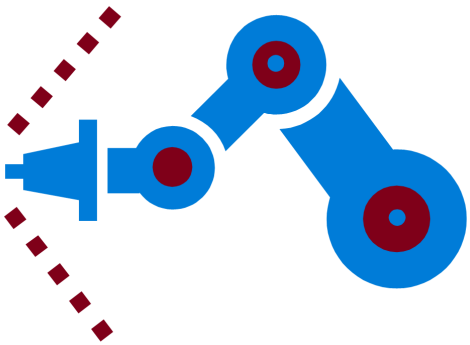
Technical documentation

Modeling and control of an industrial robot

Version 1.0

Author: AP, JK, TA, VI, AG, GA, AS
Date: December 6, 2011



Status

Reviewed	Alexander Pettersson	2011-12-06
Approved	Patrik Axelsson	2011-12-06

Course name:	Control Project Laboratory	E-mail:	toban607@student.liu.se
Project group:	Industrial robot	Document responsible:	AP, JK, TA, VI, AG, GA, AS
Course code:	TSRT10	Author's E-mail:	alepe490@student.liu.se
Project:	Industrial robot	Document name:	tekniskdokumentation.pdf

Project Identity

Group E-mail:	toban607@student.liu.se
Homepage:	http://www.isy.liu.se/edu/projekt/reglerteknik/2011/industrirobot/
Orderer:	Patrik Axelsson, Linköping University
	Phone: 013 284474, **E-mail:** axelsson@isy.liu.se
Customers:	Mikael Norrlöf, ABB Robotics
	Phone: 021 346017 , **E-mail:** mino@isy.liu.se
	Johan Sjöberg, ABB Corporate Research
	Phone: 013 284028 , **E-mail:** johans@isy.liu.se
Course Responsible:	David Törnqvist, Linköping University
	Phone: 013 281882, **E-mail:** tornqvist@isy.liu.se
	Daniel Axehill, Linköping University
	Phone: 013 284042, **E-mail:** daniel@isy.liu.se
Project Manager:	Tobias Andersson
Advisor:	André Carvalho Bittencourt, Linköping University
	Phone: 013 282622 , **E-mail:** andrecb@isy.liu.se

Group Members

Name	Responsibility	Phone	E-mail (@student.liu.se)
Tobias Andersson (TA)	Project manager	0730530440	toban607
Alexander Pettersson (AP)	Documents	0737767682	alepe490
Jonas Källman (JK)	Design	0739575719	jonka615
Victor Ingeström (VI)	Mechanical modeling	0704926973	vicin977
Kristofer Klasson (KK)	Tests	0738184392	krikl150
Gabriella Ahlbert (GA)	Information	0705911501	gabah362
Andreas Samuelsson (AS)		0730651177	andsa897
Anders Gällsjö (AG)		0733681099	andga726

Document History

Version	Date	Changes made	Sign	Reviewer
0.1	2011-11-29	First draft.	AP	André Carvalho Bittencourt
0.2	2011-12-01	First revision.	AP	Patrik Axelsson
0.3	2011-12-05	Second revision.	AP	Patrik Axelsson
1.0	2011-12-06	First version.	AP	Patrik Axelsson

Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Objectives . . . 1

2 Overview of the system **2**

3 Hardware **3**

4 Modeling **7**

4.1 Geometry . . . 7

4.2 Forward kinematics . . . 8

4.3 Inverse kinematics . . . 10

4.4 Flexible dynamic model . . . 12

4.5 System identification . . . 15

4.5.1 Stiffness and damping . . . 16

4.5.2 Friction . . . 17

5 Control system **18**

5.1 User interface . . . 18

5.2 Trajectory planner . . . 19

5.3 Reference generator . . . 22

5.4 Joint control . . . 22

5.5 Simulation model . . . 24

6 Results **26**

6.1 System identification . . . 26

6.2 Simulation . . . 28

6.3 Real robot . . . 32

6.3.1 Joint control . . . 32

6.3.2 Trajectory following . . . 35

7 Discussion and conclusions **38**

7.1 System identification . . . 38

7.2 Simulation versus reality . . . 38

7.3 Possible improvements . . . 39

References **41**

A Forward kinematics **42**

B Inverse kinematics **42**

C Flexible dynamic model **44**

D Trajectory planner **49**

E Functions in the simulation model **51**

1 Introduction

Industrial robots are widely used in industrial applications and more and more robots are installed every year. They are used in several industrial applications, from packaging and assembly to painting and welding. Applications of industrial robots require that the robot has a high precision which is why a good control system is important. By modeling the kinematics and dynamics of the robot, a motion controller and a trajectory planner have been implemented in this project. The robot used in this project was made in 2010 by Mechanical Engineering students at the Department of Management and Engineering, IEI, at Linköping University. The robot is made mostly in plastic, with a few gears in metal, see Figure 1 for a picture of the robot.

1.1 Background

This report is a part of the project course TSRT10 given by the Division of Automatic Control at Linköping University. The purpose of the course is to get experience of working in a real project related to automatic control while using knowledge from previous courses.

1.2 Objectives

The main objectives are to design a motion controller and a trajectory planner for the robot, which has three degrees of freedom. The robot should be able to do two different movements, joint movement and trajectory movement. A joint movement means that the arms go from one set of joint angles to another. A trajectory movement means that the tool position moves in a straight line from point A to point B, where A and B are two positions in room coordinates. The requirements for the controller are that the overshoot for a step in the desired joint angles should not exceed 10%. The stationary error should be less than 10% for both a joint movement and a trajectory movement.

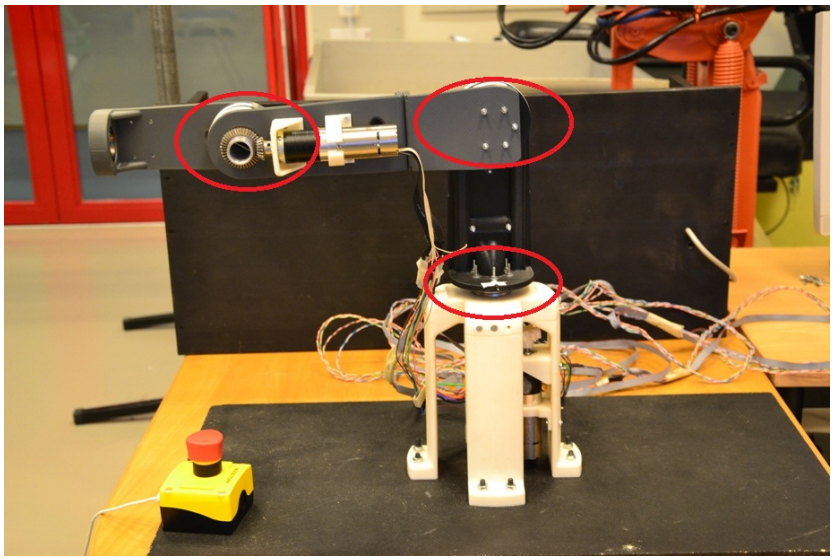


Figure 1: Picture of the robot, which has three axes, actuated with brushless DC motors.

2 Overview of the system

The system consists of a miniature industrial robot connected to a computer with a software interface via control electronics, see Section 3, models and a control system. The control electronics consist of hardware from National Instruments and are used to communicate with the robot from the computer. The user interface is made in National Instruments (NI) Labview 2010 [1] which is also used to control the robot. An overview of the system is shown in Figure 2. The robot was designed and built in a previous student project by Mechanical Engineering students, which means that the design of the models and the control system will be the focus of this document.

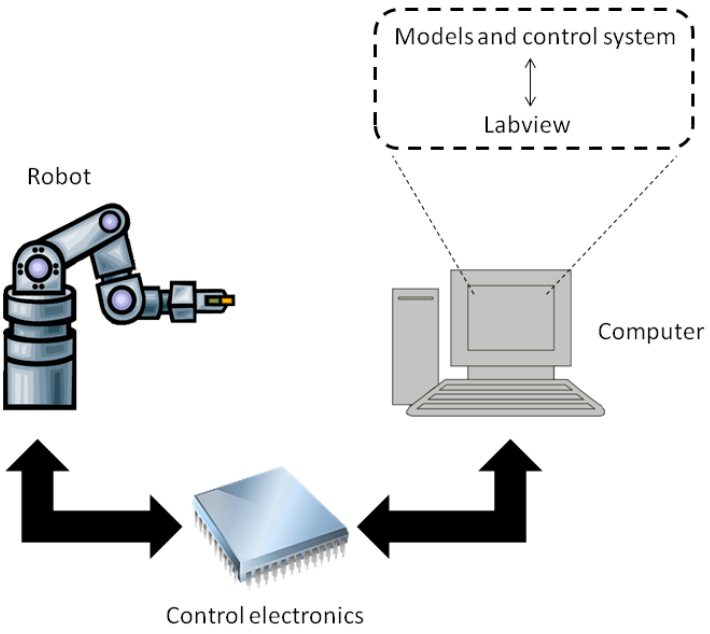


Figure 2: An overview of the system.

The robot has three revolute joints actuated with brushless DC motors. With this configuration, three degrees of freedom of the end-effector can be controlled, i.e. the three joints marked in Figure 1. The choice is to manipulate the tool position but not its orientation. A kinematic model, a flexible dynamic model and a controller for the robot are achieved. The solution was first implemented in Matlab/Simulink [2][3]. The control system and a user interface were later implemented in Labview.

3 Hardware

The hardware consists of several parts, listed below.

- The miniature robot.
- NI cRIO-9022 Real-Time Controller.
- NI 9514 C Series Servo Drive Interface with Encoder Feedback.
- Faulhaber Type BLD 5603 Servo Amplifiers.
- Faulhaber 3242G024BX4 Brushless DC Motors.

The robot is placed on a platform with four contact points, see Figure 1. The arms and other details are mainly made of plastic, printed in a 3D printer. The robot was originally constructed with six joints, however this project will just focus on the first three. Therefore, the three last axes have been removed from the robot.

The joints are controlled by brushless DC motors, of type Faulhaber 3242G024BX4 [4], see Figure 3. They have a nominal voltage of 24 V and a maximum shaft load of 50 Nm. The motors have three phases as inputs and voltage measurements from three Hall sensor as outputs. The signals are amplified and measured by Faulhaber Type BLD 5603 Servo Amplifiers [5], see Figure 4. Each motor has its own servo amplifier.

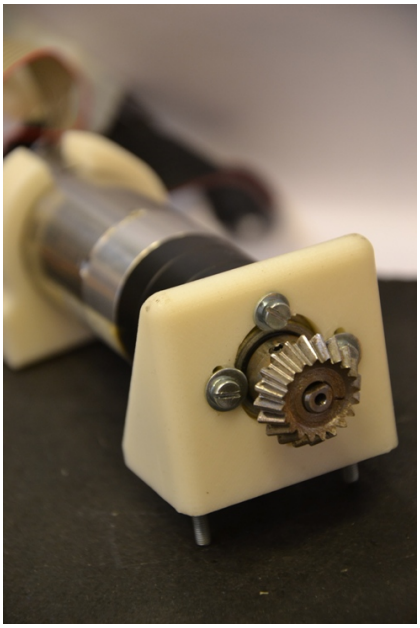


Figure 3: Picture of a Faulhaber 3242G024BX4 Brushless DC Motor.

