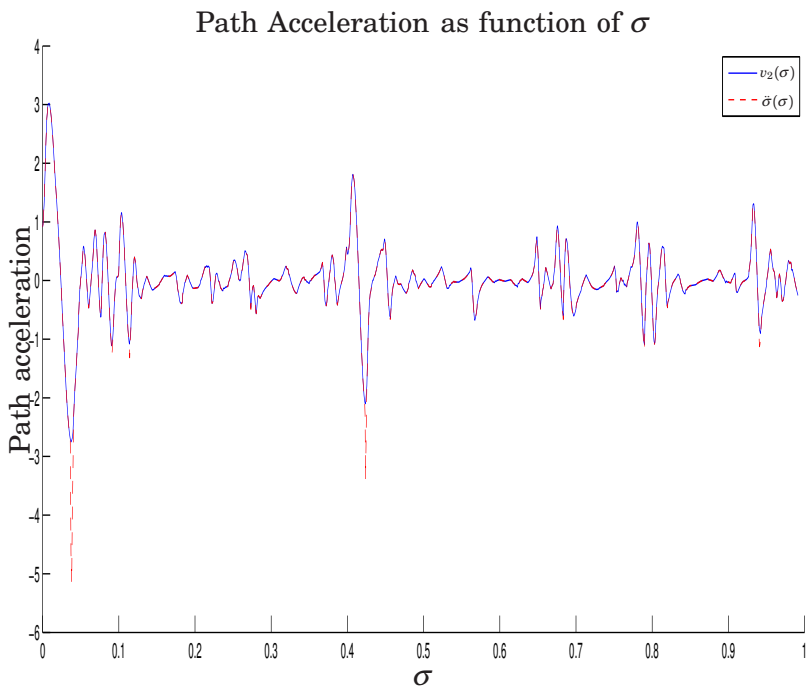

Figure 4.13 Nominal and obtained acceleration profiles. Simulation 1.

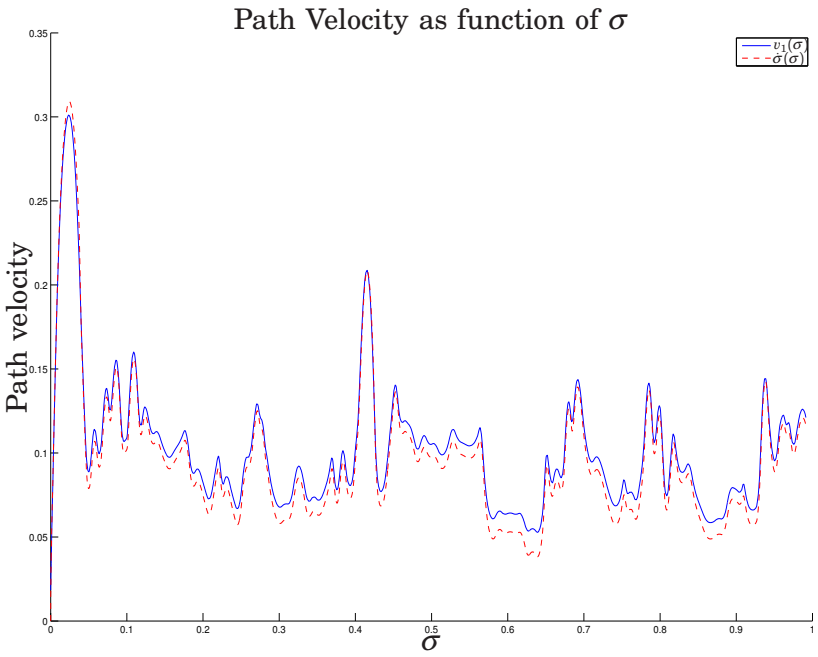

Figure 4.14 Nominal and obtained velocity profiles. Simulation 1.

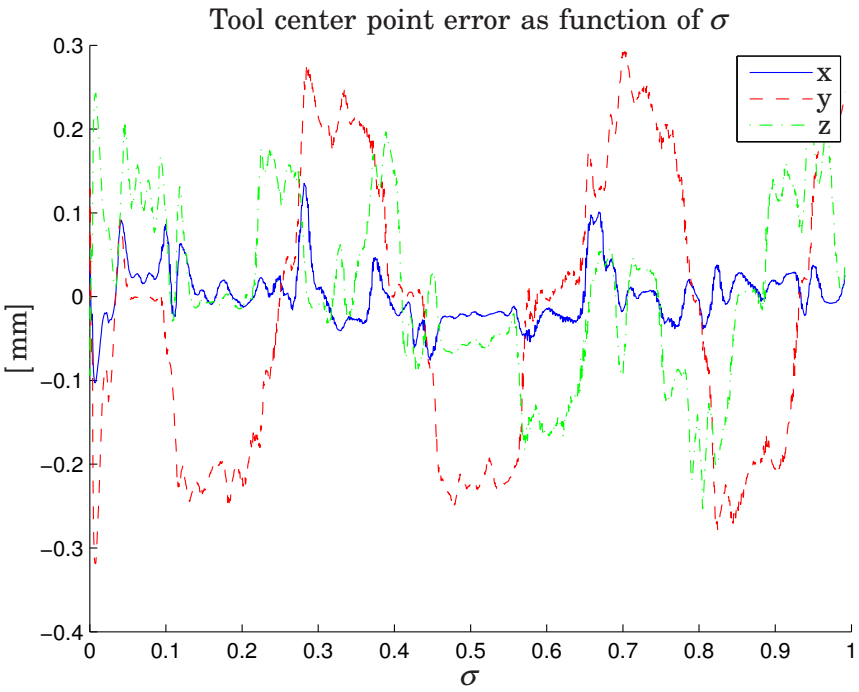

Figure 4.15 Tool center point error. Simulation 1.

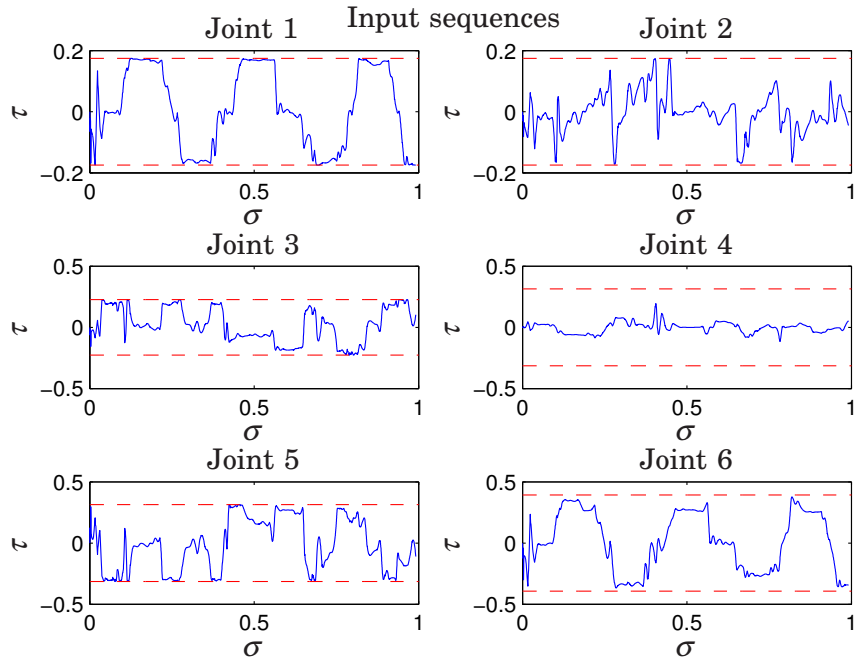

Figure 4.16 Input sequences with limits. Simulation 1.

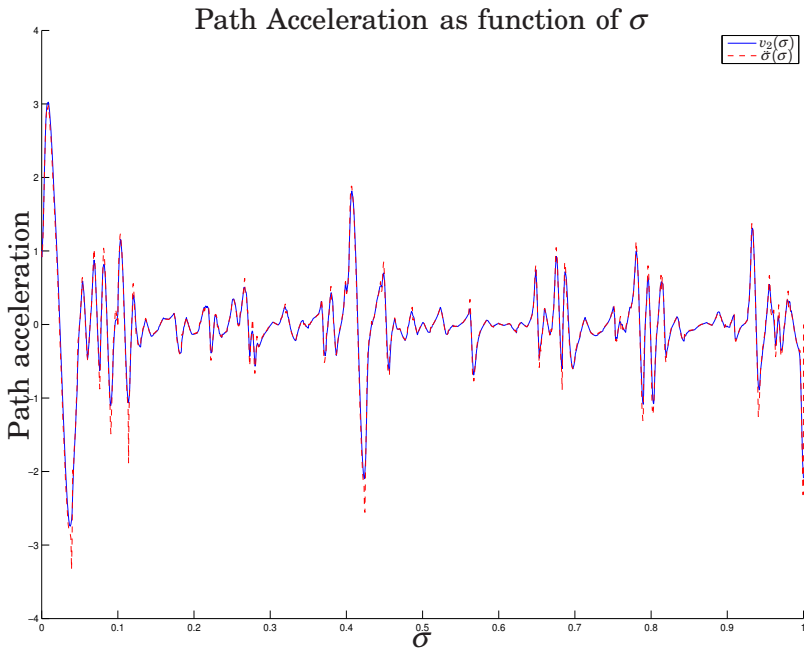

Figure 4.17 Nominal and obtained acceleration profiles. Simulation 2.

Simulation 2 The second simulation was done with the parameters $\alpha = 500$ and $k = 0$. The traversal time for this simulation was $t_f = 11.14\,\text{s}$ which is faster than the first simulation that did not use the internal feedback. By comparing Figure 4.14 and Figure 4.18 it is concluded that the path velocity is closer to the nominal in the latter as expected. Hence, the internal feedback helps reducing the errors caused by the numerical integration.

Figure 4.19 shows errors of the same magnitude as the first simulation. In Figure 4.20 it can be seen that the inputs are at limit slightly more than what was the case in simulation 1, Figure 4.16. This is expected since the internal feedback is used.

Simulation 3 In the third simulation the velocity profile scaling has been activated by setting $k = 50$. k was chosen according to the guidelines in [8]. The guidelines specifies that $\frac{1}{k}$ should be smaller than the first acceleration interval. By inspection, the interval was determined to be 0.0232 which gives $k > 43.1$. As expected, the traversal time increases when using velocity profile scaling. The traversal time in this simulation is $t_f = 11.44\,\text{s}$. Figure 4.25 show how γ develops during the simulation. In Figure 4.22 it can be seen that the path velocity profile is lower than the nominal profile.

Introducing Model Errors

In the fourth simulation a model error was introduced. The transfer function for joint 1 was multiplied by 0.8 which means that a larger input is needed in order to obtain the same results as for the unmodified model.

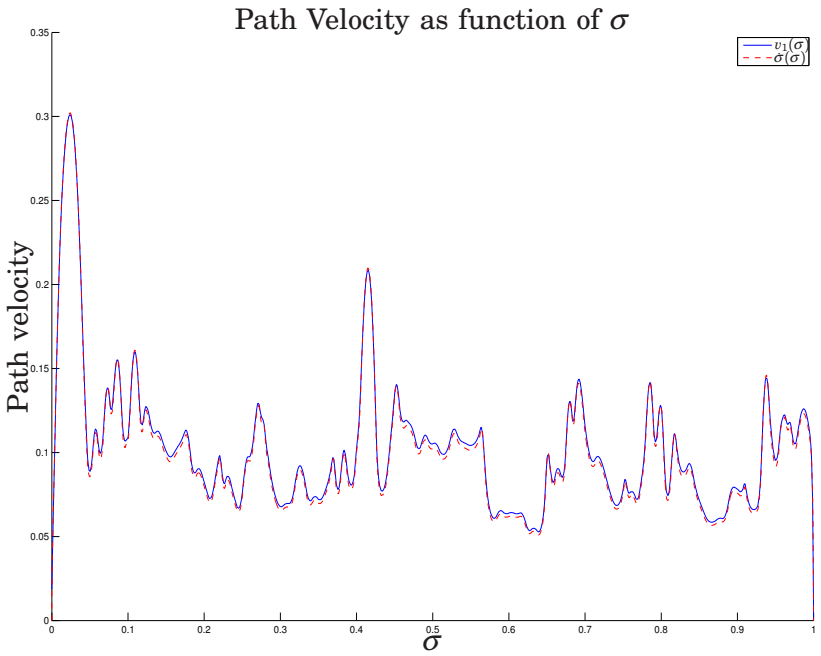

Figure 4.18 Nominal and obtained velocity profiles. Simulation 2.

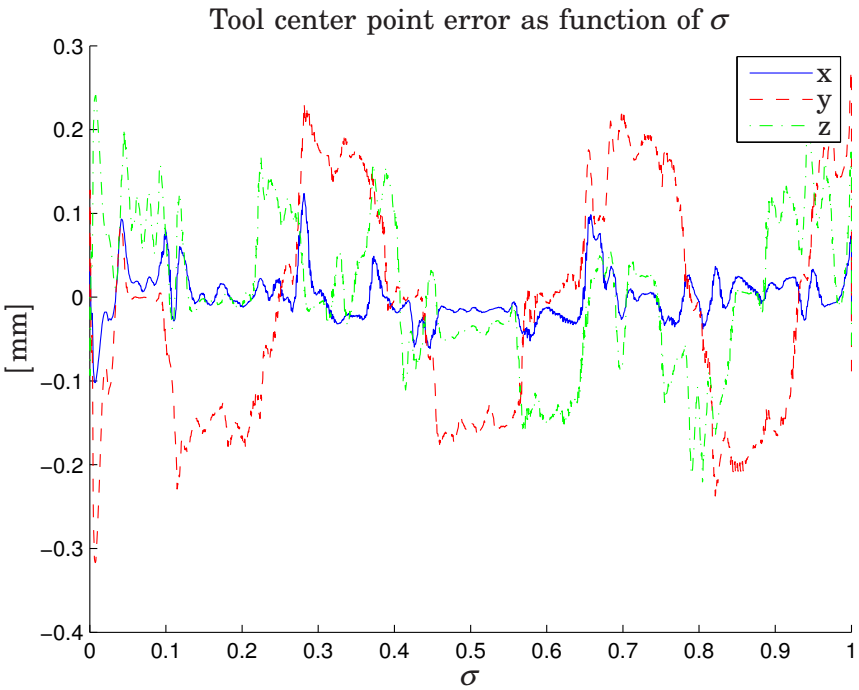

Figure 4.19 Tool center point error. Simulation 2.

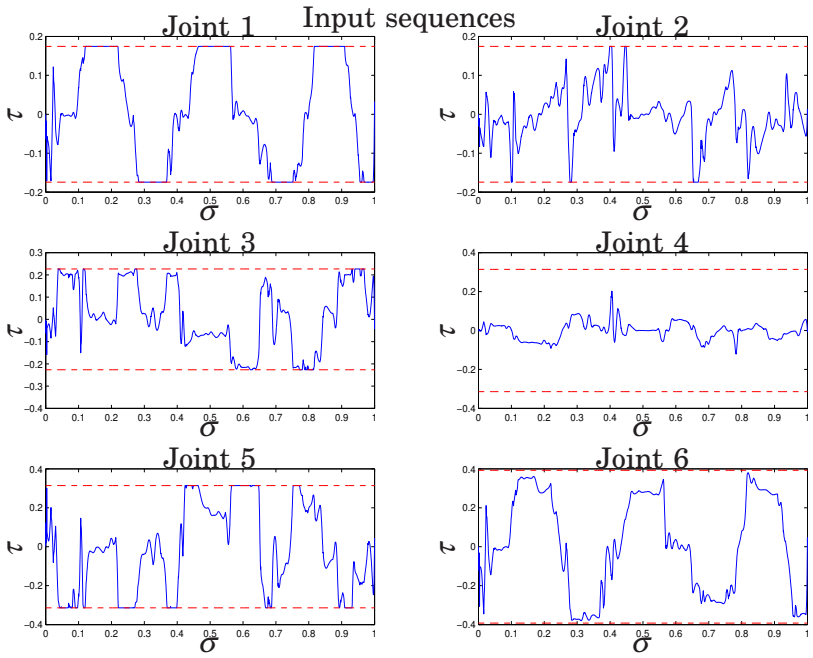

Figure 4.20 Input sequences with limits. Simulation 2.