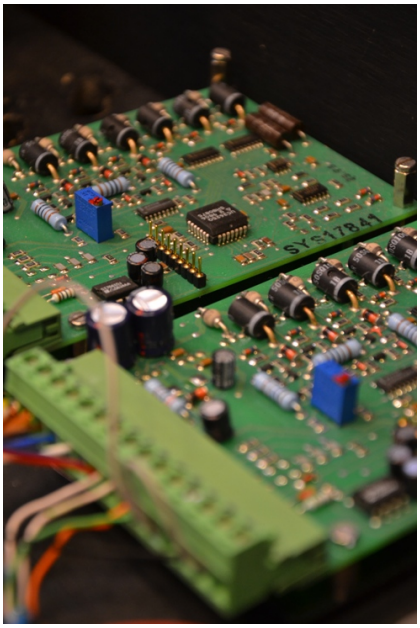


Figure 4: Picture of Faulhaber Type BLD 5603 Servo Amplifiers.

Each servo amplifier is connected to a NI 9514 module [6], see Figure 5. These modules run Labview NI Softmotion [7], and allows position and velocity control of one axis. In Figure 6, the axis configuration window is shown. Here one can change settings for the PID parameters, motor setup etc. Consequently, the current and the joint control are already implemented in the hardware. Unfortunately, the NI 9514 can not be set to run in open loop mode.

Figure 5: Picture of the NI 9514 modules that control the axes.

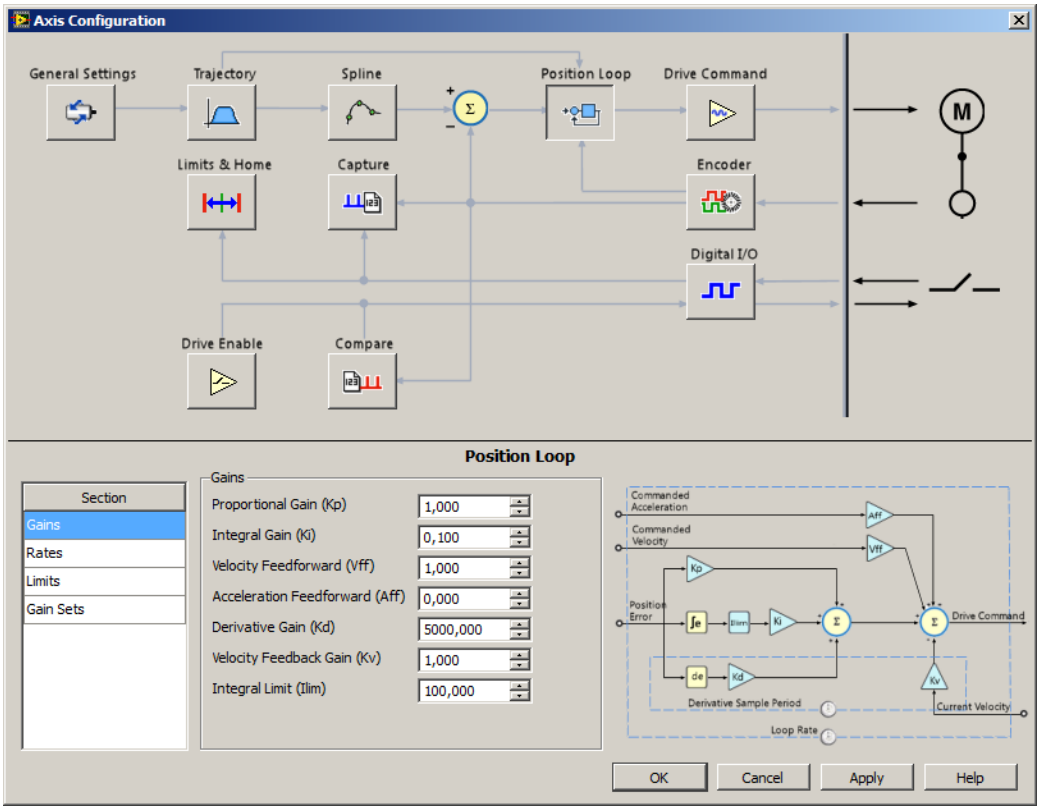


Figure 6: The axis configuration window.

The NI 9514 modules are controlled by a NI cRIO-9022 Real-Time Controller [8], see Figure 7. The NI cRIO-9022 has an Ethernet port which makes it possible to connect it to a computer with Labview installed. Furthermore, logged data will be stored at the controller's own memory. Logged data can be downloaded to a computer over the network.

Figure 7: Picture of the NI cRIO-9022 Real-Time Controller.

4 Modeling

In this section, the different models will be explained. The different models relate to a kinematic description, a flexible dynamic model and to the joint friction.

4.1 Geometry

First, the geometry of the robot has to be defined. In Figure 8, the coordinate systems are defined (in a CAD drawing) and in Figure 9 the arm lengths are defined. All coordinate systems are Cartesian right-handed systems. $\{A\}$ is the fixed room coordinate system which the tool position will be given in. The coordinate system $\{B\}$ has its origin in the same point as $\{A\}$ and they share the z-axis. However, $\{B\}$ rotates around the z-axis and the angle between the x- and y-axes of $\{A\}$ and $\{B\}$ is denoted by θ_1. The coordinate system $\{C\}$ has its origin at the bottom of the second arm and is fixed in this arm. θ_2 is the angle between the x_B- and x_C-axis. The x-axis of $\{C\}$ is fixed along the arm and the y-axis is perpendicular to it. Furthermore, the second joint rotates around the z-axis of $\{C\}$. The coordinate system $\{D\}$ has its origin at the bottom of the third arm. Similar to $\{C\}$, the x_D-axis is fixed in the arm and the third joint rotates around the z_D-axis. The angle difference between the x_C- and x_D-axis is denoted by θ_3. $\{E\}$ has the same orientation as $\{D\}$ but has its origin at the top of third arm, and serves as the coordinate system of the tool.

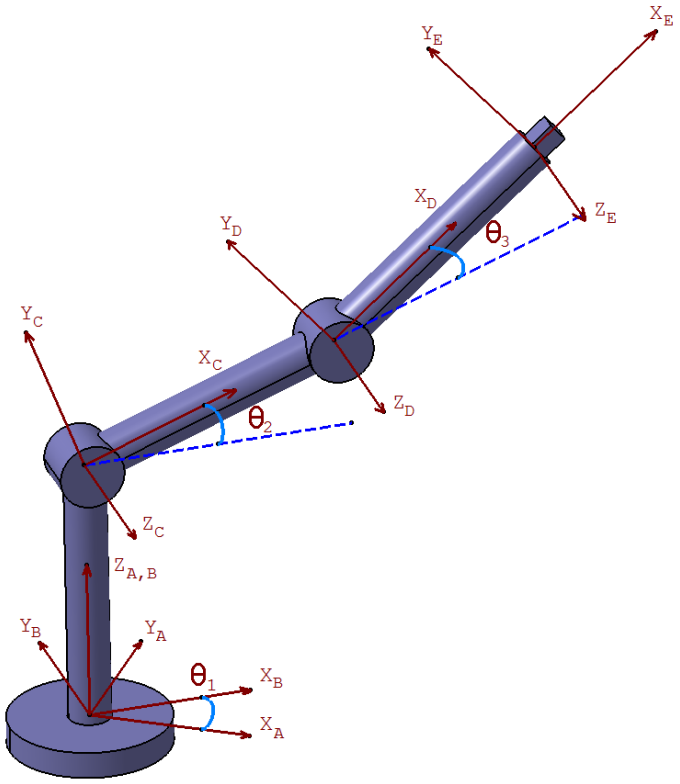


Figure 8: The geometry of the robot.

In fact, the second arm is misaligned of the second and third joint, thus creating a displacement in the y_A-direction. The third joint however displaces the third arm the same distance in the opposite direction, so that the second and third joint are aligned in the x_C-axis. Consequently, the geometry can be regarded as the one defined in Figure 8.

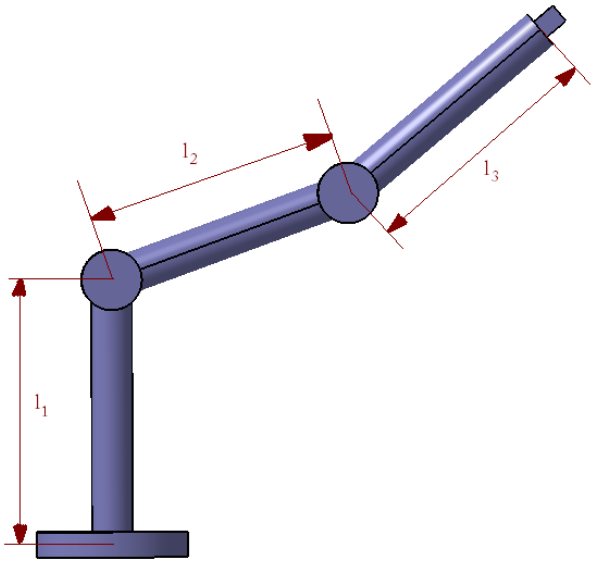


Figure 9: Definition of arm lengths.

4.2 Forward kinematics

The forward kinematics is the problem of finding the tool position in room coordinates given the angles of the joints [9]. The orientation of a coordinate system (CS) $\{B\}$ relative to another CS $\{A\}$ can be described by a rotational matrix,

$$ {}^A_B R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{4.1} $$

In the general case, with the notation $\cos(x) = c(x)$ and $\sin(x) = s(x)$, the rotational matrix looks like

$$ {}^A_B R(\gamma, \beta, \alpha) = \begin{bmatrix} c(\alpha) & -s(\alpha) & 0 \\ s(\alpha) & c(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} c(\beta) & 0 & s(\beta) \\ 0 & 1 & 0 \\ -s(\beta) & 0 & c(\beta) \end{bmatrix} . \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\gamma) & -s(\gamma) \\ 0 & s(\gamma) & c(\gamma) \end{bmatrix} \tag{4.2} $$

$$ = \begin{bmatrix} c(\alpha)c(\beta) & c(\alpha)s(\beta)s(\gamma) - c(\gamma)s(\alpha) & c(\alpha)c(\gamma)s(\beta) + s(\alpha)s(\gamma) \\ c(\beta)s(\alpha) & s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma) & s(\alpha)c(\gamma)s(\beta) - c(\alpha)s(\gamma) \\ -s(\beta) & c(\beta)s(\gamma) & c(\beta)c(\gamma) \end{bmatrix}, \tag{4.3} $$

where α, β and γ is the rotation around the defined x-, y- and z-axis. In Figure 8, each

joint only rotates in one direction. This means that only one of the angles are variable and the other two are constants.

If a coordinate system is both translated and rotated relative to another, this can be described by a homogeneous transformation,

$$ {}^A_B T = \begin{bmatrix} {}^A_B R & {}^A P_T \\ 0 & 1 \end{bmatrix}, \tag{4.4} $$

where ${}^A P_T$ is the translation of the origin of $\{B\}$ expressed in $\{A\}$ as a 3×1-vector. With this notation, a point P in CS $\{B\}$ can be expressed in $\{A\}$ by

$$ {}^A P = {}^A_B T \, {}^B P, \tag{4.5} $$

where

$$ P = \begin{bmatrix} P_{xyz} \\ 1 \end{bmatrix}. \tag{4.6} $$

P_{xyz} in (4.6) are the room coordinates.