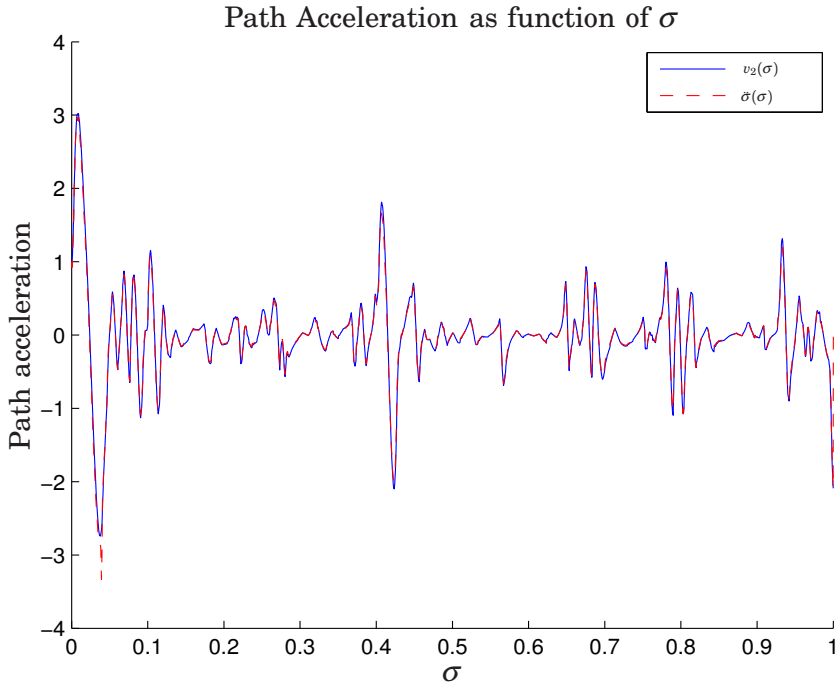

Figure 4.21 Nominal and obtained acceleration profiles. Simulation 3.

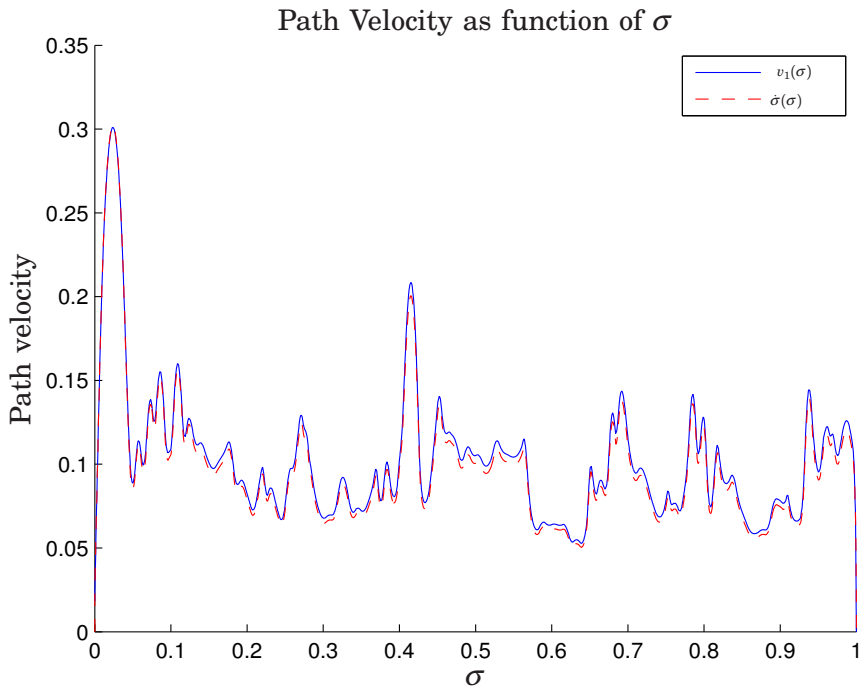

Figure 4.22 Nominal and obtained velocity profiles. Simulation 3.

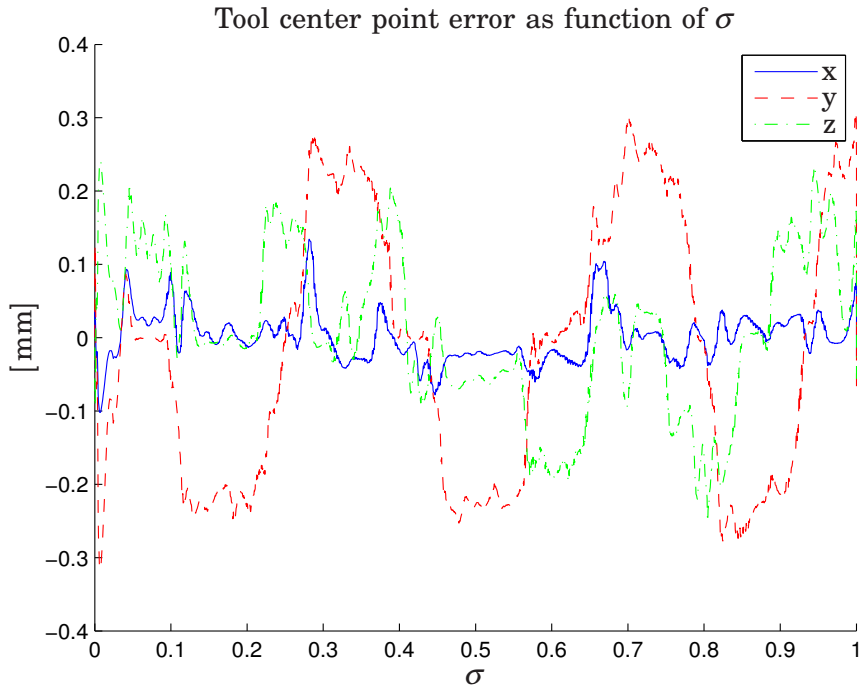

Figure 4.23 Tool center point error. Simulation 3.

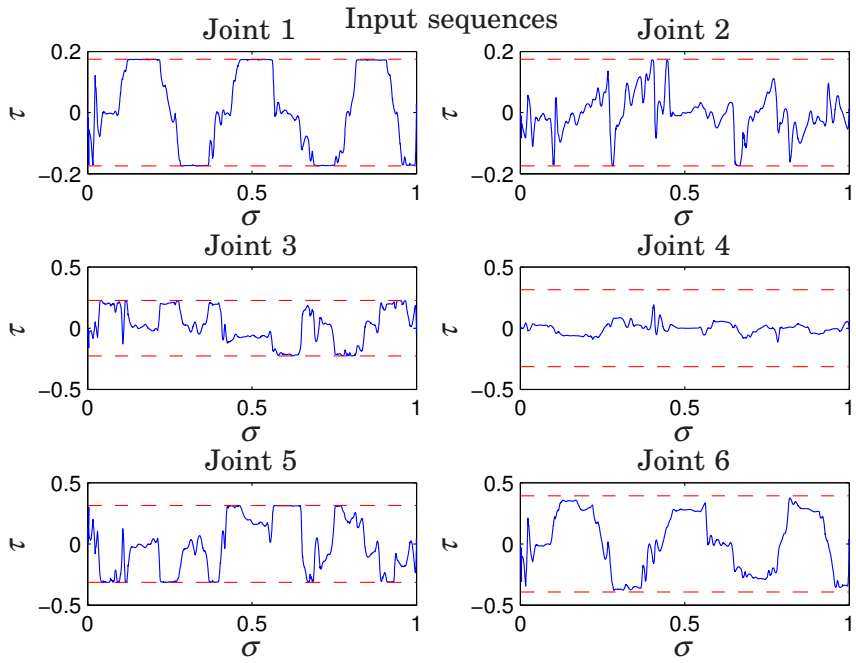

Figure 4.24 Input sequences with limits. Simulation 3.

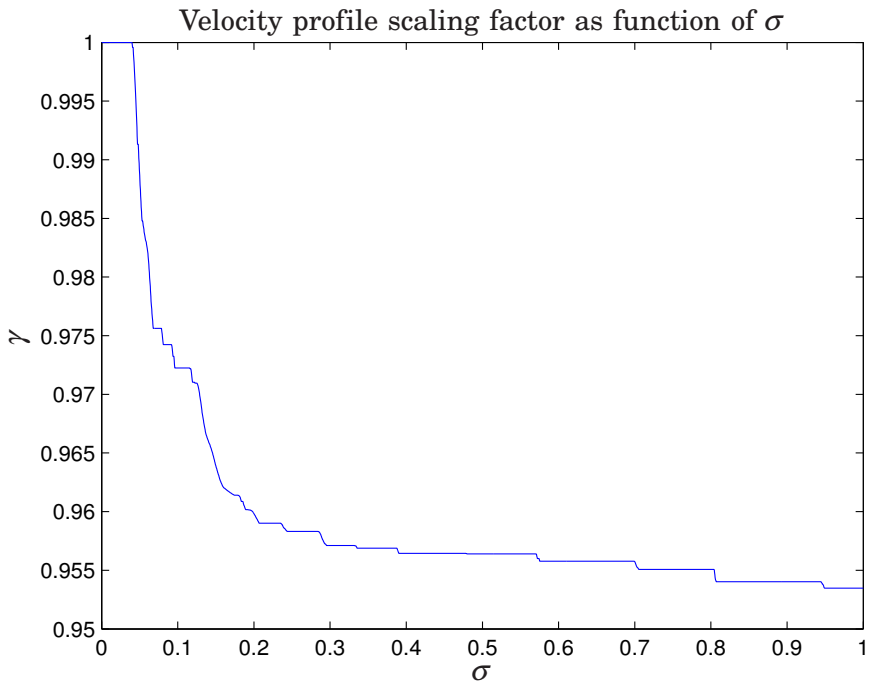

Figure 4.25 The velocity profile scaling factor in Simulation 3.

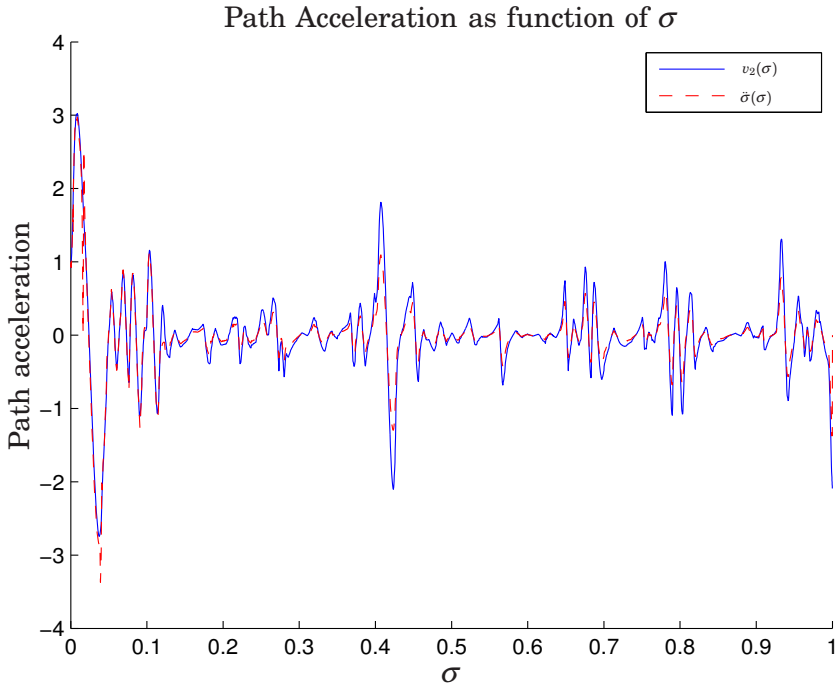

Figure 4.26 Nominal and obtained acceleration profiles. Simulation 4.

Simulation 4 The simulation was done using the parameters $\alpha = 500$ and $k = 50$. The traversal time in this simulation is $t_f = 13.83\,\text{s}$. There are multiple reasons for the longer traversal time. The first reason is the modified slower model. The second reason is the velocity profile scaling which decreases fast, see Figure 4.30, and therefore slows down along the remainder part of the path. This is also the reason that the input signals are not at limit at all time. In Figure 4.26 and Figure 4.27 the effect of the scaling factor is clearly visible. Figure 4.28 it can be seen that the errors are larger, especially in the y-direction. This is due to the fact that a movement in joint one will primarily move the tool in this direction. Note that the maximum error is only 1.4 mm. Since the controllers control the joint angles and the robot arm is extended almost a meter this is a very small error in terms of joint angle errors.

Faster Sampling

In order to investigate the effects of numerical errors due to too slow sampling, a simulation with a higher sample rate was performed. The parameters in this simulation are $\alpha = 500$, $k = 0$ and $h = 5 \cdot 10^{-4}\,\text{s}$.

Simulation 5 Comparing Figure 4.31 to Figure 4.13 and Figure 4.32 to Figure 4.14 it is clear that the decreased sample time make the obtained profile equal to the nominal profiles. Even though the decreased sample time makes the obtained profiles approach the nominal profiles it also creates new numerical errors. These errors arise from the fact that the optimization is discrete and done over a specified number of points. When the nominal profiles are used in Simulink® they are implemented as look-up tables with linear interpolation between the points in the table. This

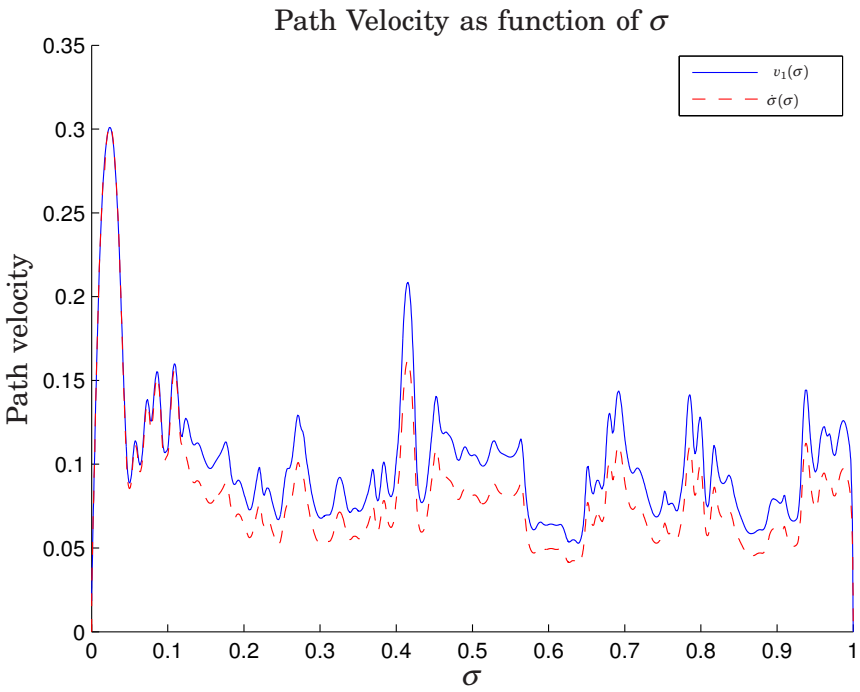

Figure 4.27 Nominal and obtained velocity profiles. Simulation 4.

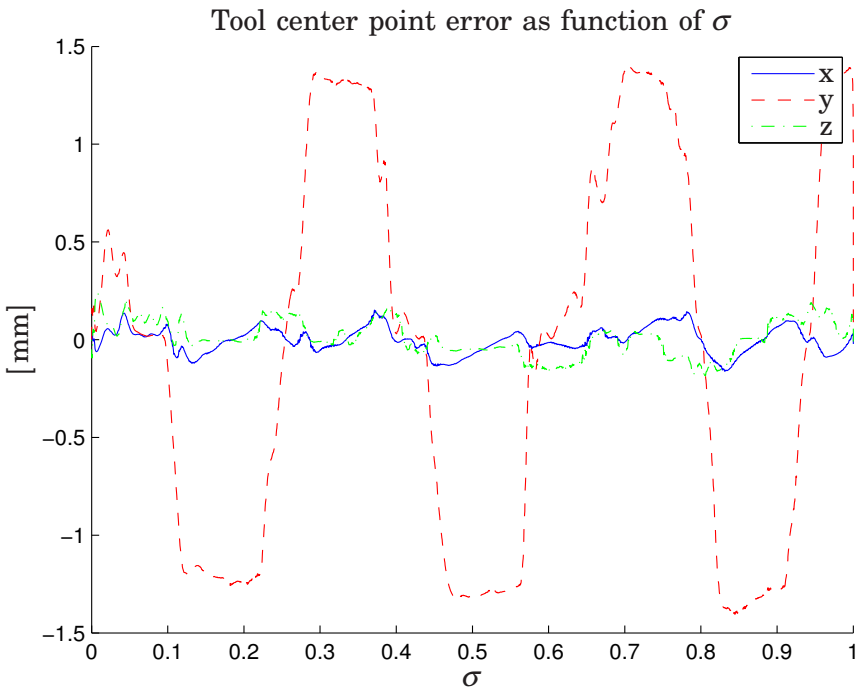

Figure 4.28 Tool center point error. Simulation 4.

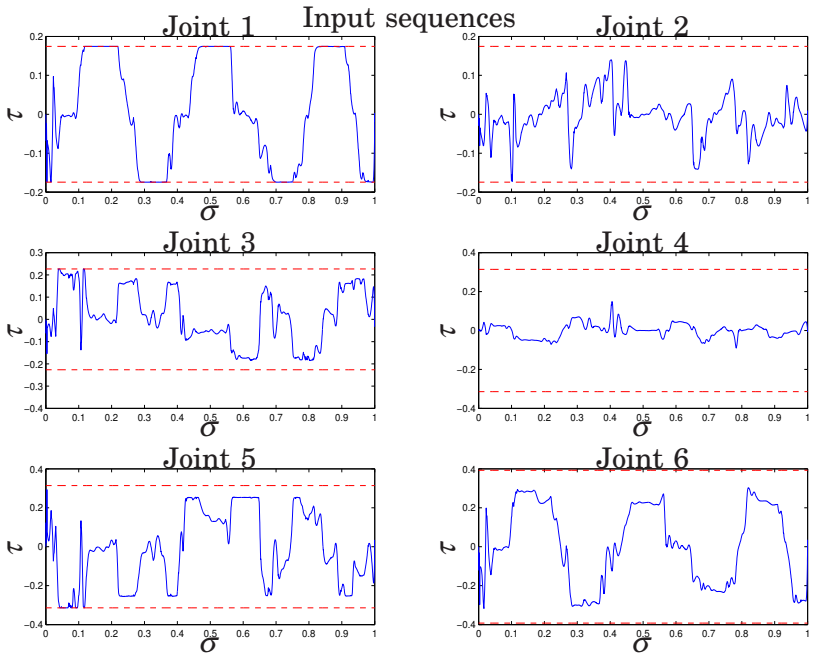

Figure 4.29 Input sequences with limits. Simulation 4.

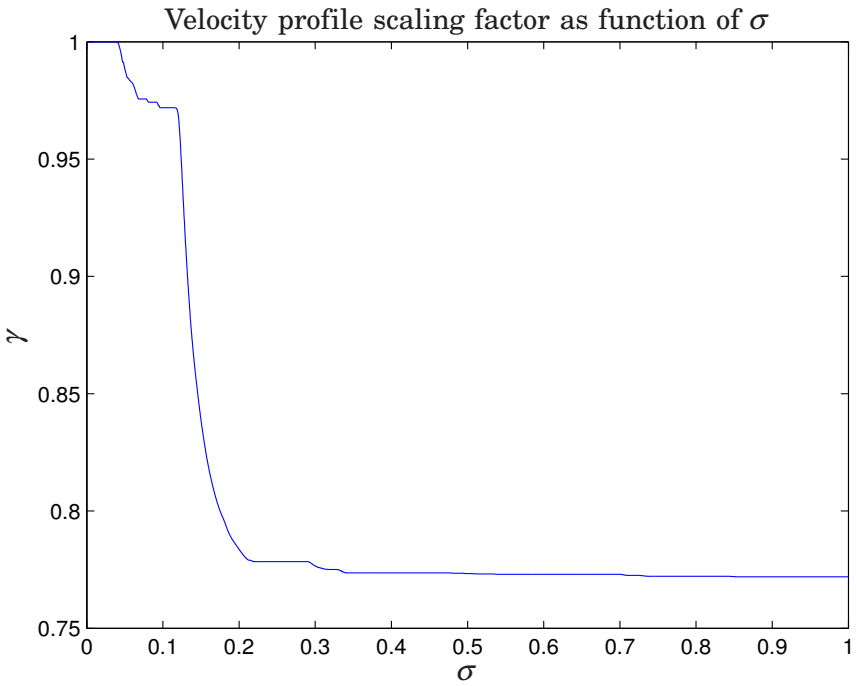

Figure 4.30 The velocity profile scaling factor in Simulation 4.

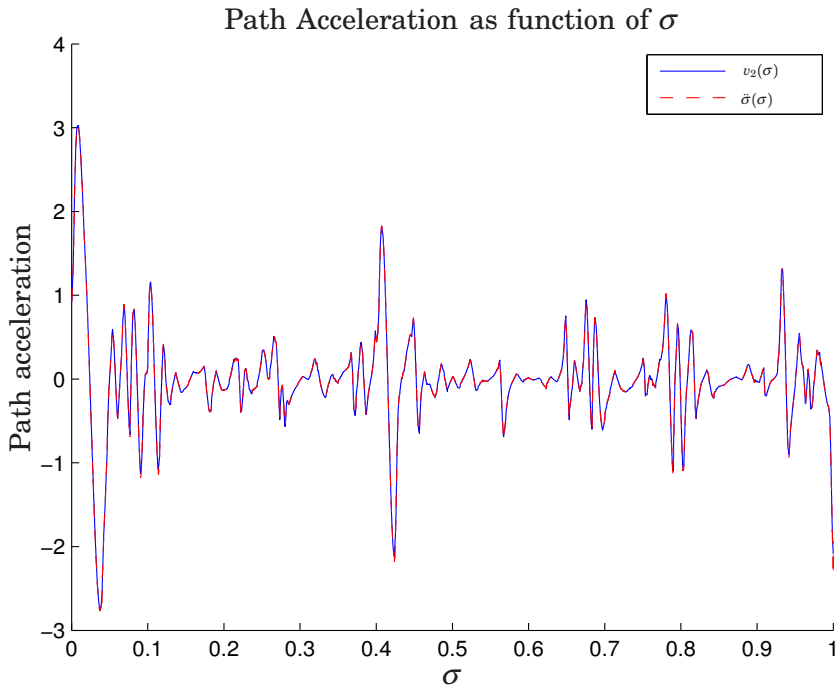

Figure 4.31 Nominal and obtained acceleration profiles. Simulation 5.

introduces a small mismatch between the two nominal trajectories, so when v_2 is numerically integrated the result is not equal to v_1. This mismatch makes $\dot{\sigma} < 0$ as σ approaches s_f. This is the reason for the big error towards the end of the track that can be seen in Figure 4.33. Note that besides this big error at the end, the error is actually smaller than in the four previous simulations.

4.7 Practical issues with the PVC algorithm

During this master thesis, different paths, models and controllers have been simulated. This section will discuss two problems with the PVC algorithm itself that were encountered.

The first problem concerns the part of the PVC algorithm that calculates the bound on the path acceleration, see Eq. (3.4) and Eq. (3.5). The calculations performed in these equations are very sensitive to noise, especially when the bounds on the input τ are small. In Eq. (3.4) a scenario where β_{2_i} is larger than τ_i^{max} or smaller than τ_i^{min} at same time as β_{1_i} is close to zero and with the same sign as β_{2_i} this could lead to a large $\ddot{\sigma}_{max}^i$ or a small $\ddot{\sigma}_{min}^i$. When performing the calculations in Eq. (3.5) the effect will be that the upper bound is smaller than the lower bound. The risk for this scenario increases when the measurement signal is noisy. Using some kind of filter could be a possible way around this problem. Caution must be taken to ensure that the filters do not cause to much lag.

The first problem, noisy mesurments corrupting the bound calculation, can cause the second problem which occurs when the calculated upper

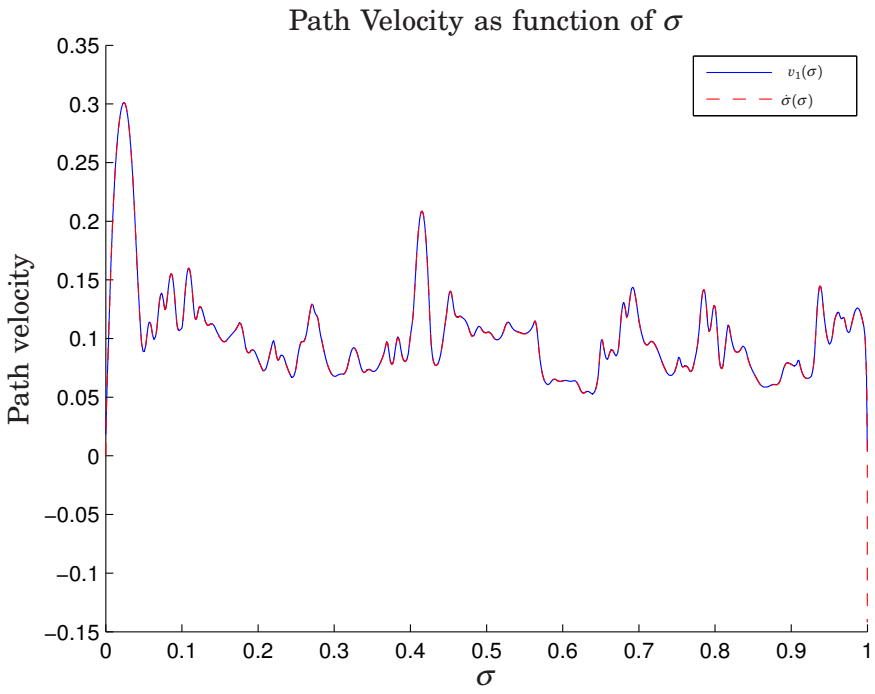

Figure 4.32 Nominal and obtained velocity profiles. Simulation 5.

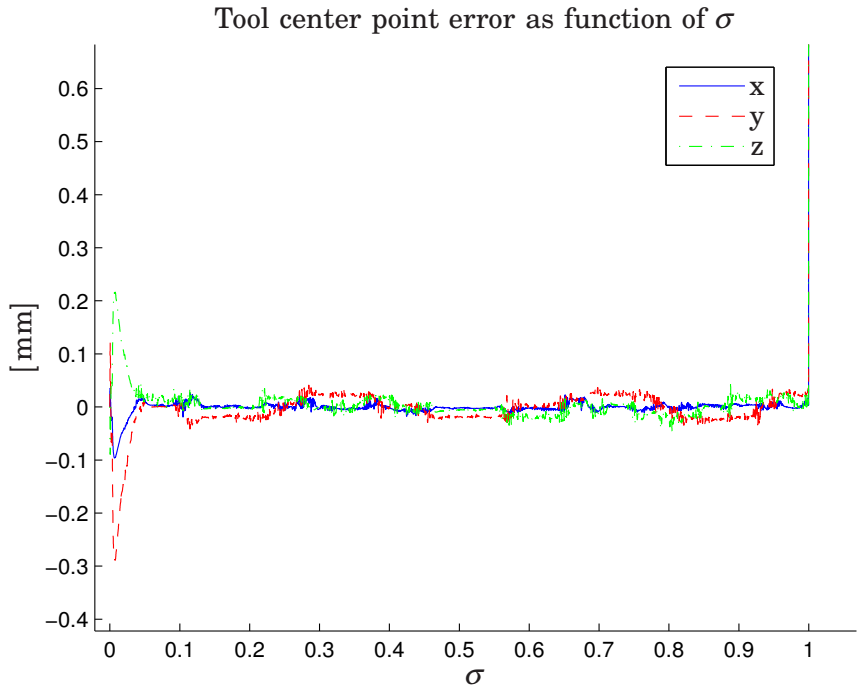

Figure 4.33 Tool center point error. Simulation 5.

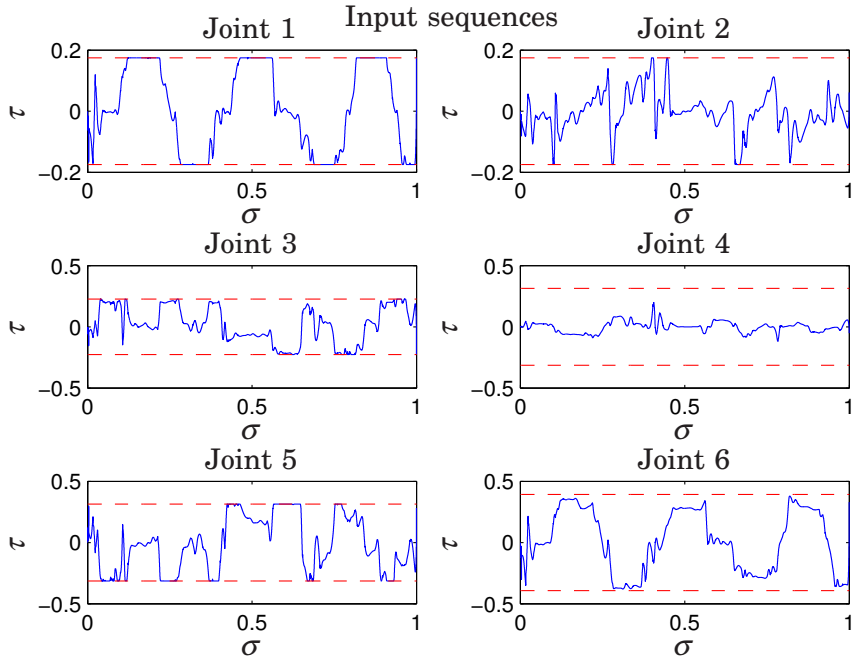

Figure 4.34 Input sequences with limits. Simulation 5.

bound on the path acceleration is smaller than the lower bound. In that case the saturation block does not saturate the signal, which instead just passes trough the block unaffected. If the saturation block switches between saturating the signal and letting it pass right through this could render a jumpy path acceleration which is far from the nominal.

4.8 Using the PVC with the Robot

The interface which allows communication with the robot through Simulink® is described in [10]. It basically allows reading from, modifying and sending data back to the robot. The modification in this case is the PVC. The PVC tested on the robot was the same used in the simulations. As mentioned in the introduction, Chapter 1, the model is converted into C-code using Real-Time Workshop. This code can then be run on the robot using the available graphical user interface, (GUI). The GUI has four modes; *unload*, *load*, *submit*, *obtain*. In *unload* no model is loaded, as the name suggests. Switching to *load*, loads the specified model. The *submit* mode allows reading the signals from the robot and using the model but it does not send data from the model to the robot. Finally, the *obtain* mode is like the *submit* mode but the signals from the model are sent to the robot. Great care should be taken when switching to the *obtain* mode.

The Simulink® model starts executing as soon as the model is loaded. Therefore switches, controlled by the variable *f_switch*, were inserted. The switches make sure that no integrators in the PVC receives anything but zero which prevents the PVC controller from starting. Setting *f_switch*

equal to one in the GUI starts the PVC.

The PVC requires measured positions and velocities for each joint. The signals used for this purpose were the *irb2ext[i].posRaw_abs* and *irb2ext[i].velFlt*. These signals contain motor angles and motor angle velocities and therefore they have to be converted into arm angles and velocities. This was done using a Simulink® block from the extctrl library, see [10].

As stated in Section 4.1, ABB's position controller was disabled. This is done by setting the gains in the position controllers to zero. A specific variable *shutOffABBController* handles the switches that either sets the controller gains to zero or back to their original values. Setting *shutOffABBController* to one switches the ABB controllers off while setting it to zero switches them on.

ABB has implemented safety mechanisms that lock the brakes if the position reference sent from the main computer or from the Simulink® model is too far from the real position. In order to be able to run the PVC algorithm this safety feature was bypassed by sending the measured position as the reference. Since the ABB position controller is turned of by setting the gains to zero this does not interfere with the PVC.