For the geometry defined in Section 4.1, the transformation matrices for the coordinate systems become

$$ {}^A_B T = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.7} $$

$$ {}^B_C T = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.8} $$

$$ {}^C_D T = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & l_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.9} $$

and

$$ {}^D_E T = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.10} $$

where l_i, $i = 1, 2, 3$, are the arm lengths. The tool position in room coordinates is given by

$$ {}^A P = {}^A_E T \, {}^E P = {}^A_B T \, {}^B_C T \, {}^C_D T \, {}^D_E T \, {}^E P, \tag{4.11} $$

where ${}^E P$ is the tool center position in CS $\{E\}$. For example, with

$$ {}^E P = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T, \tag{4.12} $$

the tool center position as a function of the joint angles can be determined from (4.11) according to

$$ {}^{A}P = \begin{bmatrix} \cos\theta_1(\cos\theta_2(l_3\cos\theta_3 + l_2) - l_3\sin\theta_2\sin\theta_3) \\ \sin\theta_1(\cos\theta_2(l_3\cos\theta_3 + l_2) - l_3\sin\theta_2\sin\theta_3) \\ l_3\cos\theta_2\sin\theta_3 + \sin\theta_2(l_3\cos\theta_3 + l_2) + l_1 \\ 1 \end{bmatrix}. \tag{4.13} $$

For an implementation of the forward kinematics in Matlab code, see Appendix A. Note that θ is denoted in degrees in the code.

4.3 Inverse kinematics

The inverse kinematics is the problem of finding the possible joint angle combinations to reach a given point $P_{xyz} = \begin{bmatrix} x & y & z \end{bmatrix}^T$. For every reachable point except where $x = y = 0$ there are two possible values of θ_1; one where the x-axis in CS $\{B\}$ is directed towards the point and one where it is directed away from the point. For the special case $x = y = 0$ the point is reachable for any values of θ_1.

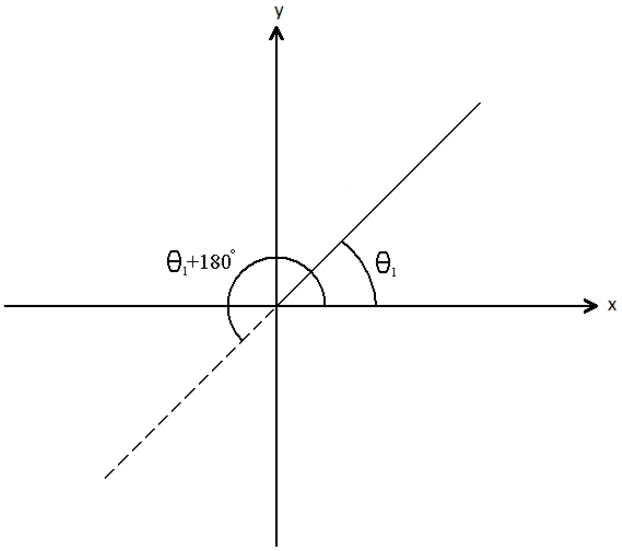


Figure 10: The two possible sets of θ_1.

As seen in Figure 10, θ_1 can be expressed as

$$ \theta_1 = \operatorname{atan2}(y, x), \tag{4.14} $$

where atan2 is a Matlab command that keeps track of which quadrant the point is in. For every value of θ_1 there are two possible sets of θ_2 and θ_3, except the case where $\theta_3 = 0$, then there is only one set of angles for every θ_1. This means that there are four possible sets of angles in most cases. To derive θ_2 and θ_3 help angles and vectors are need to be defined, as shown in Figure 11.

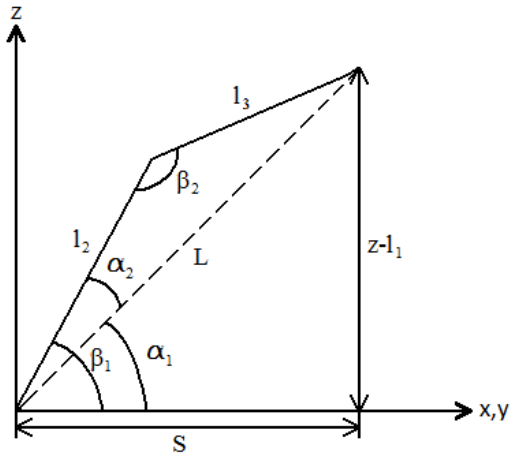


Figure 11: Definition of the help angles. Note that the origin lies in joint 2, i.e. axis 1 is not present.

In the plane shown in Figure 11 the distances S (which is in the xy-plane) and L are calculated as

$$S = \sqrt{x^2 + y^2}, \tag{4.15}$$

and

$$L = \sqrt{S^2 + (z - l_1)^2}. \tag{4.16}$$

According to Figure 11 and the law of cosines, the help angles can be expressed as

$$\alpha_1 = \text{atan2}\left(z - l_1, S\right), \tag{4.17}$$

$$\alpha_2 = \arccos\left(\frac{l_2^2 - l_3^2 + L^2}{2 l_2 L}\right), \tag{4.18}$$

$$\beta_1 = \alpha_1 + \alpha_2 \tag{4.19}$$

and

$$\beta_2 = \arccos\left(\frac{l_2^2 + l_3^2 - L^2}{2 l_2 l_3}\right). \tag{4.20}$$

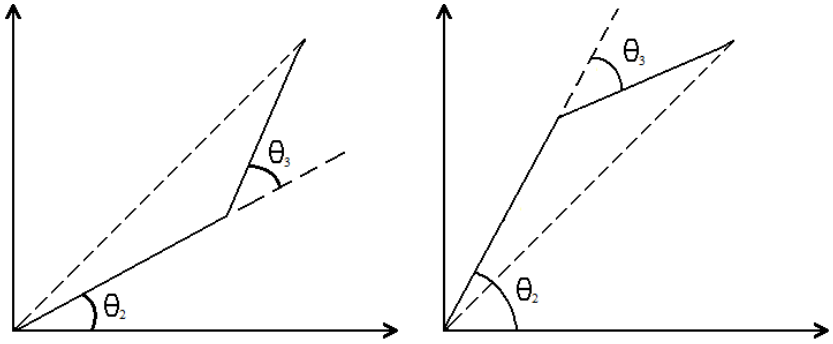


Figure 12: Two possible sets of the angles θ_2 and θ_3.

By combining Figure 11 and 12, where the cases *elbow down* and *elbow up* are shown, one gets

$$\theta_2 = \alpha_1 + \alpha_2 \tag{4.21}$$

and

$$\theta_3 = 180 - \beta_2. \tag{4.22}$$

With these formulas, the joint angles can be calculated from the position and therefore defines the inverse kinematics. For an implementation of the inverse kinematics in Matlab code, see Appendix B. Because some angles are not in the robot's workspace, saturations have been implemented in the code. The joint angles are restricted to

$$-180^\circ \leq \theta_1 \leq 180^\circ \tag{4.23}$$
$$-30^\circ \leq \theta_2 \leq 210^\circ \tag{4.24}$$
$$-135^\circ \leq \theta_3 \leq 135^\circ. \tag{4.25}$$

4.4 Flexible dynamic model

The flexible dynamic model describes the motion of the robot when forces act on it. The model assumes rigid links but flexible joints and is on the form

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) + D(\dot{q}) + K(q) + F(\dot{q}) = \tau, \tag{4.26}$$

where $q = [q_a \;\; q_m]^T$ are the arm and the motor angles, M is the joint-space inertia matrix, C includes Coriolis and centrifugal forces, G includes external forces (gravity), D includes damping, K includes spring forces and F includes friction. The applied torque is τ. [9]

The rigid body contributions to the dynamic model, which are related to the M, C and G matrices, are calculated using the Newton – Euler dynamic formulation as specified in [9]. But the final versions of the M, C and G matrices are calculated using the Lagrangian method and it will be explained in more detail below. In practice the calculations to derive the dynamic model is too complex and time consuming to do by hand. All the calculations needed are done in Matlab using the Symbolic Toolbox [10]. All the robot link properties such as masses, inertia tensor and center of gravity are assumed to be

known and they are extracted from a CAD-model of the robot provided by IEI.

Before starting the calculations, some approximations can be made. One approximation made is that the angular velocity of the rotors is mainly due to their own spinning. This simplifies the dynamic equations considerably as the dynamic couplings between the rotors and their links, only depend on the elastic torque transferred through the gears [11]. This means that when forming the Lagrangian of the system, the potential energy added by the torsion of the joints do not have to be included, and that the flexibilities and the friction can just be added as diagonal matrices when M, C and G are calculated. The main reference when calculating the dynamic model is [12].

The Lagrangian of the system is the difference between the kinetic and potential energy,

$$\mathcal{L} = \mathcal{K} - \mathcal{P}, \tag{4.27}$$

and the equations of motion of the system, called the Euler-Lagrange equations of motion is received by differentiating the Lagrangian as

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_j} - \frac{\partial \mathcal{L}}{\partial q_j} = \tau_j, \tag{4.28}$$

where j relates to the jth generalized coordinate. In this case, the total number of generalized coordinates are six, the arm and motor angles. However, since the M, C and G only depend on the arm angles and because of the approximations made, the Lagrangian can be differentiated only with respect to the arm angles which means that $j = 1, 2, 3$. Equation (4.28) is similar to (4.26) but written on another form. The goal here is to derive M, C and G from (4.28).

The next step is to derive the Jacobians of the robot. The Jacobian matrices are used to calculate the linear and angular velocities of the robot arms and is very important when setting up the kinetic energy of the robot. The angular velocity Jacobians are calculated as

$$J_{\omega 1} = \begin{bmatrix} z_A & 0 & 0 \end{bmatrix}, \tag{4.29}$$

$$J_{\omega 2} = \begin{bmatrix} z_A & z_B & 0 \end{bmatrix} \tag{4.30}$$

and

$$J_{\omega 3} = \begin{bmatrix} z_A & z_B & z_C \end{bmatrix}, \tag{4.31}$$

where the z_n vectors are the z_n-coordinate axis expressed in the world inertial frame $\{A\}$.

The linear velocity Jacobians are calculated as

$$J_{vi} = \frac{\partial^E P}{\partial q_i}. \tag{4.32}$$

The total kinetic energy of the robot is the sum of the translational and rotational kinetic energies and it is calculated as

$$\mathcal{K} = \frac{1}{2}\dot{q}_a^T M \dot{q}_a, \tag{4.33}$$

where the joint-space inertia matrix M is calculated as

$$M(q_a) = \sum_{i=1}^{3} m_i J_{vi}(q_a)^T J_{vi}(q_a) + J_{\omega i}^T R_i(q_a) I_i R_i(q_a)^T J_{\omega i}(q_a), \tag{4.34}$$

where m_i is the mass of link i, I_i is the inertial tensor of link i with respect to its own center of gravity and R_i is the transformation matrix that transforms the inertial tensor to the world inertial frame $\{A\}$.

The C matrix can be calculated from the M matrix by first defining the Christoffel Symbols of the first kind as

$$c_{ijk} = \frac{1}{2}\left[\frac{\partial M_{kj}}{\partial q_i} + \frac{\partial M_{ki}}{\partial q_j} - \frac{\partial M_{ij}}{\partial q_k}\right], \tag{4.35}$$

where M_{ab} is the a-th row and b-th column in the M matrix.

A matrix $\mathcal{C}$ is introduced and it relates to C as

$$C = \mathcal{C}\dot{q}_a. \tag{4.36}$$

The k, j-th element of the matrix $\mathcal{C}$ is calculated as

$$\mathcal{C}_{kj} = \sum_{i=1}^{3} c_{ijk}\dot{q}_{ai}, \tag{4.37}$$

and the C vector is finally calculated with (4.36).

The gravity vector G is calculated by differentiating the potential energy as

$$G(q_a)_i = \frac{\partial \mathcal{P}}{\partial q_i}, \tag{4.38}$$

where the potential energy in this case only depends on the gravitational effect on the robot.