4.9 Experiment

Unfortunately, no experiments on the real robot using the full PVC with the path in Section 4.2 succeeded. The robot was able to follow the path, using only the path trajectories and the controller, but only at a speed lower than the optimal. However, when connecting the PVC algorithm, it did not behave as expected. Logs from experiments show that some values unexpectedly goes towards infinity. The source of this behavior has not been found, but the same problem occurs, sometimes, in simulations. Since these problems arose at the end of this master thesis there was no time to further investigate the reasons for this behavior. Initial tests using simple models, double integrators, and paths defined as sinus functions did actually work in both simulations and on the real robot. These tests did only include joint one, two and three. Since the inital tests were not intended to be used in this thesis, no experimental data was saved.

5. Conclusions and Future Work

In this thesis, two models have been identified for each joint. A path has been recorded using lead-through. A minimum time optimization problem has been formulated, using the path and the identified second order model, and modified to reduce the number of states. The problem has also been transformed so that the optimization is done over a fixed interval. This was necessary in order to find an optimal solution without having to generate an initial guess that was close to the optimal solution. The reformulation also decreased the time it took for the solver to find a optimal solution.

Usually when formulating optimization problem it has to be done for instance in AMPL or in some other mathematical language which can be a cumbersome work. Modelica and Optimica provides a natural way of formulating optimization problems.

Optimica can also be used, even-though this was not the case in this thesis, to solve optimization problems with Modelica models which were not made intended for optimization. Thus optimizations can be done using a model that was developed with another intention, e.g., simulations.

In this thesis a beta version of Optimica has been used. In future releases the syntax will be changed and the optimization problem will not be using AMPL but instead be solved directly using IPOPT [13].

The result of using Modelica and Optimica to solve the minimum time optimization problem is satisfactory. When solving optimization problems, you often knows what kind of answer you would like. The problem is to formulate a question that gives the desired result.

The PVC algorithm has been implemented in Simulink®. Unfortunately, some problems remain to be addressed in order to make function in the desired way. The unexpected behavior that renders infinite values would have to be solved in order to use the PVC on the robot.

The PVC algorithm has been validated through simulations where the effect of different parameters has been investigated. The internal feedback has proven efficient for obtaining faster motion along the path and provides some compensation for the numerical errors that occur.

Had there been even more time to work on this project there are a number of different things that could have been done.

Getting the PVC to work on the real robot would have been first priority. Secondly, investigating how to filter the β signals in order ensure robustness against noise. The third thing that would be interesting to examine is what to do if the lower bound is greater than the upper bound in the saturation block.

Trying the identified higher order model, in both optimization and in the PVC, would also be interesting. Finally, it would be nice to use the ABB control structure in the PVC and to identify models from motor voltage or torque to joint position.

6. Bibliography

[1] ABB. IRB140B Data Sheet. www.abb.com/robotics/.

[2] ABB. IRC5 Data Sheet. www.abb.com/robotics/.

[3] ABB. ABB Home Page, 2009. http://www.abb.com.

[4] Johan Åkesson. The optimica compiler – 0.3 users guide. Technical report, Department of Automatic Control, 2007. Distributed together with Optimica 0.3.

[5] Johan Åkesson. *Tools and Languages for Optimization of Large-Scale Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, November 2007.

[6] Modelica Association. Modelica overview. http://www.modelica.org/documents/ModelicaOverview14.pdf.

[7] Modelica Association. Modelica® - a unified object-oriented language for physical systems modeling - language specification. http://www.modelica.org/documents/ModelicaSpec30.pdf.

[8] Ola Dahl. *Path Constrained Robot Control*. PhD thesis, Department of Automatic Control, Lund University, Sweden, April 1992.

[9] Henrik Danielsson. Vehicle path optimisation. Master's Thesis ISRN LUTFD2/TFRT--5797--SE, Department of Automatic Control, Lund University, Sweden, June 2007.

[10] Isolde Dressler. Force Control Interface for ABB S4. Technical report, Lund Univeristy, Department of Automatic Control, LTH, 2008.

[11] AMPL A Modeling Language for Mathematical Programming. Ampl home page, 2009. http://www.ampl.com/.

[12] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons, jan 2004.

[13] JModelica. JModelica Home page. http://www.jmodelica.org/.

[14] Rolf Johansson. *System Modeling and Identification*. Prentice Hall, Englewood Cliffs, New Jersey, January 2008.

[15] Mathworks®. Mathworks® matlab® control toolbox. http://www.mathworks.com/products/control/.

[16] IPOPT Interior Point OPTimizer. Ipopt home page, 2009. https://projects.coin-or.org/Ipopt.

A. Model Identification

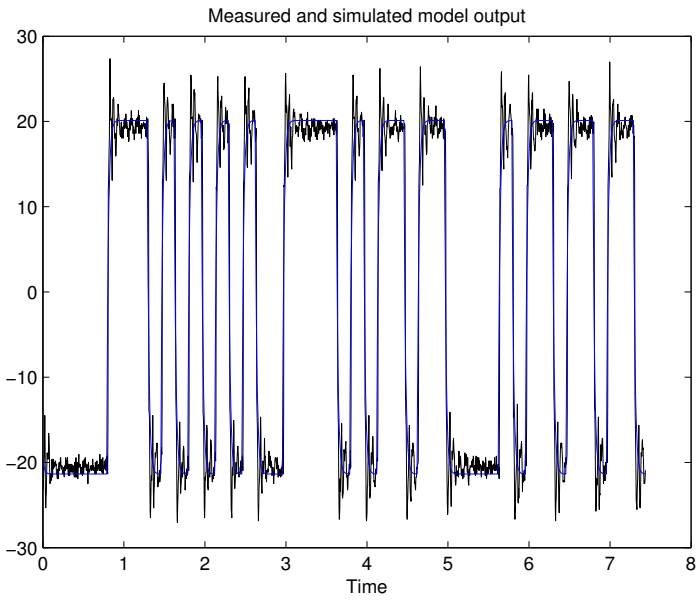

Figure A.1 Fit between model and cross-validation data. Joint 1

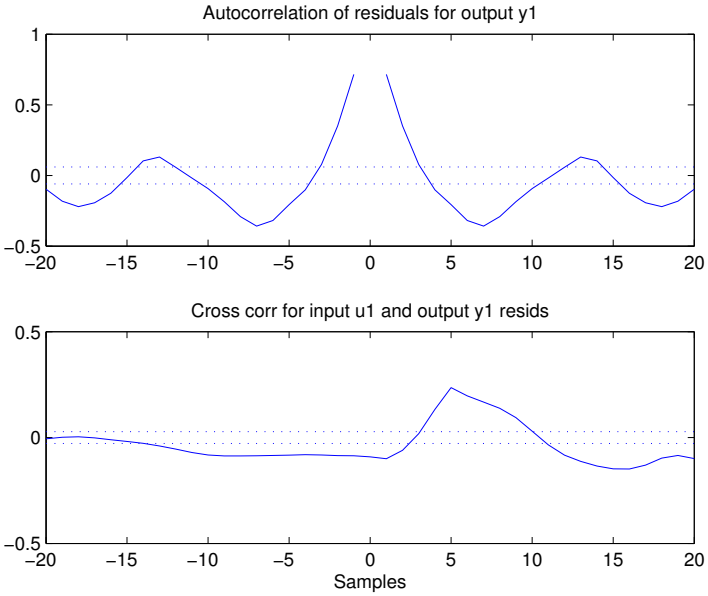

Figure A.2 Auto correlation and cross-correlation for the residuals. Joint 1

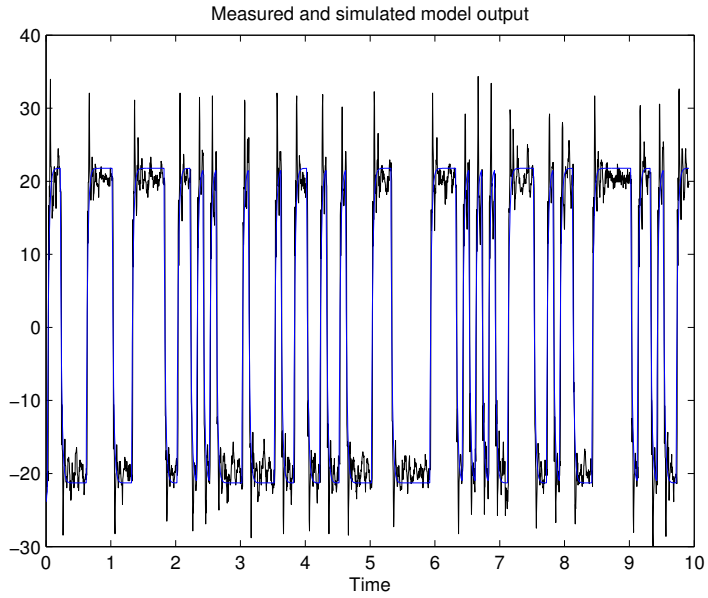

Figure A.3 Fit between model and cross-validation data. Joint 2.