Now when the rigid body contributions to the dynamic model have been derived, only the joint flexibilities and the friction need to be added.

The elastic torque transferred through the gearbox is modeled as

$$K(q_a - q_m), \tag{4.39}$$

where q_a are the arm angles and q_m are the motor angles which are transformed to the arm side via the gear ratio. K is a positive definite, diagonal matrix containing the joint stiffnesses for each gearbox.

The damping in the gearboxes are modeled as

$$D(\dot{q}_a - \dot{q}_m), \tag{4.40}$$

where D is a positive definite, diagonal matrix containing the damping coefficients of the gearboxes.

The friction is modeled as

$$F(\dot{q}_m), \tag{4.41}$$

where F is a positive definite, diagonal matrix containing the friction coefficients. Both the link and motor angle velocities will affect the total friction but for convenience, the total effect of friction is collected and assumed to only be dependent on the motor angle velocities.

Gathering equations (4.26), (4.39), (4.40), (4.41) and rewriting them as one equation explaining the link movement and one equation explaining the motor movement results in

$$M(q_a)\ddot{q}_a + C(q_a, \dot{q}_a) + G(q_a) + K(q_a - q_m) + D(\dot{q}_a - \dot{q}_m) = 0 \tag{4.42}$$

and

$$B\ddot{q}_m - K(q_a - q_m) - D(\dot{q}_a - \dot{q}_m) + F(\dot{q}_m) = \tau, \tag{4.43}$$

where B is a positive definite, diagonal mass matrix containing the moments of inertia for the motors. It is very important to note that both equations (4.42) and (4.43) are described on the link side of the robot. This means that q_m, τ and the moments of inertia in the B matrix must be multiplied with the corresponding gear ratio. One can also note that the sign of the K and D matrices are different in the equations. This is natural considering that the K and D matrices in the arm equation is the reaction of the K and D matrices in the motor equation.

Equations (4.42) and (4.43) are the final dynamic model and system identification is needed to determine the unknown parameters for the stiffness, damping and friction, see Section 4.5.

The dynamic model is an important part of the project as it will be used in many instances later on, e.g. in simulation. To simulate the robot in Simulink, both the arm and motor angle accelerations in (4.42) and (4.43) must be solved by rewriting the equations to

$$\ddot{q}_a = M(q_a)^{-1}(-C(q_a, \dot{q}_a) - G(q_a) - K(q_a - q_m) - D(\dot{q}_a - \dot{q}_m)) \tag{4.44}$$

and

$$\ddot{q}_m = B^{-1}\left(\tau + K(q_a - q_m) + D(\dot{q}_a - \dot{q}_m) - F(\dot{q}_m)\right). \tag{4.45}$$

Equations (4.44) and (4.45) calculates the arm and motor angle accelerations given the current robot configuration and torque input. The arm and motor angles and angle velocities can then be obtained by integrating the angle accelerations. For an implementation of the flexible dynamic model in Matlab code, see Appendix C.

4.5 System identification

System identification is used with the purpose to identify the coefficients for the flexible stiffness, damping and friction in the dynamic model in equations (4.42) and (4.43). The main reference is [13] as the setup considered is very similar to this configuration.

4.5.1 Stiffness and damping

The idea is to perform experiments on the robot using different input signals and measure the calculated torque and angular velocities of the motors. The sets of input-output data are then fitted to a model using the System Identification Toolbox [14] in Matlab and the parameters can then be extracted from it.

Only one axis is moved at a time and the robot is tilted when axis two and three are experimented so that the arms move in a horizontal plane to remove the effects of gravity. The experiments are carried out with the controller active meaning that the input signal is the angle reference. The first set of experiments was done with a white noise signal as input but this was later changed to a smoother signal, a sum of sine-waves.

Insight of the system model is needed if the experiments are to extract the physical parameters. The robot, when undergoing these types of movements, is approximated as a two mass model, see Figure 13.

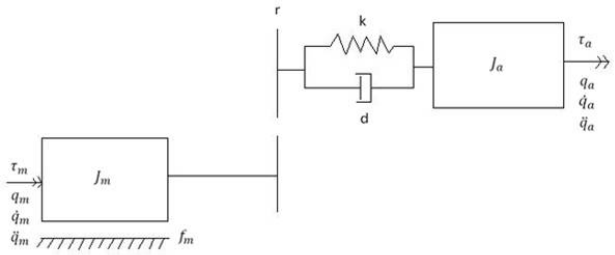


Figure 13: Two mass model, where J_m is the motor inertia, J_a is the arm inertia, k is the gear stiffness, d is the damping and f_m is the friction.

The two mass model is cut in half and a torque balance for the two halves gives a state space model $\dot{x} = Ax(t) + Bu(t)$, $y = Cx(t)$ where

$$x = \begin{bmatrix} rq_m - q_a \\ \dot{q}_m \\ \dot{q}_a \end{bmatrix}, \tag{4.46}$$

$$A = \begin{bmatrix} 0 & r & -1 \\ \frac{-kr}{J_m} & -\frac{dr^2+f_m}{J_m} & \frac{dr}{J_m} \\ \frac{k}{J_a} & \frac{dr}{J_a} & -\frac{d+f_a}{J_a} \end{bmatrix}, \tag{4.47}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{J_m} \\ 0 \end{bmatrix}, \tag{4.48}$$

and

$$C = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \tag{4.49}$$

where $r = 0.5$ is the gear ratio for all axes, and $u = \tau_m$ is the applied torque from the motor.

The System Identification Toolbox is then used to fit the lower 6 elements in the A matrix and the middle element in the B vector to the input-output signals as good as possible.

4.5.2 Friction

The method used to determine the friction parameters is to move one axis at a time at constant angular velocity. This means that the dynamics of the robot simplifies to

$$G(q) + F(\dot{q}) = \tau. \tag{4.50}$$

Equation (4.26) is simplified to (4.50) because $\ddot{q} = 0$, only one axis is moved at a time meaning that $C(q, \dot{q}) = 0$ and because the effects of the flexibilities is rather low during these experiments.

Equation (4.50) is simplified even further by tilting the robot so that it only moves in a horizontal plane, removing the effects of gravity, i.e.

$$F(\dot{q}) = \tau. \tag{4.51}$$

The problem is then to fit the function $F(\dot{q})$.

All the axes are moved with constant angular velocity, the measured torque and two fitted models can be seen in Figures 26, 27 and 28 in Section 6.1. The form of the two models are

$$F(\dot{q}) = a \cdot \text{sign}(\dot{q}_m) + b \cdot \dot{q}_m \tag{4.52}$$

and

$$F(\dot{q}) = c \cdot \dot{q}_m, \tag{4.53}$$

where a, b and c are constants, and they are fitted in Matlab by the least square method.

5 Control system

The control system is used to control the movement of the joints. The controllers are implemented in Simulink along with the models of the system so that testing can be performed before the controller is used on the robot. In Figure 14, a block diagram of the control system is shown. The trajectory planner calculates a reference signal on the form of the desired joint angles, $q_{a,ref}$, and is transformed to reference motor angles, $q_{m,ref}$, by the reference generator. The input to the torque controller are $q_{a,ref}$, $q_{m,ref}$ and the measured motor angles q_m. The torque controller output is the desired torque τ for the motors, which is transformed into a current i by the current controller. This current is applied to the motors to obtain the right angles. The current controller is dashed in Figure 14, because it is already implemented in hardware. Also, this loop is assumed to be much faster than the robot itself, hence the current controller is not modeled. Furthermore, the dynamic model, and therefore the simulation model of the robot, takes torque as input signal. The different blocks in Figure 14 will be explained in more detail later in this chapter.

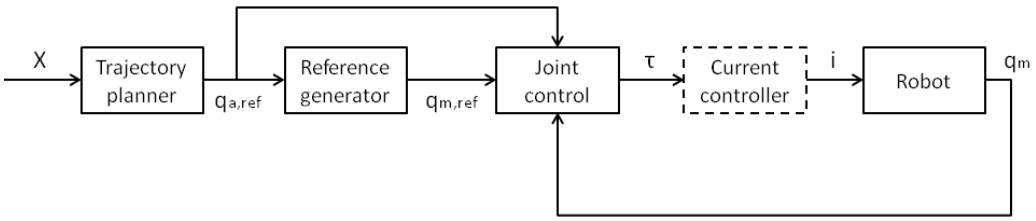


Figure 14: Block diagram of the control system.

5.1 User interface

In Figure 15, a picture of the Labview user interface is shown. In the user interface, an operator can choose between manual control, joint control or trajectory following. Furthermore, graphs of the joint angles and angular velocities are plotted in real time. Also, the calculated tool position is updated in real time. The trajectory calculation and forward kinematics are implemented as Mathscripts, i.e. the code in Appendix A and D. For more information how to use the user interface, see [15].

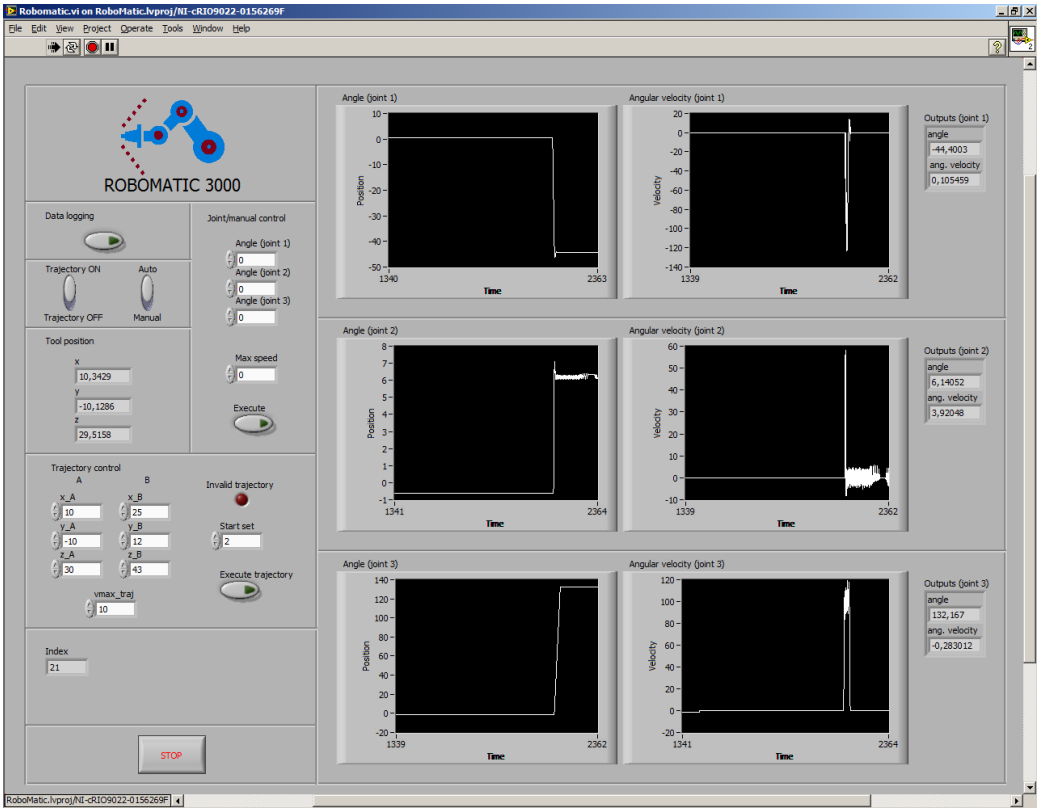


Figure 15: The Labview user interface, with different buttons, graphs and indicators.