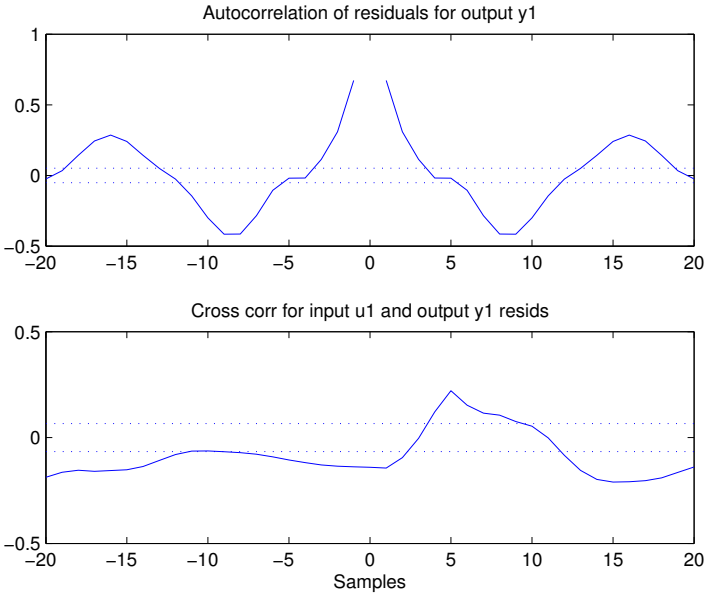

Figure A.4 Auto correlation and cross-correlation for the residuals. Joint 2

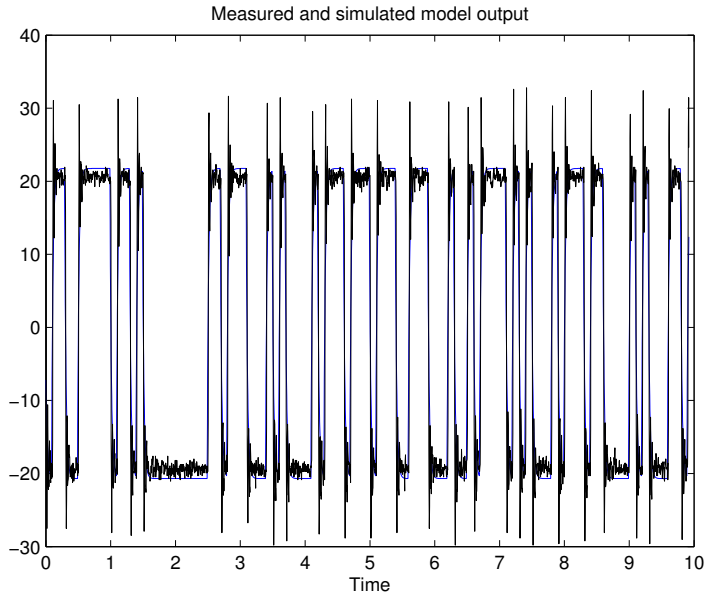

Figure A.5 Fit between model and cross-validation data. Joint 3

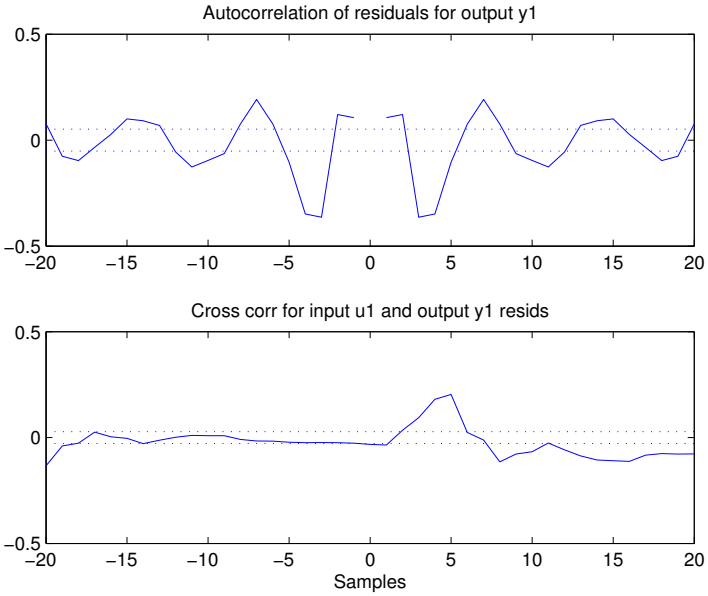

Figure A.6 Auto correlation and cross-correlation for the residuals. Joint 3

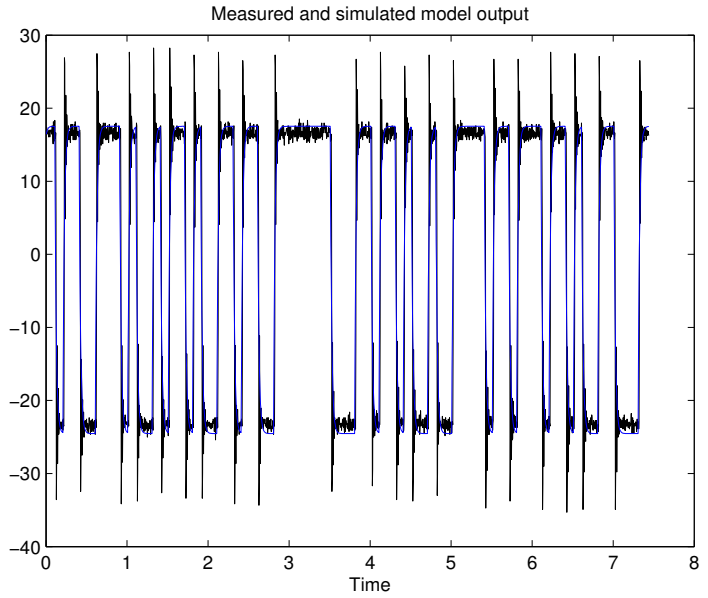

Figure A.7 Fit between model and cross-validation data. Joint 4

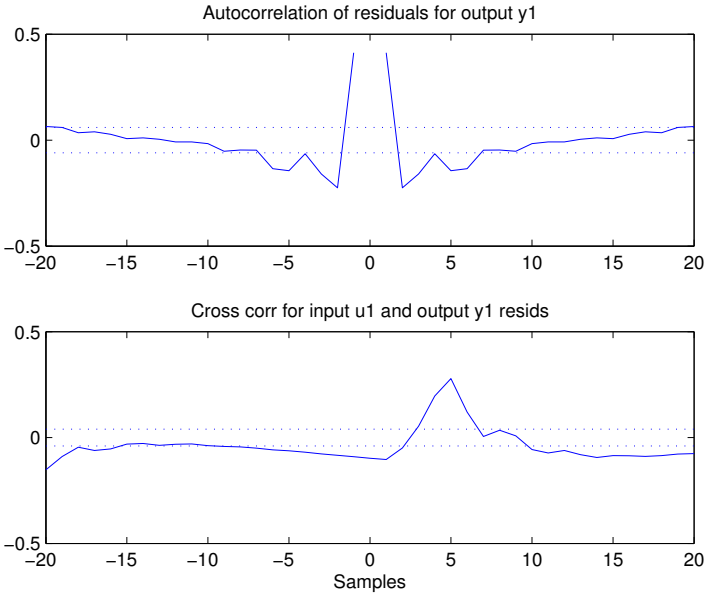

Figure A.8 Auto correlation and cross-correlation for the residuals. Joint 4

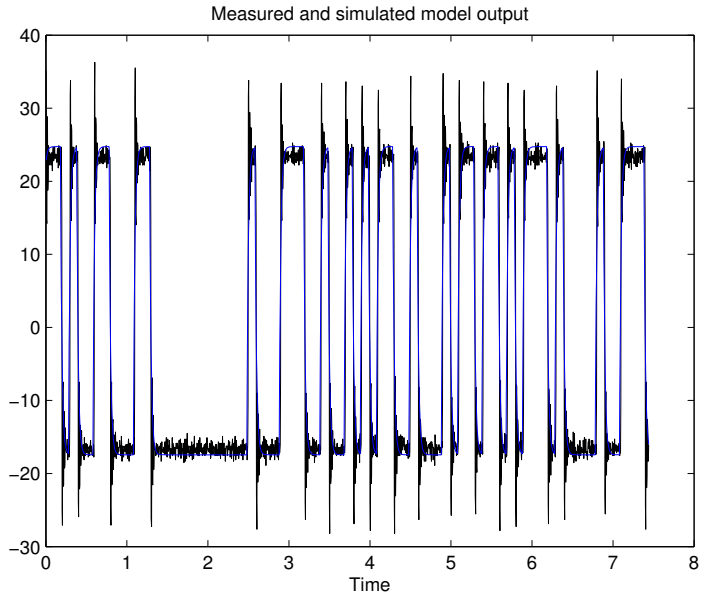

Figure A.9 Fit between model and cross-validation data. Joint 5

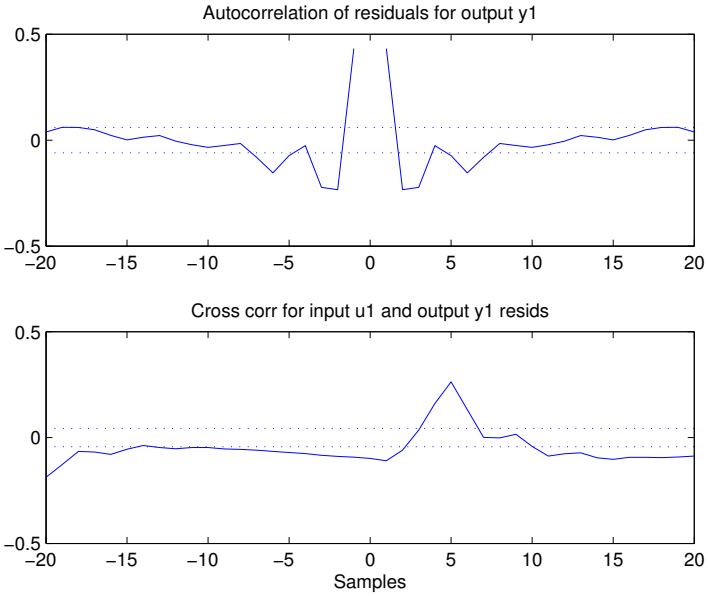

Figure A.10 Auto correlation and cross-correlation for the residuals. Joint 5

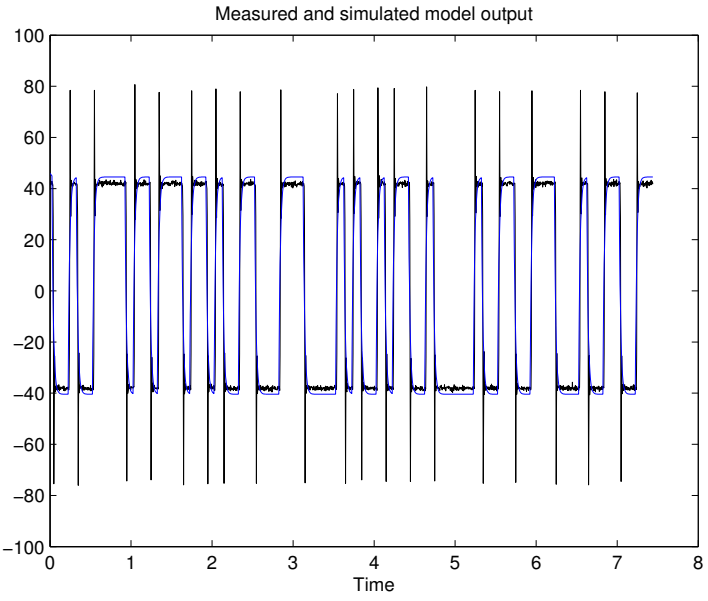

Figure A.11 Fit between model and cross-validation data. Joint 6

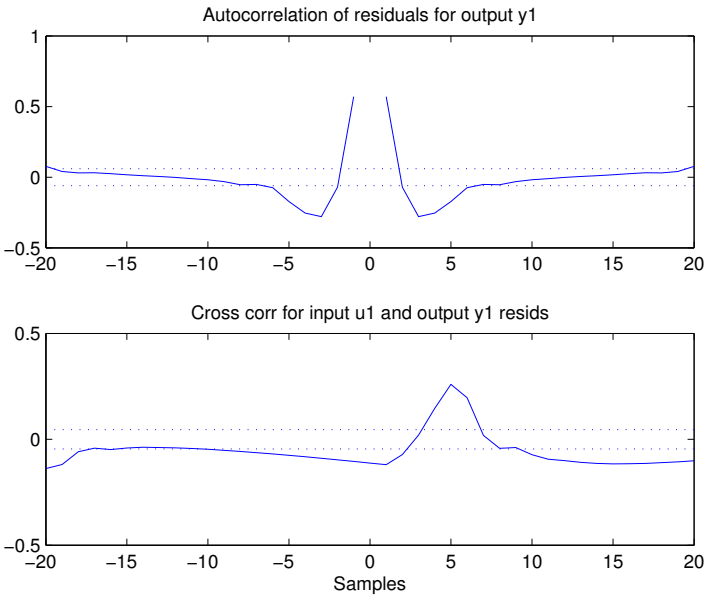

Figure A.12 Auto correlation and cross-correlation for the residuals. Joint 6

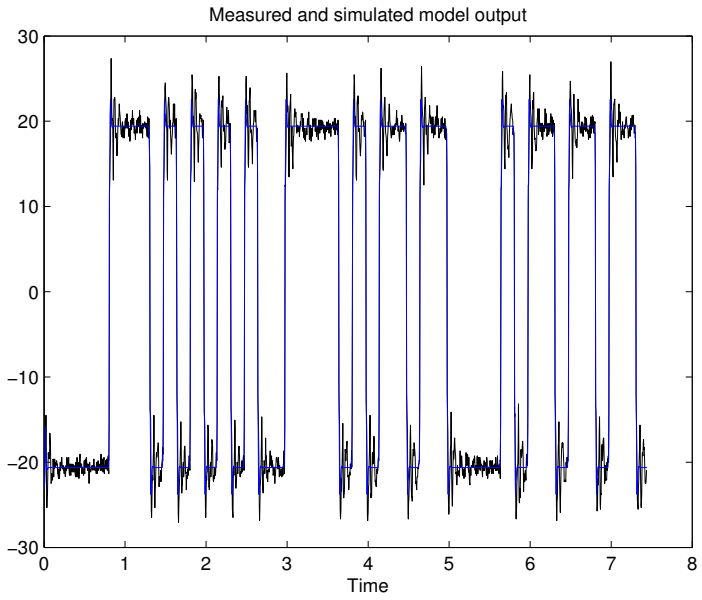

Figure A.13 Fit between ARMAX-model and cross-validation data. Joint 1

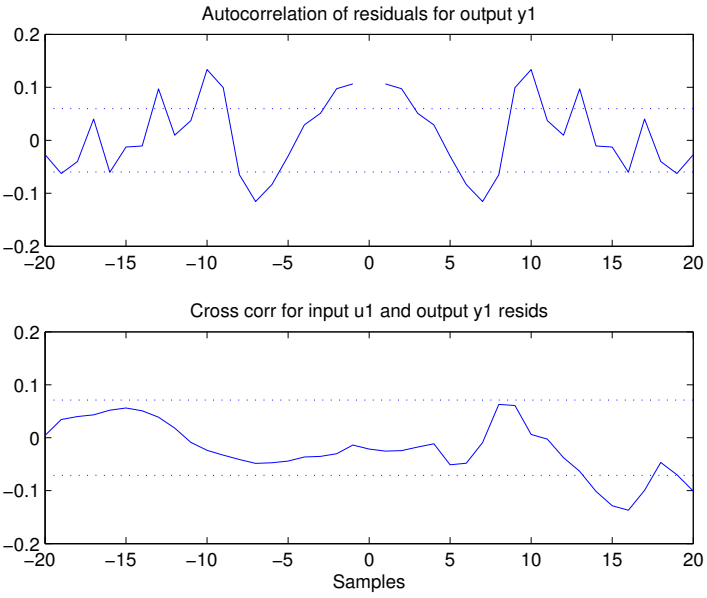

Figure A.14 Auto correlation and cross-correlation for the residuals. Joint 1

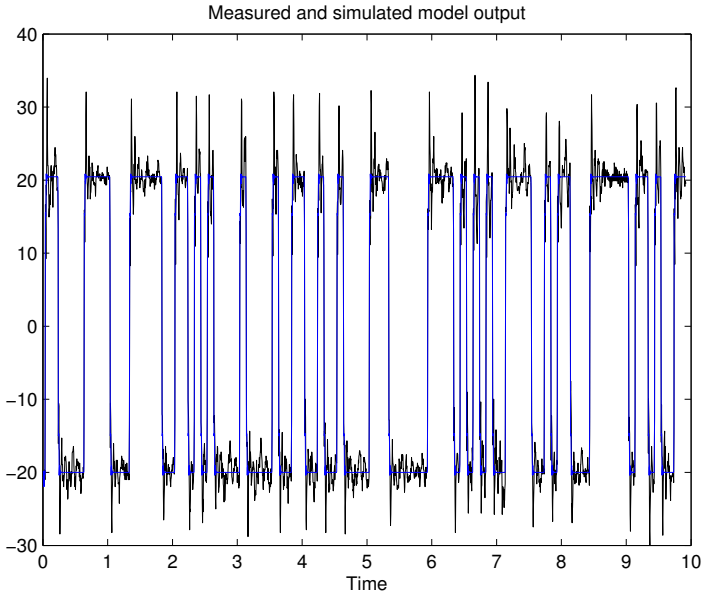

Figure A.15 Fit between ARMAX-model and cross-validation data. Joint 2

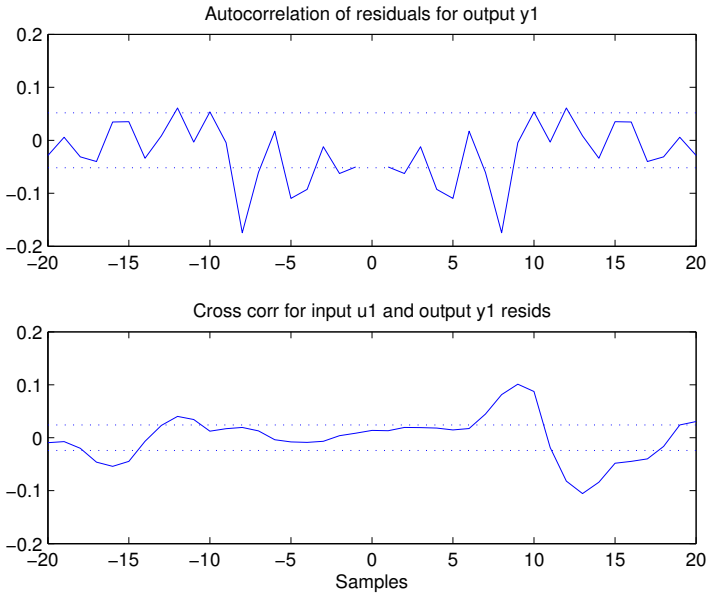

Figure A.16 Auto correlation and cross-correlation for the residuals. Joint 2

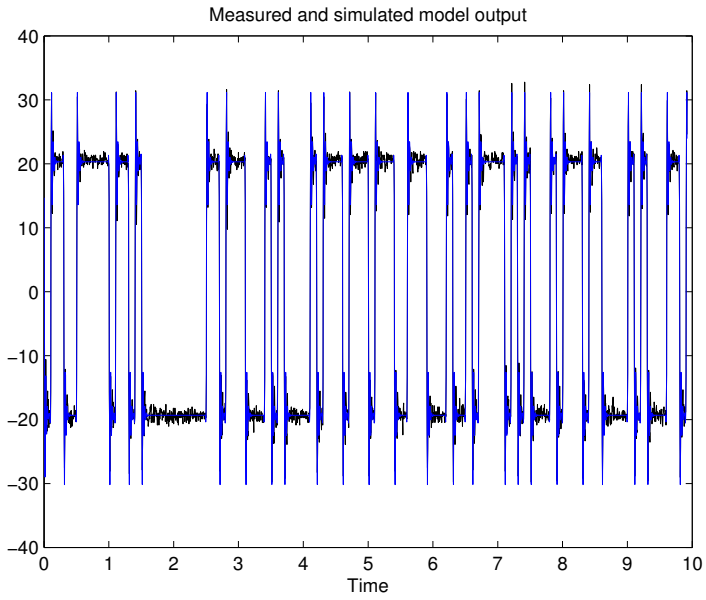

Figure A.17 Fit between ARMAX-model and cross-validation data. Joint 3

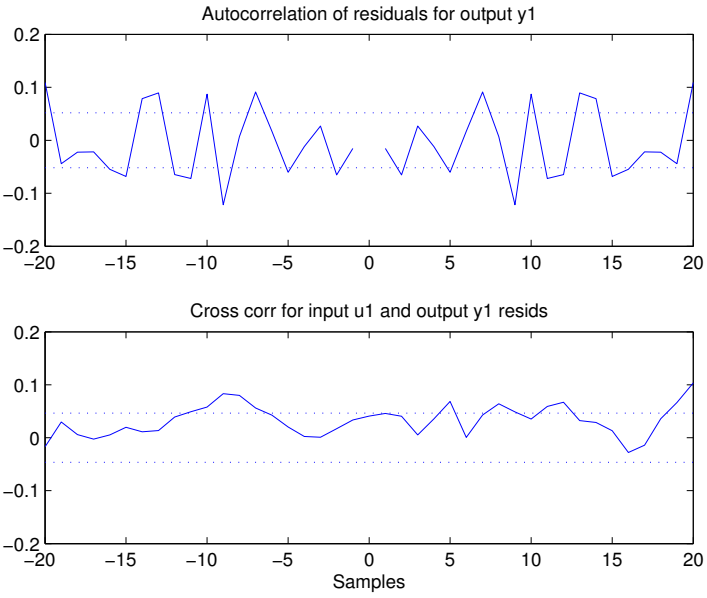

Figure A.18 Auto correlation and cross-correlation for the residuals. Joint 3

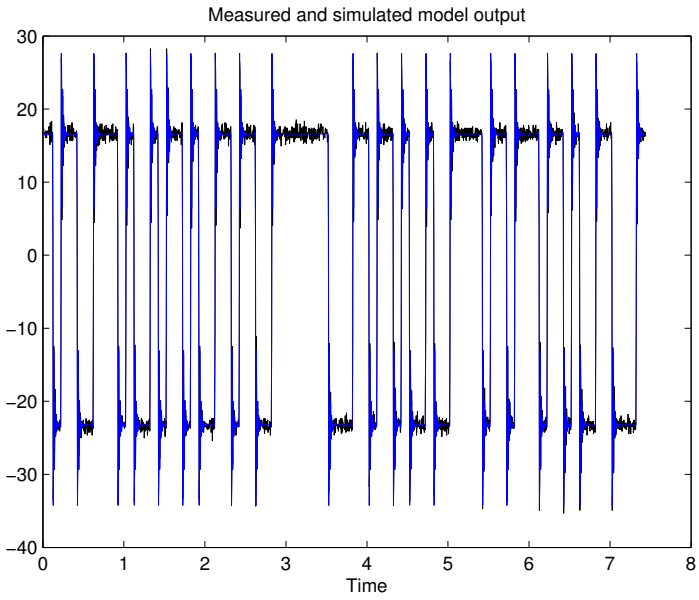

Figure A.19 Fit between ARMAX-model and cross-validation data. Joint 4

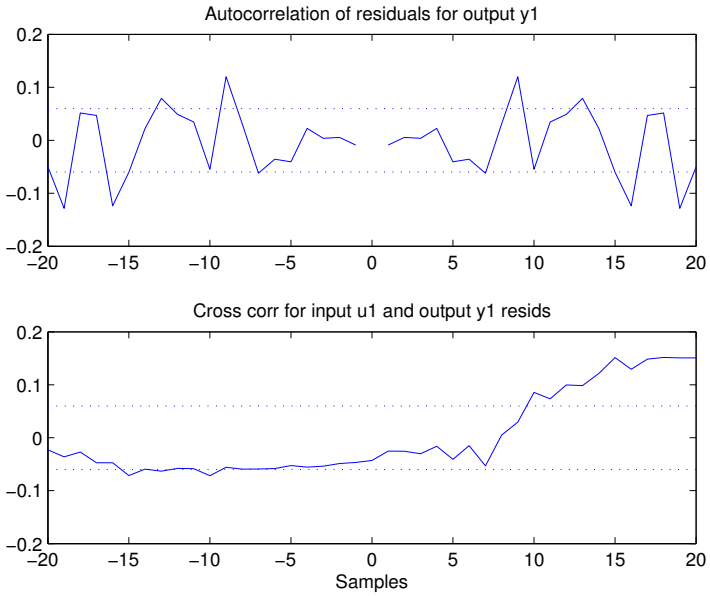