5.2 Trajectory planner

The trajectory planner calculates angle references as function of time, so that the robot moves the tool in a straight line from point A to point B in room coordinates. See Figure 16 for a block diagram of the trajectory planner. X are room coordinates and velocity for the trajectory, and q_{ref} is reference joint angles for the joint controller.

Figure 16: Block diagram of the trajectory planner.

The trajectory planner receives a target position and a maximum speed from the user interface. Based on the current position of the robot's tool, the length between position A and B can be calculated (S_{AB}). The maximum speed, that the user states, is used to calculate the final time, t_f. Since the arm is always moving with a speed close to its

maximum, the maximum speed is almost the same as the average speed ($v_{max} \approx v_{average}$) This approximation makes it easy to calculate the final time ($t_f = \frac{S_{AB}}{V_{average}}$).

To get a smooth path trajectory for the tool movement along the line between A and B a Quintic polynomial trajectory is calculated as described in Chapter 5.6.1 in [12]. The resulting polynomial which describes the position along S_{AB} over time can be seen in Figure 17.

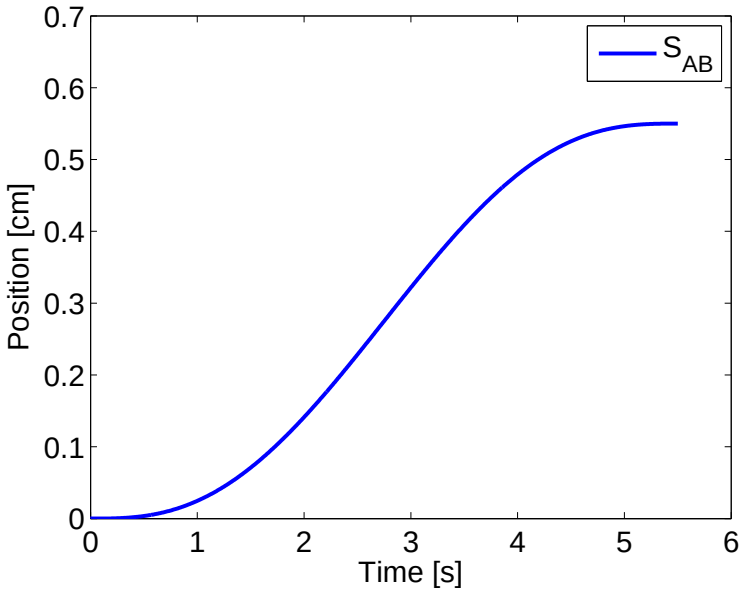


Figure 17: Position along AB as function of time.

For each position in the calculated trajectory the inverse kinematics is used to calculate the corresponding angles. The inverse kinematics generates four different sets of angles. The set that is closest to the previous set is chosen. If one of the calculated positions is invalid the trajectory planner stops the execution and returns a shorter vector. This makes it possible to use the length of the vector as a test for an approved trajectory. The resulting angles as a function of time can be seen in Figure 18, and the trajectory in 3D can be seen in Figure 19. For the implementation of the trajectory planning in Matlab code, see Appendix D.

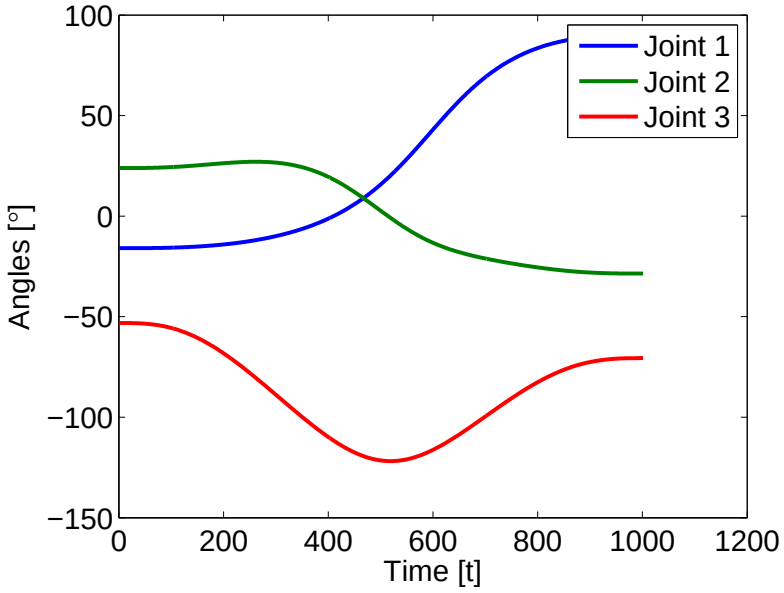


Figure 18: Angles as function of time.

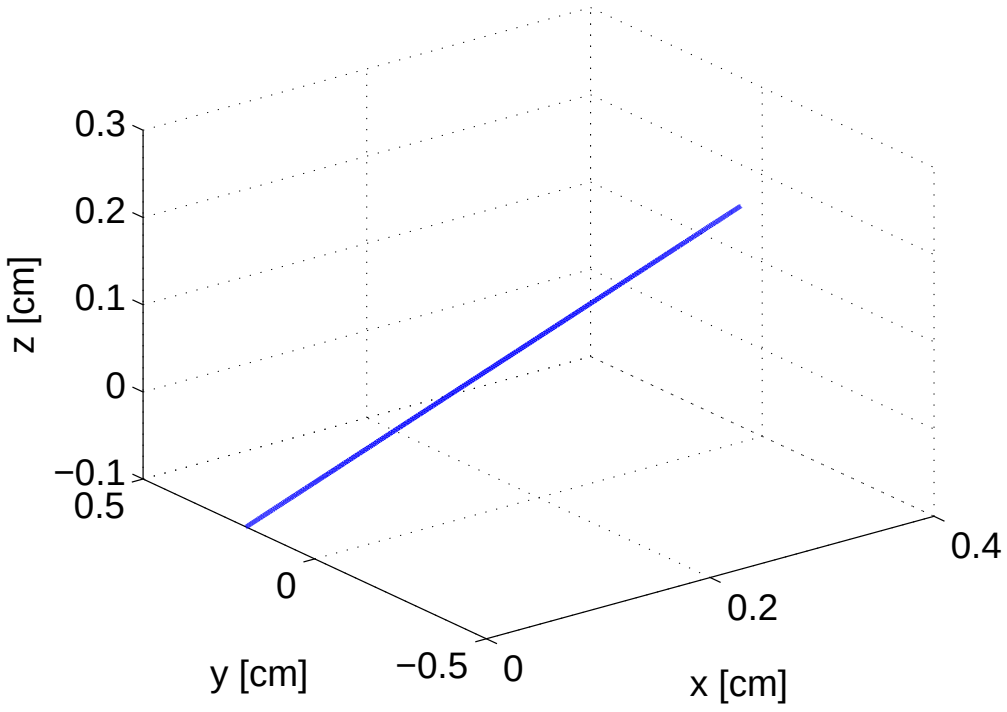


Figure 19: A trajectory between tool position A and B.

5.3 Reference generator

The user will generate reference joint angles, and these must be transformed to motor angles so they can be compared with the measured motor angles in the joint controller. Hence, the flexible dynamic model (see Section 4.4) must be used to do this transformation. Assume that the joint angle reference $q_{a,ref}$ is known, and its time derivatives can be estimated by

$$\dot{q}_{a,ref} = \frac{s}{\alpha s + 1} q_{a,ref} \tag{5.1}$$

and

$$\ddot{q}_{a,ref} = \frac{s}{\alpha s + 1} \dot{q}_{a,ref}, \tag{5.2}$$

where s is the Laplace operator and α is a constant. Now, Equation (4.42) can be rewritten as

$$M(q_{a,ref})\ddot{q}_{a,ref} + C(q_{a,ref}, \dot{q}_{a,ref}) + G(q_{a,ref}) + K(q_{a,ref}) + D(\dot{q}_{a,ref}) = \frac{1}{2}(K + sD)q_{m,ref}, \tag{5.3}$$

which gives

$$q_{m,ref} = 2(K+sD)^{-1}(M(q_{a,ref})\ddot{q}_{a,ref} + C(q_{a,ref}, \dot{q}_{a,ref}) + G(q_{a,ref}) + K(q_{a,ref}) + D(\dot{q}_{a,ref})), \tag{5.4}$$

where s is the Laplace operator. This filter can easily be implemented in Simulink.

5.4 Joint control

The joint controller is used to control the angle of the motors, and consequently also the joint angles. A block diagram of the joint controller is shown in Figure 20.

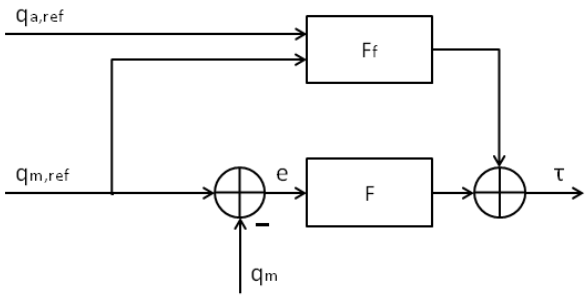


Figure 20: Block diagram of the joint controller.

The control error e is calculated as the difference between the reference and the measured motor angles,

$$e = q_{m,ref} - q_m, \tag{5.5}$$

and is used as input to the feedback controller F. F is a diagonal PID controller and generates a torque as control signal.

The rigid dynamics model (see Section 4.4) is used to feed forward the reference angles $q_{a,ref}$ and $q_{m,ref}$, and the output from the feed forward block (F_f) is therefore also a torque. The output signals from F and F_f is summed to give the total desired torque for the motors. The desired torque will be the input to the next block, the current controller.

With a feed forward loop, the motors generate a torque even when the control error is zero, so that the robot can maintain its angles in steady state without using an integral controller. Furthermore, with a feed forward loop, the feedback controller F can have a simple structure, i.e. a PID controller. The feed forward will contribute to joints 2 and 3, because these two experience a torque generated by the gravitational force. With the reference joint angles $q_{a,ref}$ known, the gravitational torque can be compensated with

$$\tau_2 = \frac{1}{2} G_{a2}(q_{a2}) \tag{5.6}$$

for joint 2 and

$$\tau_3 = \frac{1}{2} G_{a3}(q_{a3}) \tag{5.7}$$

for joint 3, where G_{ai} is row i in G from (4.42), and the halves come from the gear ratio.

On the real system, the feed forward can not be implemented because of the already implemented feedback loop in the hardware, see Figure 21. Hence, only the motor angles and not the torque can be directly controlled. However, the PID parameters, the velocity feedback gain and the velocity and acceleration feed forward in Figure 21 can be tuned in Labview. The Axis configuration window in Figure 6 is a part of the toolbox NI Softmotion [7]. This toolbox calculates a velocity profile for a motor movement via a cubic B spline. Hence, no trajectory control for single joints needs to be designed.

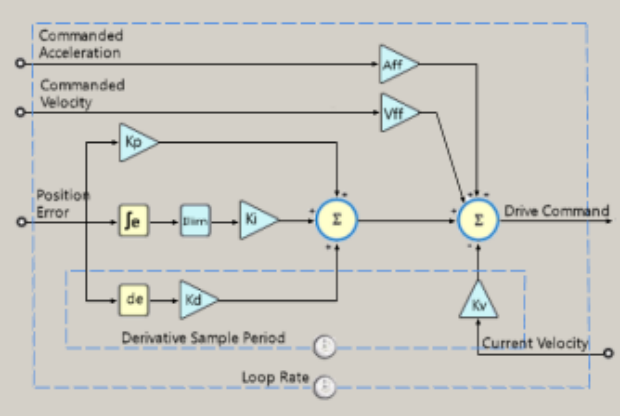


Figure 21: The implemented joint control loop.

5.5 Simulation model

The simulation model for the robot and its control system is implemented in Matlab and Simulink. The numerical solver for the model is ode45 with variable step length and automatic minimum and maximum step lengths, and a relative tolerance of 1e-3. In Figure 22, the top layer of the simulation interface is shown.

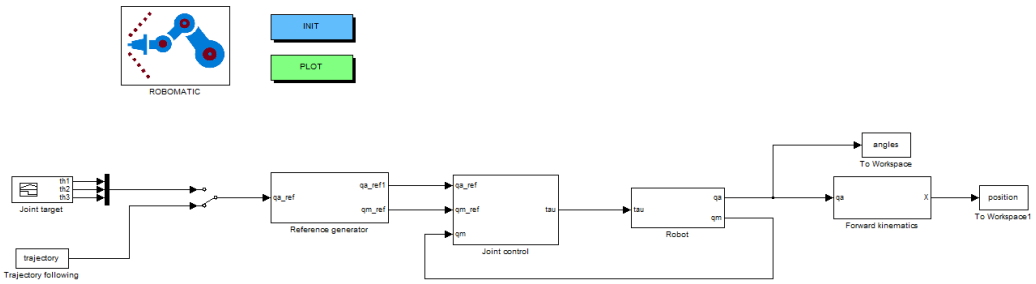


Figure 22: The simulation interface in Simulink.

In Figure 23, the motor angle reference generator is shown in the Simulink diagram. This block converts joint angle references to motor angle references using the flexible dynamic model. See Section 5.3 for an explanation of the reference generator. In Appendix E, the code for the function *qa to qm* is listed.

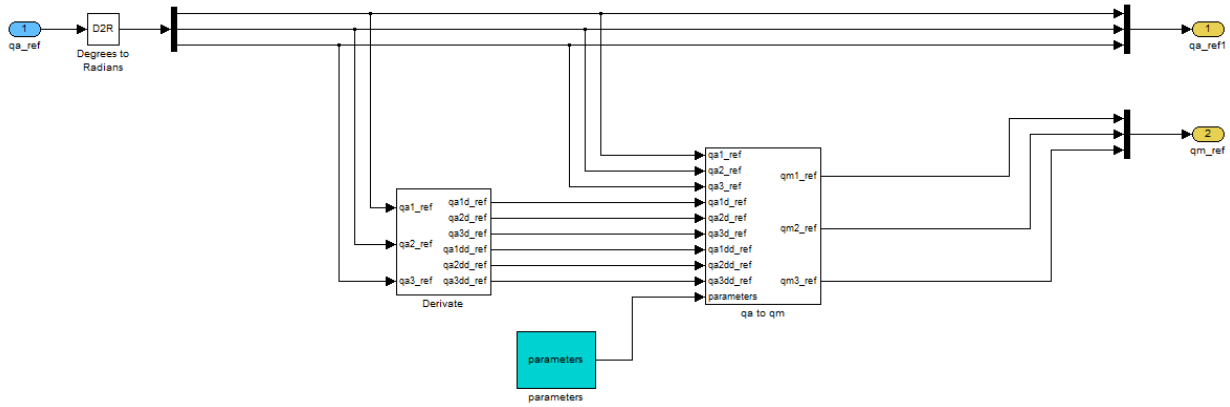


Figure 23: The reference generator implemented in Simulink.

The interior of the *Joint control* block in Figure 22 is shown in Figure 24. As explained in Section 5.4, the feedback loop consists of a diagonal PID controller and the feed forward signals are added to joints 2 and 3. The code for the feed forward block is listed in Appendix E. Furthermore, a limit for the applied torque is implemented on the output of the joint control block. This limit is set to ±5 Nm to correspond to the real robot.

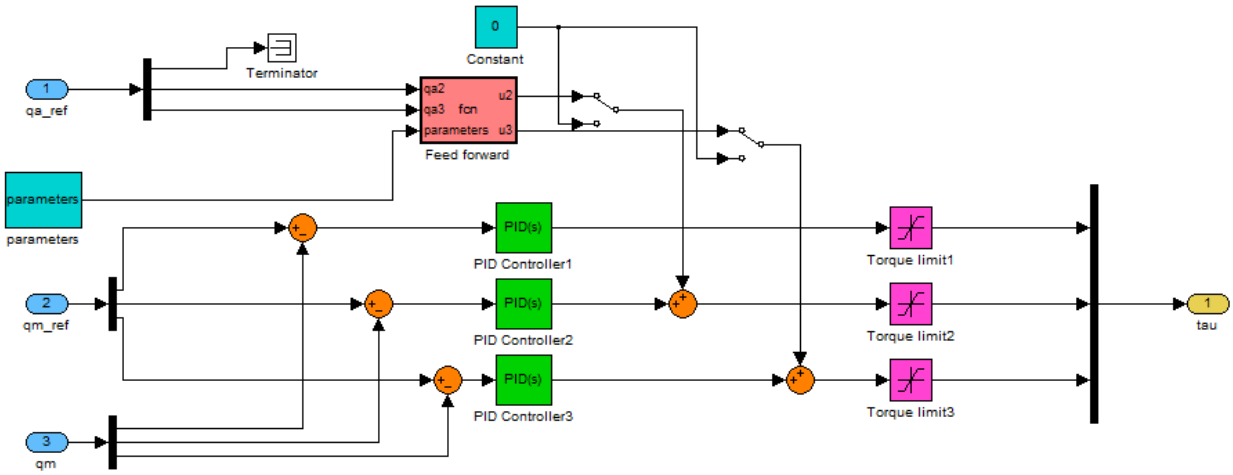


Figure 24: The joint controller implemented in Simulink.

The implementation of the robot model, i.e. the flexible dynamic model, is shown in Figure 25. Note that the signal *qa* is converted to degrees, because the forward kinematics wants the angles in degrees as input. The block *Robot model* simply contains the function in Appendix C.

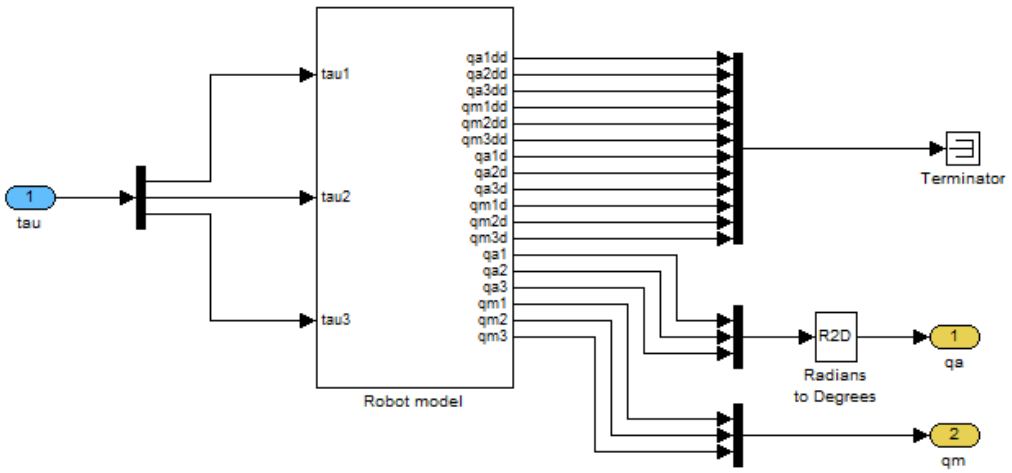


Figure 25: The flexible dynamic robot model implemented in Simulink.

The *Forward kinematics* block in Figure 22 contains the function for the forward kinematics listed in Appendix A. For information on how to use the simulation model, see the User manual [15].

6 Results

In this section, the results of the different parts of the project are presented.

6.1 System identification

In the friction experiments, the measured torque and the two fitted friction models can be seen in Figures 26 - 28. For axis 1 there is no restriction for how much the arm can rotate. For each different motor angular velocity the data were collected over several laps around the axis to get a good mean value of the torque. This is however not possible for axis 2 and 3 since the robot is restricted within a certain set of angles. This together with the slow sampling rate means that only a few data points are gathered with a large motor angular velocity, meaning that the mean torque is not as accurate as for joint 1. This might explain the diversity in the torque at higher angle velocities for axis 3.

The friction models have the parameters

$$F_{A,red} = \begin{bmatrix} 0.29196 sign(\dot{q}_{m1}) + 0.015743 \dot{q}_{m1} \\ 0.1905 sign(\dot{q}_{m2}) + 0.19194 \dot{q}_{m2} \\ 0.18099 sign(\dot{q}_{m3}) + 0.014088 \dot{q}_{m3} \end{bmatrix} \tag{6.1}$$

and

$$F_{B,blue} = \begin{bmatrix} 0.08356 \dot{q}_{m1} \\ 0.33343 \dot{q}_{m2} \\ 0.053467 \dot{q}_{m3} \end{bmatrix}. \tag{6.2}$$

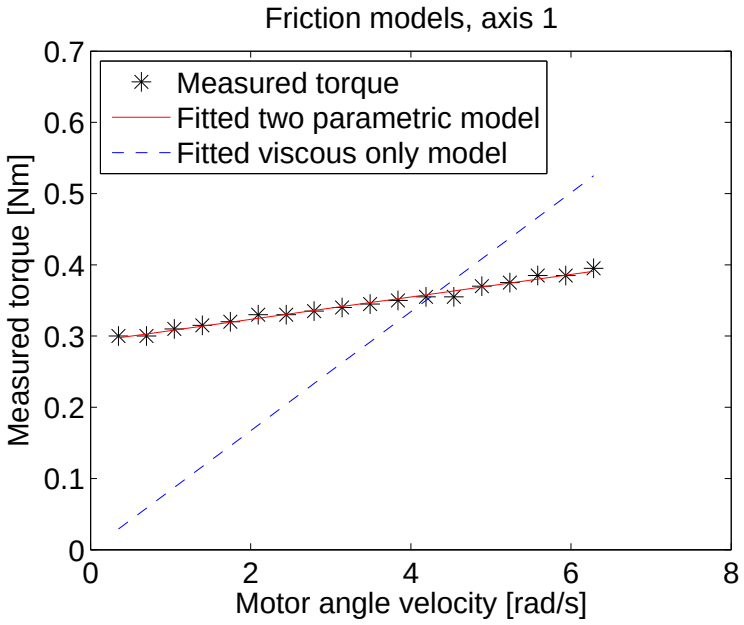


Figure 26: Measured torque and two fitted models for the friction, axis 1.