Figure A.20 Auto correlation and cross-correlation for the residuals. Joint 4

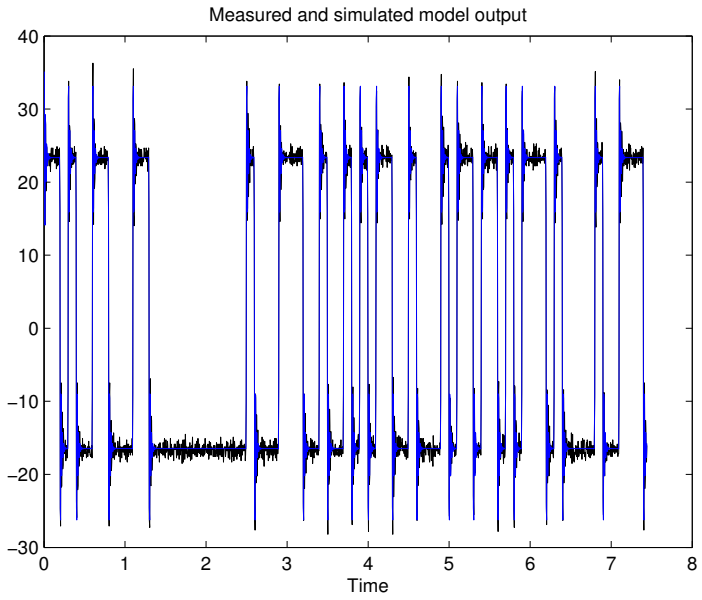

Figure A.21 Fit between ARMAX-model and cross-validation data. Joint 5

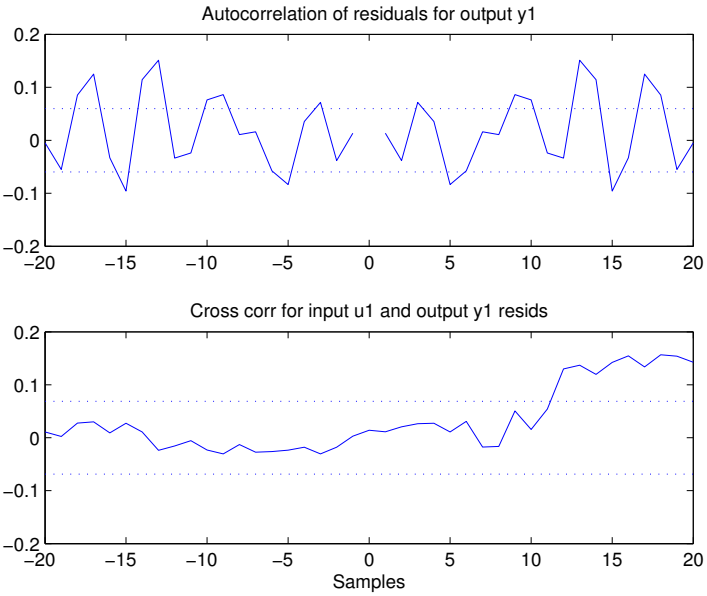

Figure A.22 Auto correlation and cross-correlation for the residuals. Joint 5

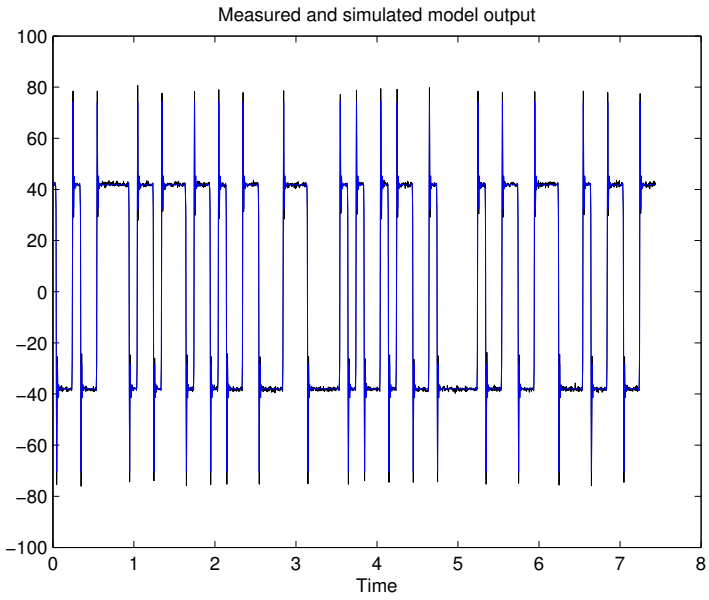

Figure A.23 Fit between ARMAX-model and cross-validation data. Joint 6

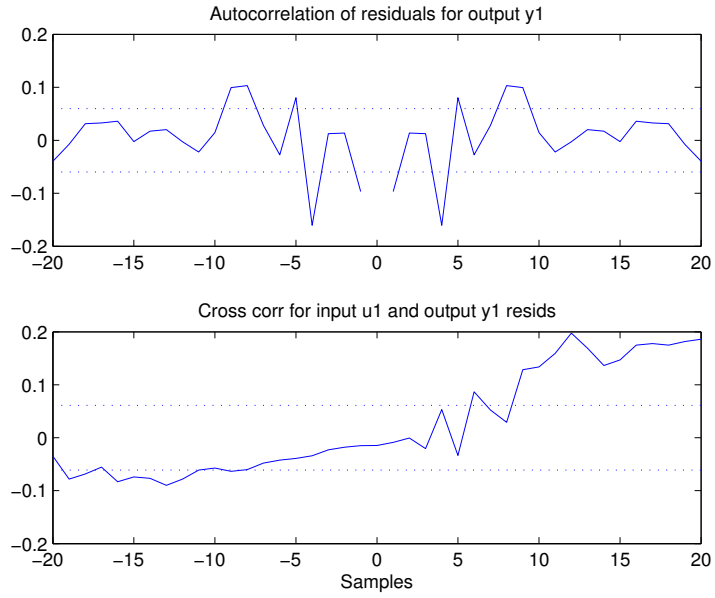

Figure A.24 Auto correlation and cross-correlation for the residuals. Joint 6

B. Optimization Results

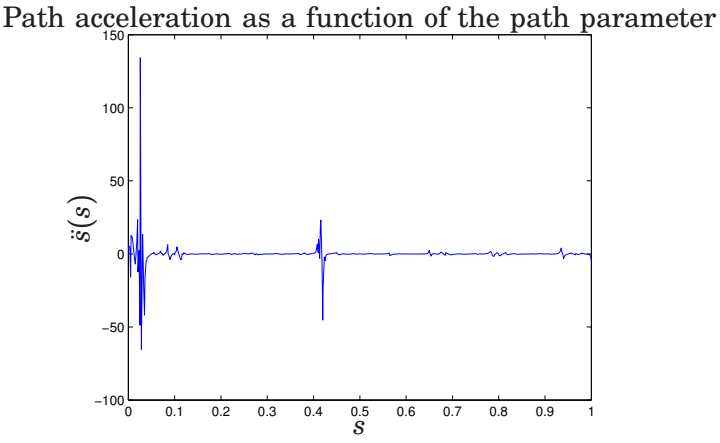

Figure B.1 Optimal path acceleration profile for the *Finding the Minimum Traversal Time* problem in Section 4.3

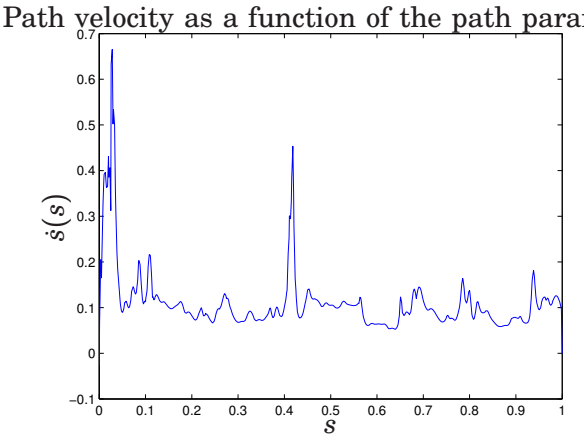

Figure B.2 Optimal path velocity profile for the *Finding the Minimum Travesal Time* problem in Section 4.3

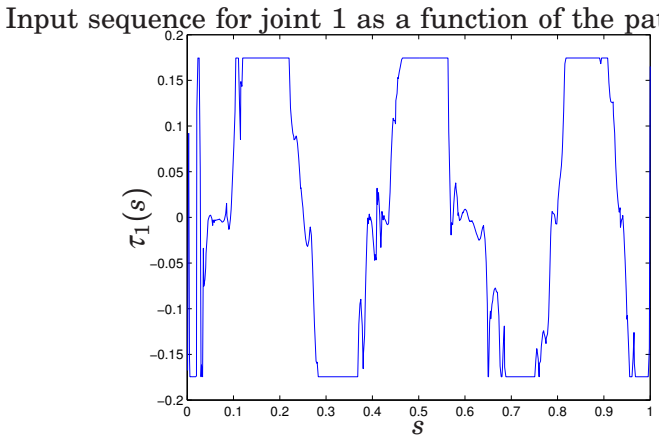

Figure B.3 Optimal input sequence *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3. Joint 1.

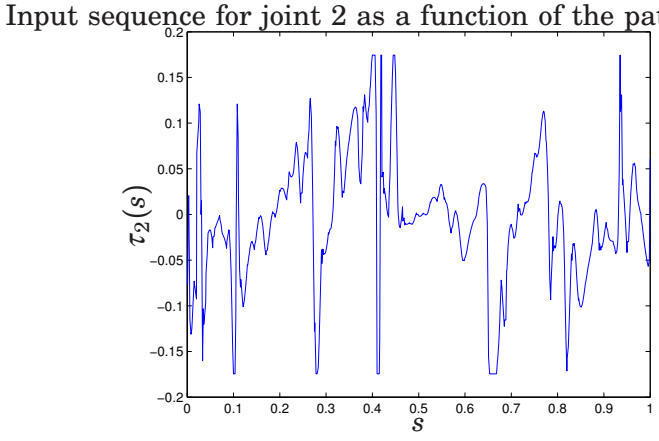

Figure B.4 Optimal input sequence *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3. Joint 2.

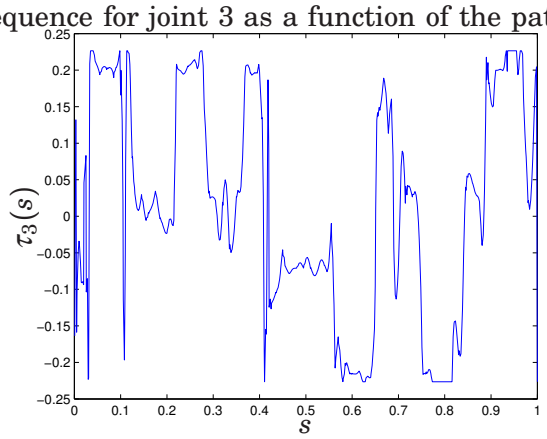

Figure B.5 Optimal input sequence *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3. Joint 3.

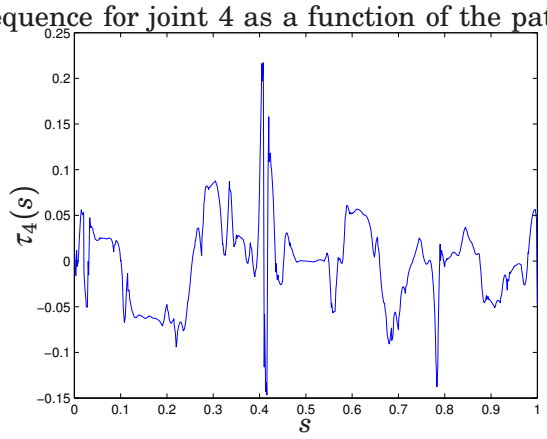

Figure B.6 Optimal input sequence *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3. Joint 4.

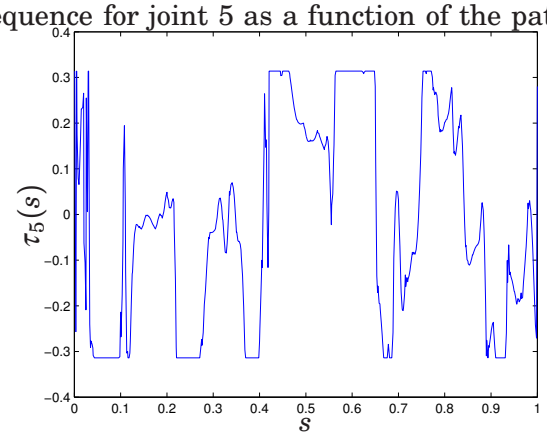

Figure B.7 Optimal input sequence *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3. Joint 5.

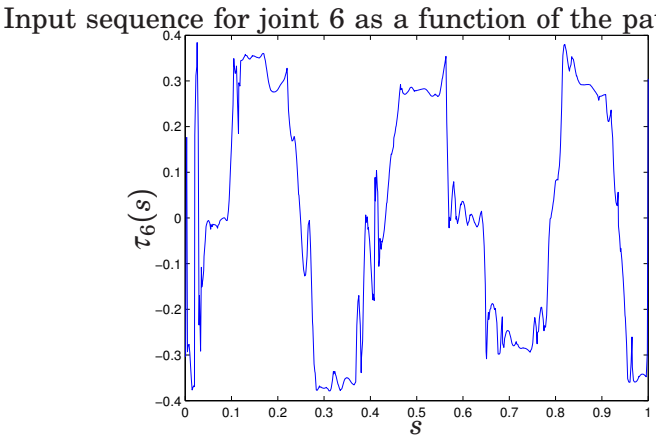

Figure B.8 Optimal input sequence *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3. Joint 6.

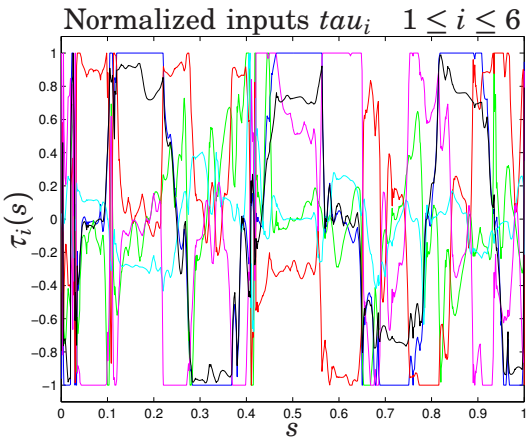

Figure B.9 Optimal normalized input sequences *tau* for the *Finding the Minimum Travesal Time* problem in Section 4.3.

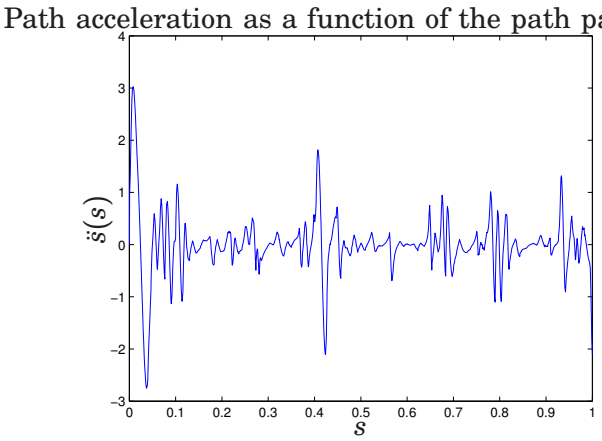

Figure B.10 Optimal path acceleration profile for the *findNominalTraj* problem in Section 4.3

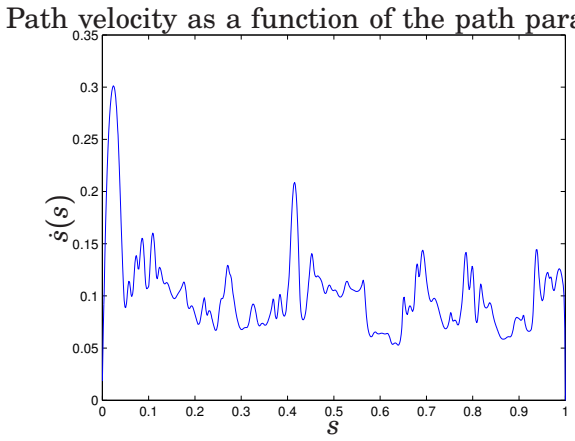

Figure B.11 Optimal path velocity profile for the *findNominalTraj* problem in Section 4.3

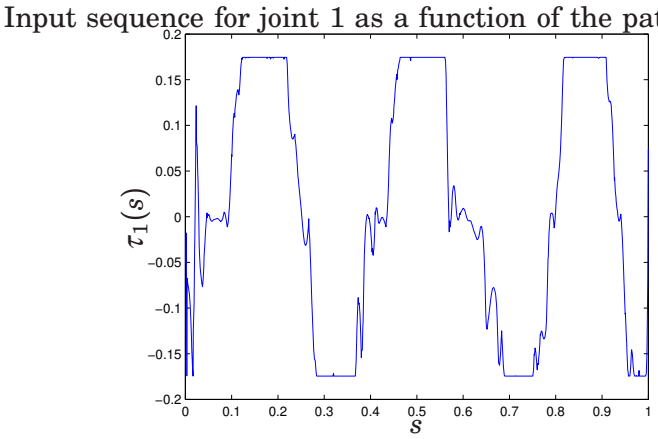

Figure B.12 Optimal input sequence *tau* for the *findNominalTraj* problem in Section 4.3. Joint 1.

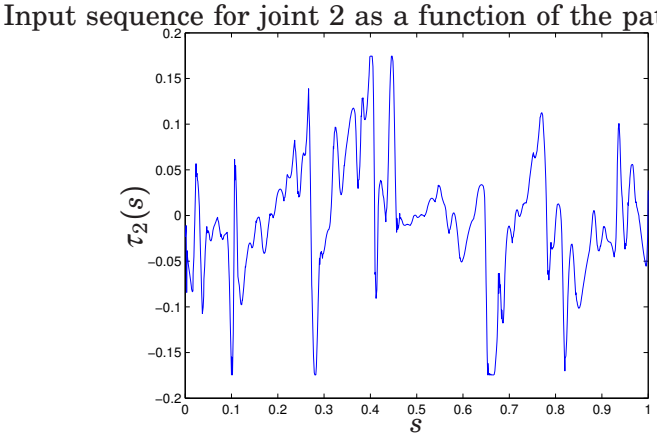

Figure B.13 Optimal input sequence *tau* for the *findNominalTraj* problem in Section 4.3. Joint 2.

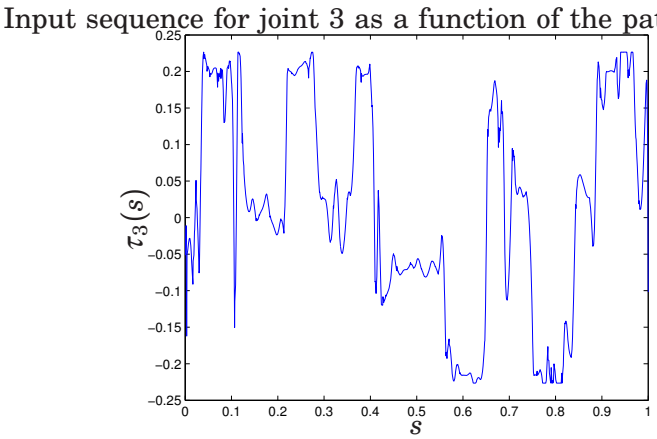

Figure B.14 Optimal input sequence *tau* for the *findNominalTraj* problem in Section 4.3. Joint 3.

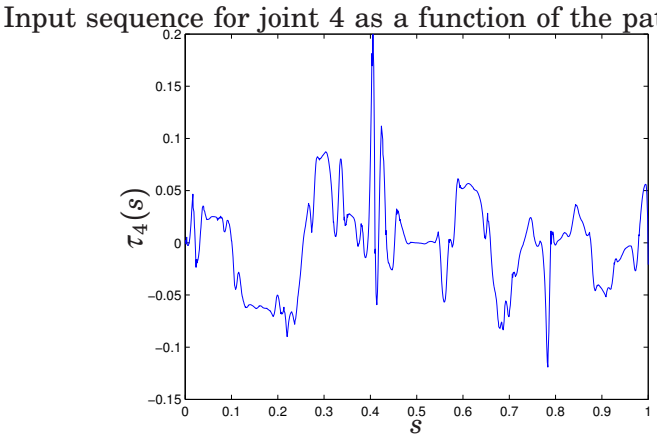

Figure B.15 Optimal input sequence *tau* for the *findNominalTraj* problem in Section 4.3. Joint 4.

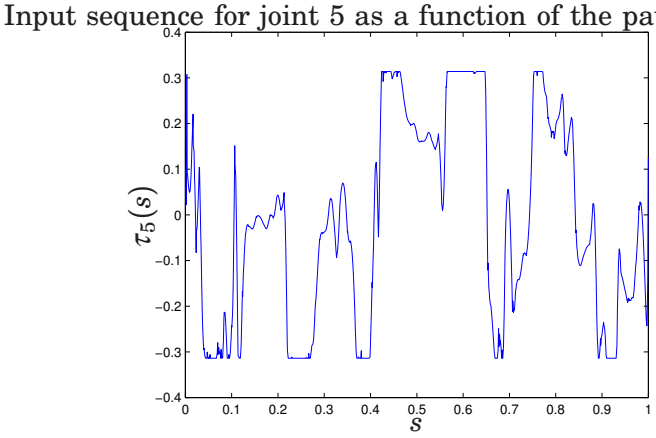

Figure B.16 Optimal input sequence *tau* for the *findNominalTraj* problem in Section 4.3. Joint 5.

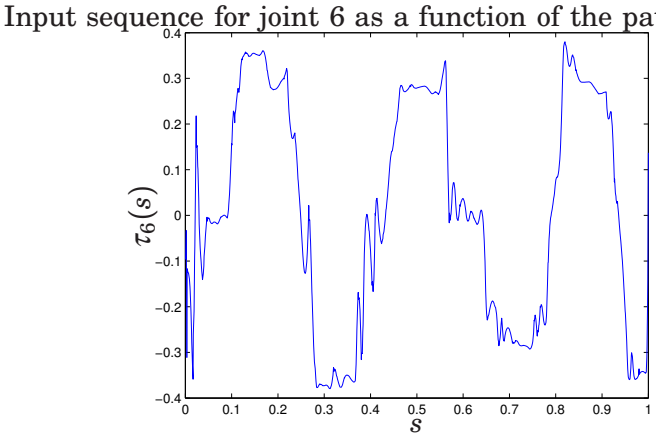

Figure B.17 Optimal input sequence *tau* for the *findNominalTraj* problem in Section 4.3. Joint 6.

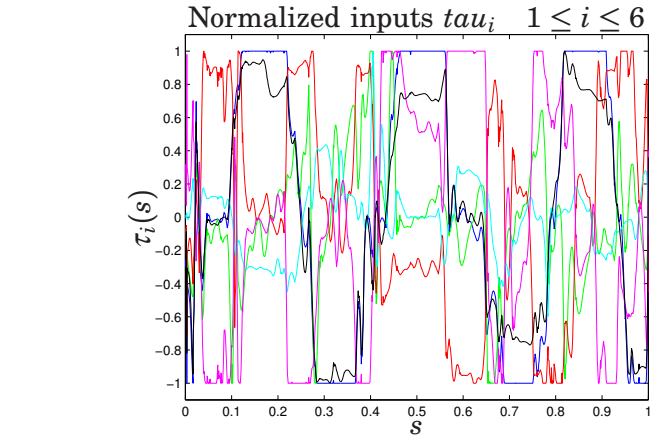

Figure B.18 Optimal normalized input sequences *tau* for the *findNominalTraj* problem in Section 4.3.

C. Simulink® models

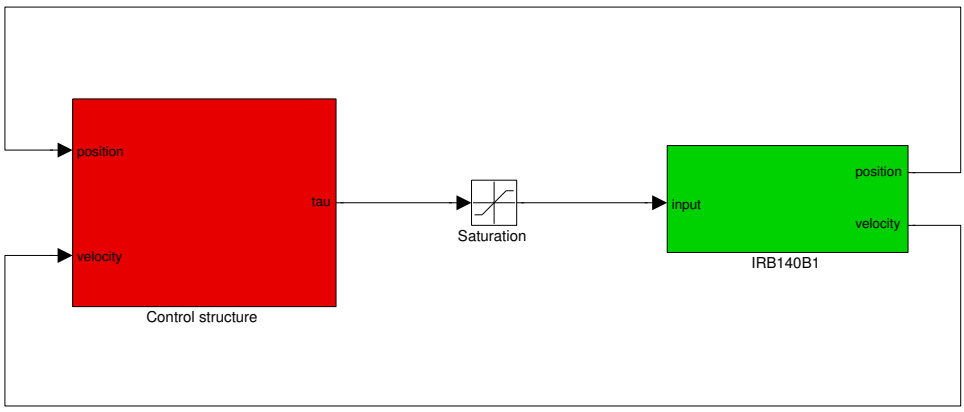

Figure C.1 The block containing the full controller (red) to the left connected to the robot model (green) via a saturation block

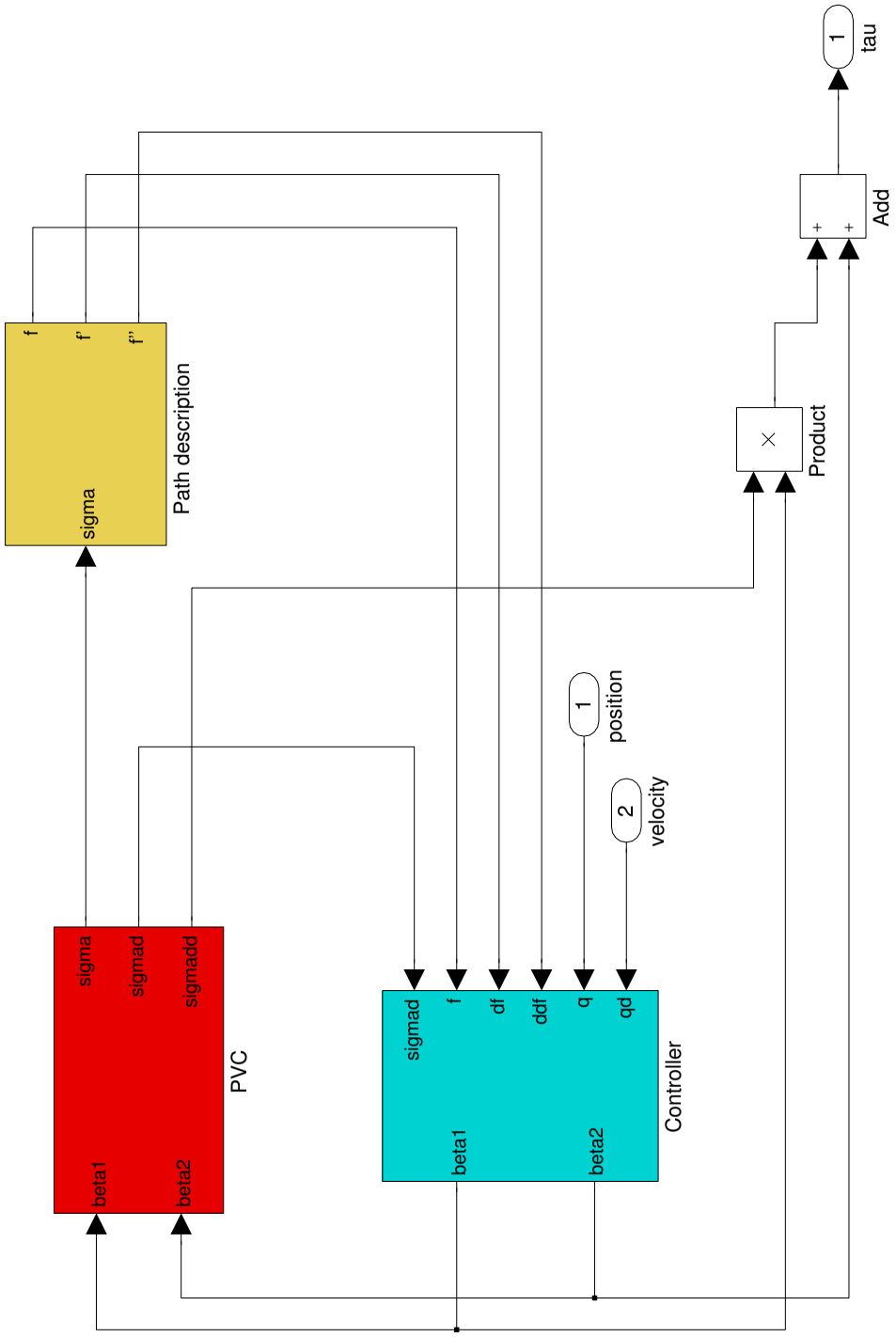

Figure C.2 The control structure inside the full controller block shown in figure C.1.

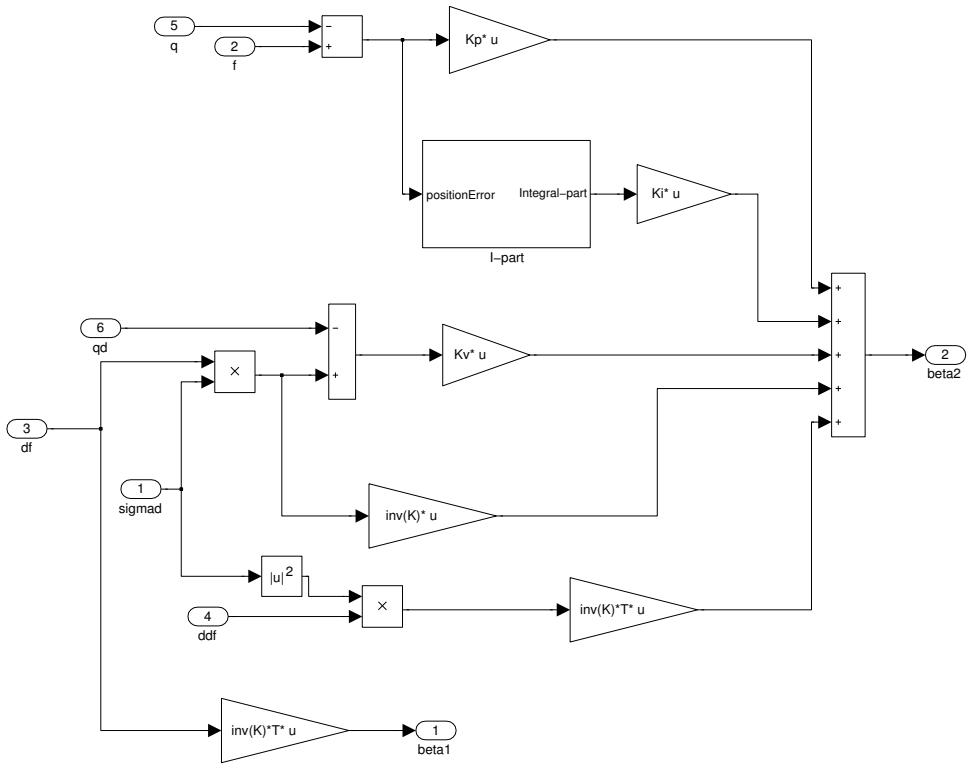

Figure C.3 Controller used to calculate β_1 and β_2.