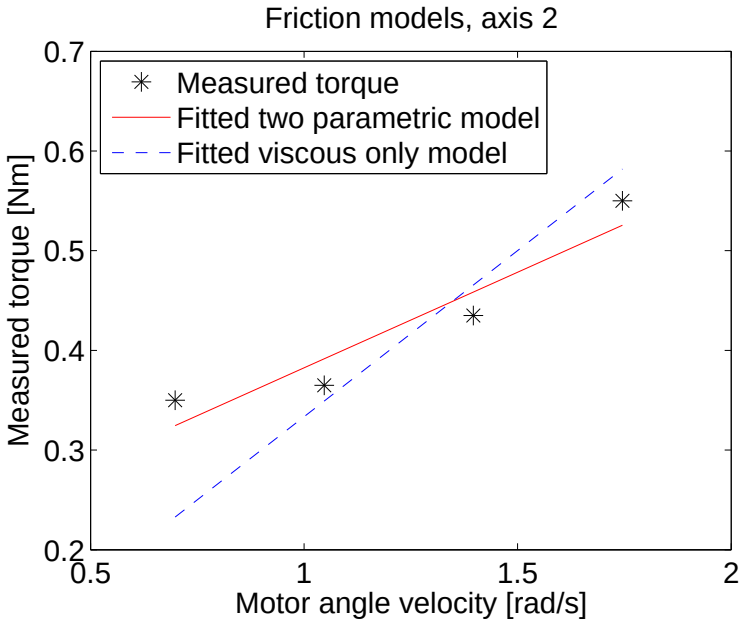


Figure 27: Measured torque and two fitted models for the friction, axis 2. The motor in axis 2 is slower than the other two which is why the angle velocity only goes up to around 1.8 rad/s. This explains why only 4 data points were gathered.

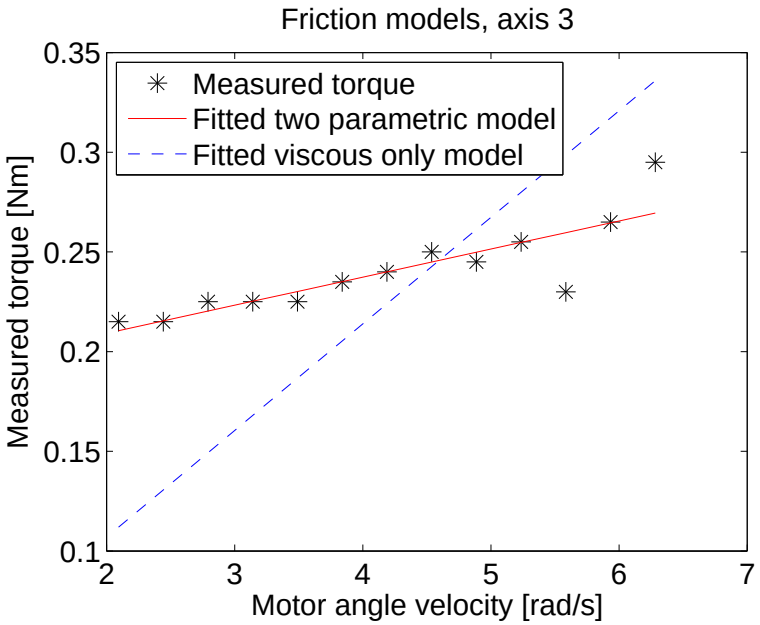


Figure 28: Measured torque and two fitted models for the friction, axis 3. The angular velocity is needed to be higher here to obtain a smooth movement.

6.2 Simulation

A step response in closed loop with the PID parameters

$$P_1 = P_2 = 1 \tag{6.3}$$
$$I_1 = I_2 = 0 \tag{6.4}$$
$$D_1 = D_2 = 0.1 \tag{6.5}$$

and

$$P_3 = 1 \tag{6.6}$$
$$I_3 = 0 \tag{6.7}$$
$$D_3 = 0, \tag{6.8}$$

and without feed forward control, is shown in Figure 29. The steps in reference angles are from 0° to 50°, 20° and 40° for joint 1, 2 and 3 respectively. In this case, the stationary error becomes 0 % for joint 1, 130 % for joint 2 and 2.2 % for joint 3. The overshoot becomes 0 % for all three joints.

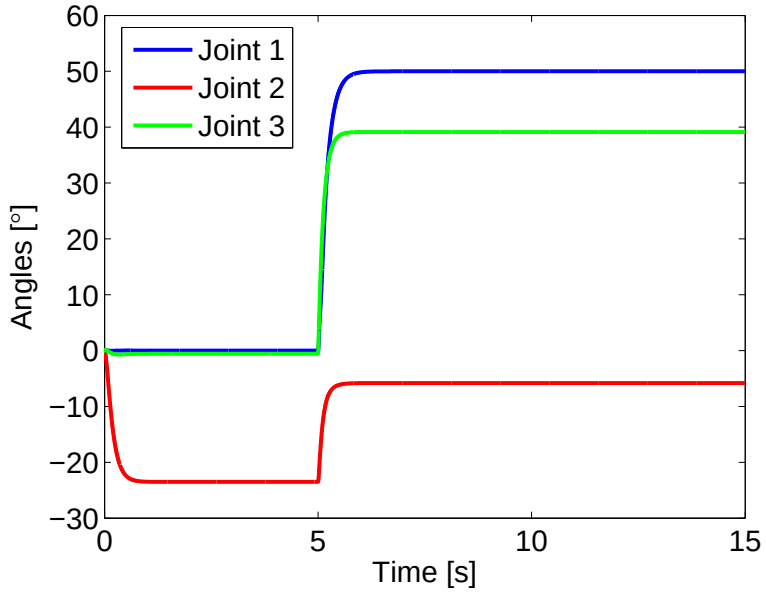


Figure 29: Simulated joint angles as function of time, without feed forward.

With the same PID parameters and steps in reference joint angles, but with a feed forward loop added as described in Section 5.4, the step responses look like in Figure 30. In this case, the stationary error becomes 0 % for all three joints and the overshoot becomes 0 % for all three joints as well.

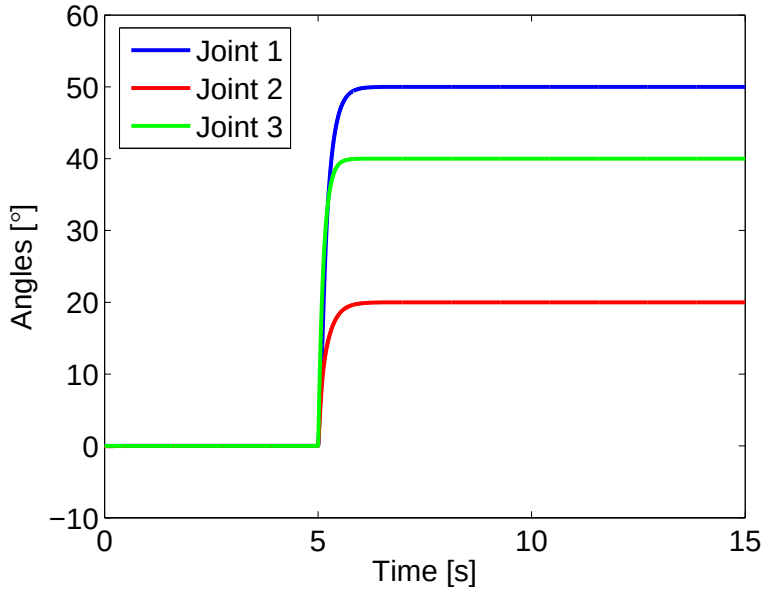


Figure 30: Simulated joint angles as function of time, with feed forward.

With the same PID paramaters as before, but with an integral part of 0.5 for joint 2, a simulation with a model error in the feed forward block is done. The mass of the second arm of the robot is changed to 25 % higher than the real value. This gives the step responses in Figure 31.

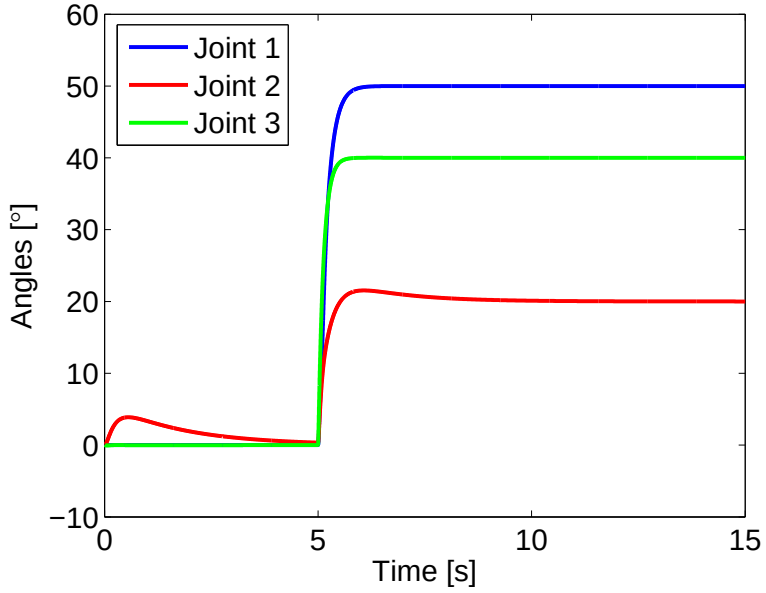


Figure 31: Simulated joint angles as function of time, with feed forward and a model error.

In Figure 32, a simulated trajectory between

$$A = \begin{bmatrix} 15 \\ -10 \\ 25 \end{bmatrix} \text{[cm]} \tag{6.9}$$

and

$$B = \begin{bmatrix} 25 \\ 15 \\ 40 \end{bmatrix} \text{[cm]}, \tag{6.10}$$

in room coordinates is shown. The x-, y- and z-coordinates as function of time is shown in Figure 33. The error of the tool position coordinates as function of time, calculated as

$$e = \sqrt{(x_{ref} - x)^2 + (y_{ref} - y)^2 + (z_{ref} - z)^2}, \tag{6.11}$$

is shown in Figure 34. The trajectory speed reference is 10 cm/s, and the calculated speed along the path is shown in Figure 35.

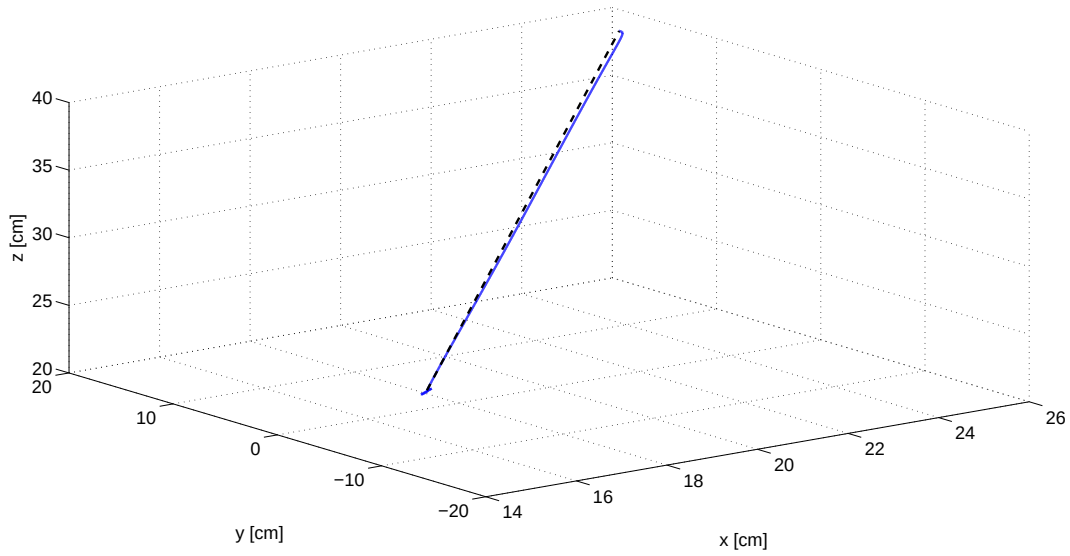


Figure 32: Simulated trajectory between A and B.