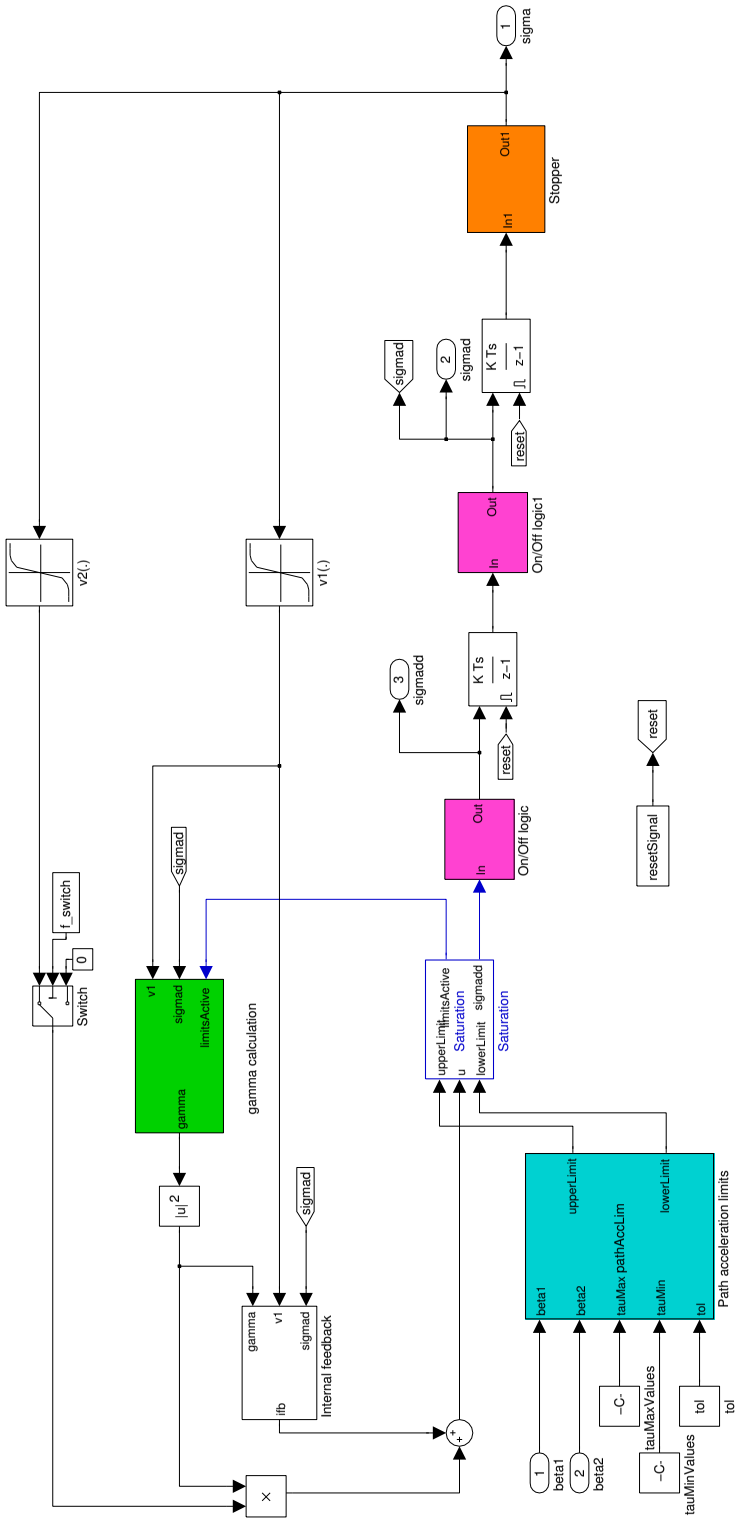

Figure C.4 The Path Velocity Controller.

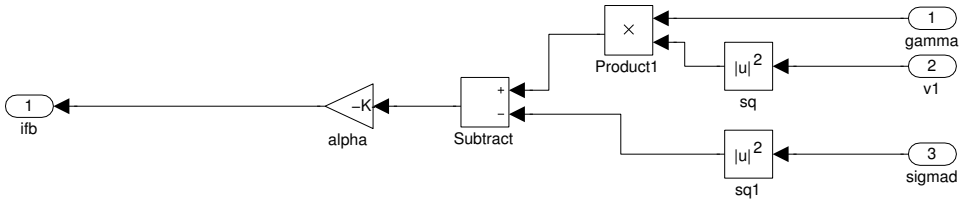

Figure C.5 The internal feedback.

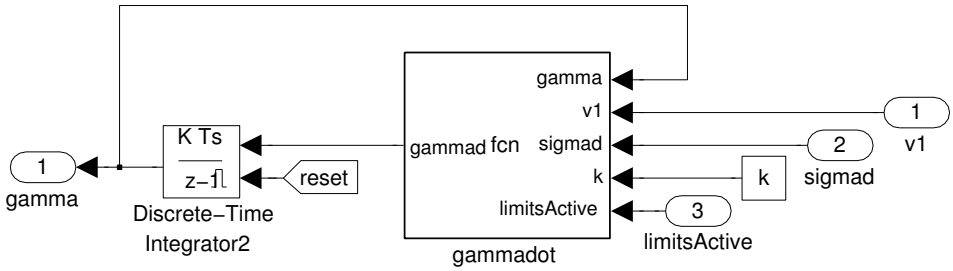

Figure C.6 Gamma calculation block.

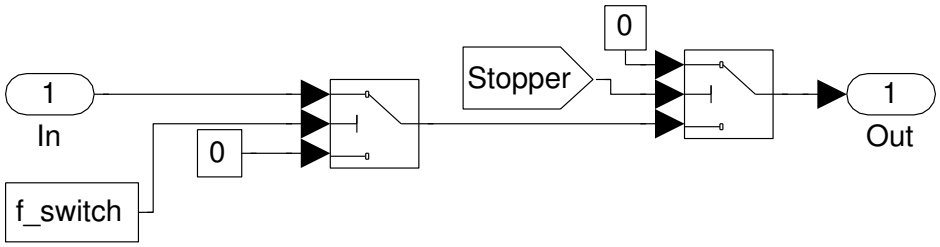

Figure C.7 On/Off logic connected to the integrators in the PVC.

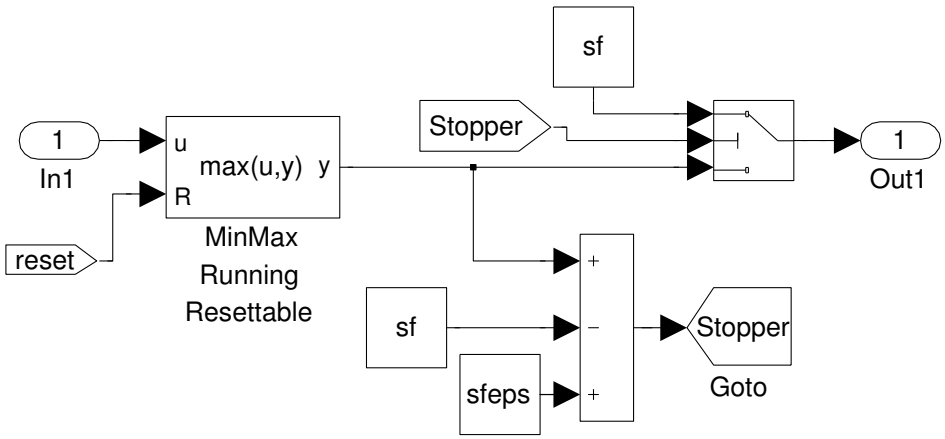

Figure C.8 Stopper block

D. Modelica and Optimica Code

D.1 Modelica Code

```
model Optimization "Formulation of the time minimum
                    optimization problem"
  // Declareation of model parameters and inputs
  parameter Real K1 = 1.031;
  parameter Real K2 = 1.077;
  parameter Real K3 = 1.061;
  parameter Real K4 = 1.051;
  parameter Real K5 = 1.0543;
  parameter Real K6 = 1.062;
  parameter Real T1 = 0.019086;
  parameter Real T2 = 0.020433;
  parameter Real T3 = 0.019129;
  parameter Real T4 = 0.017158;
  parameter Real T5 = 0.017909;
  parameter Real T6 = 0.017447;
  Real tau1;
  Real tau2;
  Real tau3;
  Real tau4;
  Real tau5;
  Real tau6;
  // Declareation of the t_f variable
  Real tf(start=0);
  // Dynamics definiton
  Real sd(start=0);
  Real sdd;
  Real x1;
  // Path definition
  Splines path;
  Real df1;
  Real df2;
  Real df3;
  Real df4;
  Real df5;
  Real df6;

  Real ddf1;
  Real ddf2;
  Real ddf3;
  Real ddf4;
  Real ddf5;
  Real ddf6;
  // Optimization input
  Modelica.Blocks.Interfaces.RealInput u
```

```
equation
  // Calculation of t_f
  der(tf) = if sd <= 0.000000000001 then 0 else 1/sd;

  // Caclulation of the derivatives of the
  // path with respect to s
  df1 = der(path.f1);
  df2 = der(path.f2);
  df3 = der(path.f3);
  df4 = der(path.f4);
  df5 = der(path.f5);
  df6 = der(path.f6);

  ddf1 = der(df1);
  ddf2 = der(df2);
  ddf3 = der(df3);
  ddf4 = der(df4);
  ddf5 = der(df5);
  ddf6 = der(df6);

  // Constraints
  K1*tau1 = T1*(ddf1*sd^2 + df1*sdd) + df1*sd;
  K2*tau2 = T2*(ddf2*sd^2 + df2*sdd) + df2*sd;
  K3*tau3 = T3*(ddf3*sd^2 + df3*sdd) + df3*sd;
  K4*tau4 = T4*(ddf4*sd^2 + df4*sdd) + df4*sd;
  K5*tau5 = T5*(ddf5*sd^2 + df5*sdd) + df5*sd;
  K6*tau6 = T6*(ddf6*sd^2 + df6*sdd) + df6*sd;

  // Link the splines parameter t__ to
  // the Modelica built-in variable time
  path.t__ = time;

  // Dynamics
  x1=sd^2/2;
  der(x1) = u;
  sdd = u;

end Optimization;
```

Listing D.1 Optimization model written in Modelica as described in Section 4.3

```
class Splines
  Real t__(start=0);
  Real f1 =  if (t__ >= 0 and t__ < 0.4931) then ...
  Real f2 =  if (t__ >= 0 and t__ < 0.4931) then ...
  Real f3 =  if (t__ >= 0 and t__ < 0.4931) then ...
  Real f4 =  if (t__ >= 0 and t__ < 0.4931) then ...
  Real f5 =  if (t__ >= 0 and t__ < 0.4931) then ...
  Real f6 =  if (t__ >= 0 and t__ < 0.4931) then ...
```

Listing D.2 Modelica model for the splines. Since the implementation is done using if-clausules only the structure is displayed here.

D.2 Optimica Code

```
class MinTimeOpt
  tau1(lowerBound=-3.49*0.05, upperBound=3.49*0.05);
  tau2(lowerBound=-3.49*0.05, upperBound=3.49*0.05);
  tau3(lowerBound=-4.53*0.05, upperBound=4.53*0.05);
  tau4(lowerBound=-6.28*0.05, upperBound=6.28*0.05);
  tau5(lowerBound=-6.28*0.05, upperBound=6.28*0.05);
  tau6(lowerBound=-7.85*0.05, upperBound=7.85*0.05);
optimization
  grid(finalTime = fixedFinalTime(finalTime=1),
                   nbrElements=200);
  minimize(lagrangeIntegrand=1/sqrt(2*x1+1e-10));
subject to
  terminal x1 = 0;
  terminal sd = 0;
  sd >= 0;
end MinTimeOpt;
```

Listing D.3 Optimica file used for finding the minimum traversal time T_f described in Section 4.3

```
class findNominalTraj
  tau1(lowerBound=-3.49*0.05, upperBound=3.49*0.05);
  tau2(lowerBound=-3.49*0.05, upperBound=3.49*0.05);
  tau3(lowerBound=-4.53*0.05, upperBound=4.53*0.05);
  tau4(lowerBound=-6.28*0.05, upperBound=6.28*0.05);
  tau5(lowerBound=-6.28*0.05, upperBound=6.28*0.05);
  tau6(lowerBound=-7.85*0.05, upperBound=7.85*0.05);
optimization
  grid(finalTime = fixedFinalTime(finalTime=1),
                   nbrElements=300);
  minimize(lagrangeIntegrand=der(sdd)^2 + 1*(der(tau1)^2
  + der(tau2)^2 + der(tau3)^2 + der(tau4)^2 + der(tau5)^2
  + der(tau6)^2));
subject to
  terminal tf = 10.653*1.02;
  terminal x1 = 0;
  terminal sd = 0;
  sd >= 0;
end findNominalTraj;
```

Listing D.4 Optimica file used for finding smooth trajectories described in Section 4.3

D.3 MATLAB® code

Listing D.5 Code for calculation of the Path Acceleration Limits. Found in Section 3.2

```
function [upperLimit,lowerLimit] =
          pathAccLim(beta1,beta2,tauMax,tauMin,tol)
% This block calculates the upper and lower bound on
% the path acceleration so that the input limits are
% not violated.

nbrJoints = 6;
```

```
tmpMax = zeros(nbrJoints,1);
tmpMin = zeros(nbrJoints,1);

for i=1:nbrJoints
    if(beta1(i) > tol)
        tmpMax(i) = (tauMax(i) - beta2(i)) / beta1(i);
        tmpMin(i) = (tauMin(i) - beta2(i)) / beta1(i);
    elseif(beta1(i) < -tol)
        tmpMax(i) = (tauMin(i) - beta2(i)) / beta1(i);
        tmpMin(i) = (tauMax(i) - beta2(i)) / beta1(i);
    else
        tmpMax(i) = 1000000;
        tmpMin(i) = -1000000;
    end
end

upperLimit = min(tmpMax);
lowerLimit = max(tmpMin);
```

Listing D.6 Code for Saturation block. Found in Section 3.2

```
function [limitsActive,sigmadd] =
          Saturation(upperLimit, u, lowerLimit)
% This block limits the input u between upperLimit and
% lowerLimit. If the input is greater than upperLimit or
% smaller than lowerLimit, limitsActive is equal to 1.
% If lowerlimit is greater than upperLimit then
% limitsActive= 0 and sigmadd=u;

if lowerLimit > upperLimit
    sigmadd = u;
    limitsActive = 0;
else
    if u < lowerLimit
        sigmadd = lowerLimit;
        limitsActive = 1;
    elseif u > upperLimit
        sigmadd = upperLimit;
        limitsActive = 1;
    else
        sigmadd = u;
        limitsActive = 0;
    end
end
```

Listing D.7 Code for Velocity Profile Scaling. Found in Section 3.2

```
function gammad = fcn(gamma,v1,sigmad,k,limitsActive)
% Calculates the time derivative of gamma

if gamma*v1 >= sigmad && v1 ~= 0 && limitsActive == 1
    gammad = sigmad*k*(sigmad/v1 - gamma);
else
```

```
    gammad = 0;
end
```