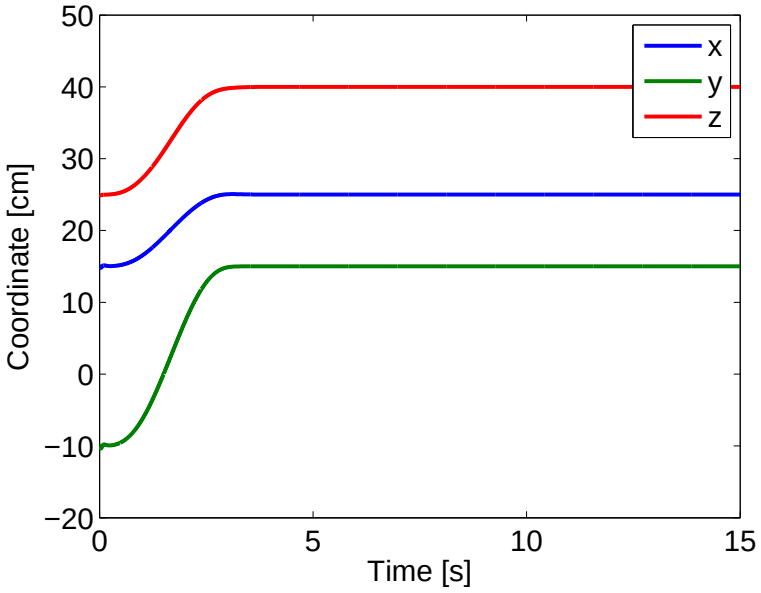


Figure 33: Simulated coordinates as function of time along the trajectory.

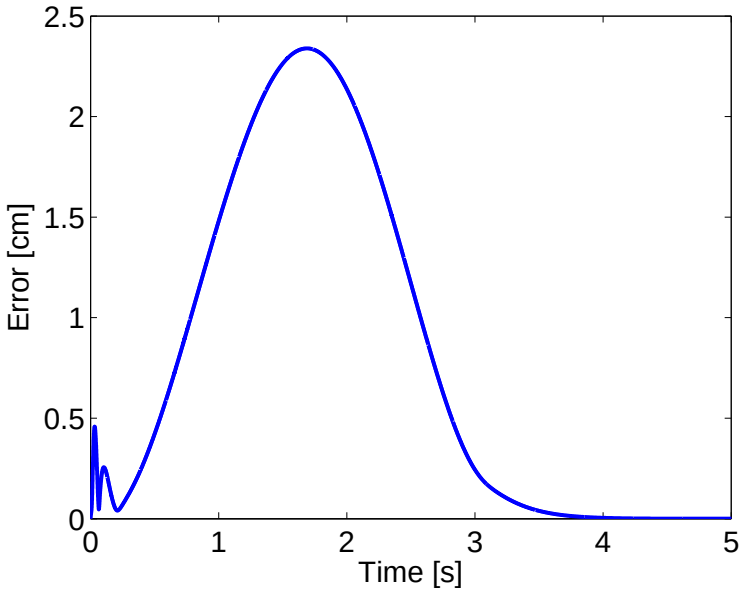


Figure 34: Error as function of time for the tool position coordinates.

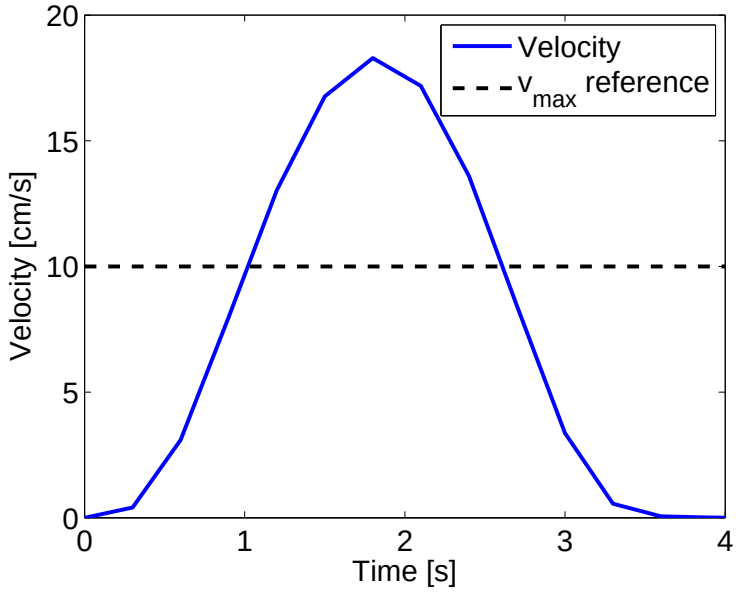


Figure 35: Simulated velocity as function of time along the trajectory.

6.3 Real robot

In this section, results from measurements on the real robot are presented.

6.3.1 Joint control

The data in this section is the results from a step in desired joint angles, from 0° to 50°, 20° and 40° for joint 1, 2 and 3 respectively. The desired joint angular velocity is set to maximum, 60°/s. The controller parameters for axis 1, 2 and 3 are shown in Figure 36, 37 and 38, respectively.

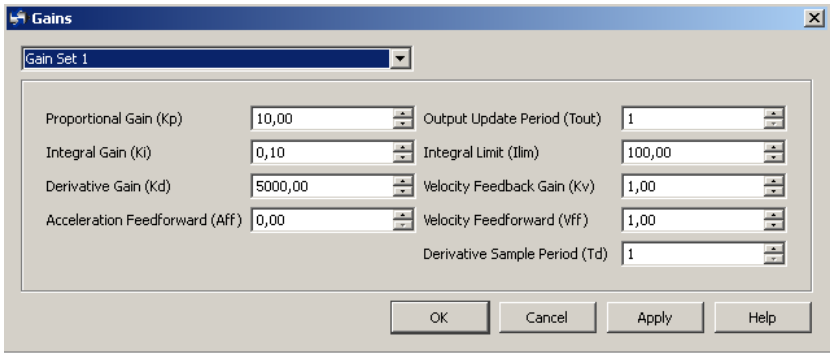


Figure 36: Controller parameters for axis 1.

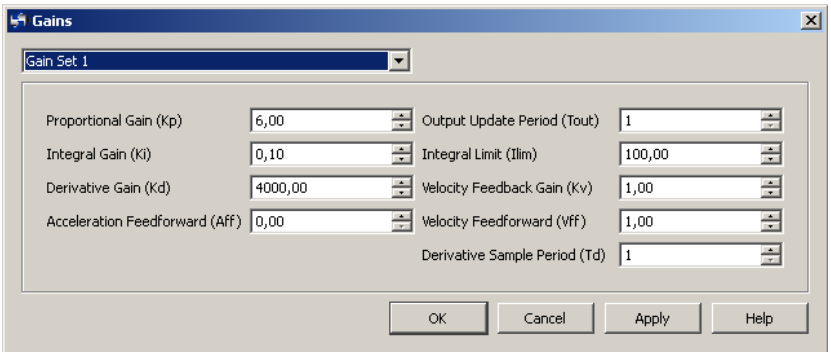


Figure 37: Controller parameters for axis 2.

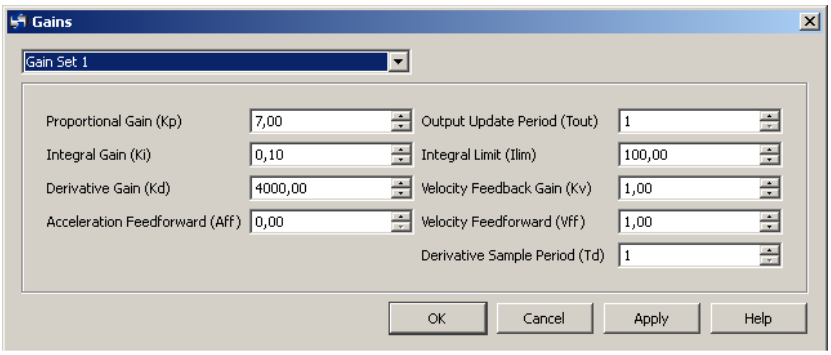


Figure 38: Controller parameters for axis 3.

The step responses for the three joint angles are shown in Figure 39, and the corresponding angular velocities are shown in Figure 40. The maximum angular velocity reference is also plotted in Figure 40 as the dashed straight line. The stationary errors in Figure 39 are 4.0 % for joint 1, 3.0 % for joint 2 and 1.0 % for joint 3. The overshoots are 0 % for joint 1 and 3, and 2.5 % for joint 2.

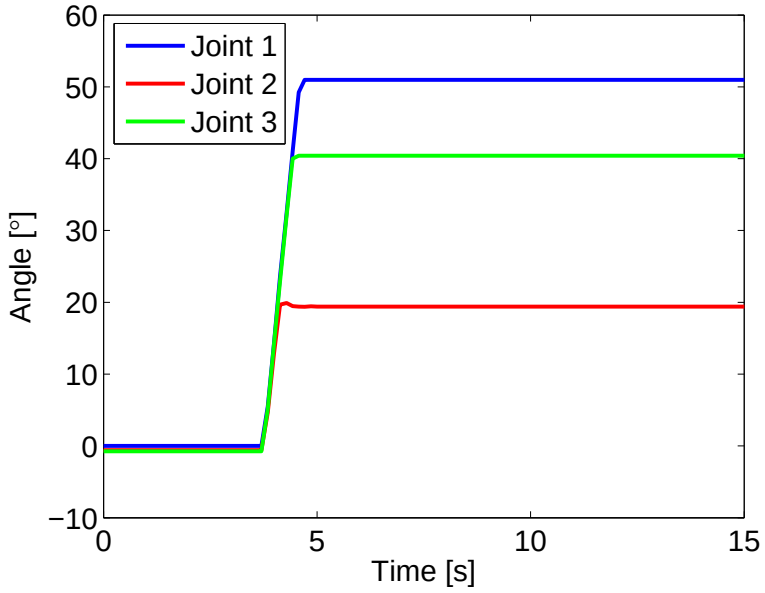


Figure 39: Joint angles as function of time on the real robot.

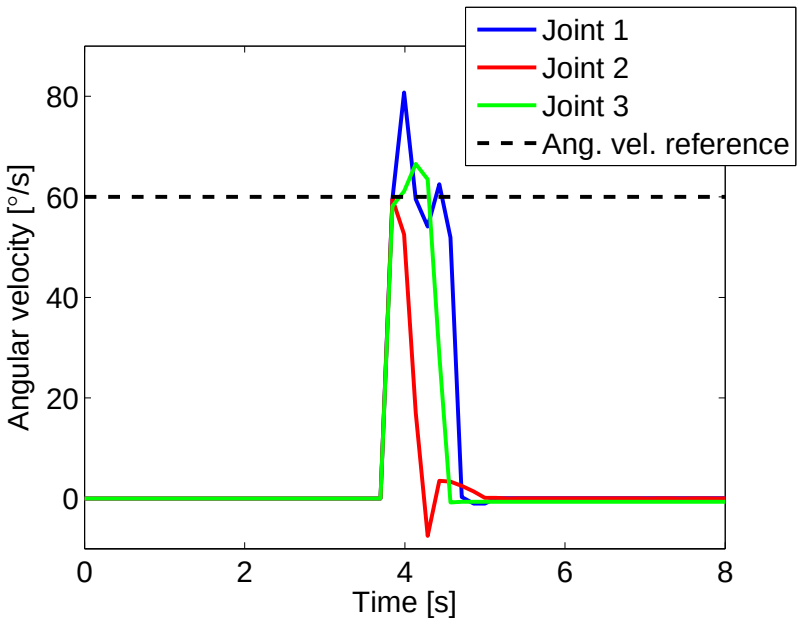


Figure 40: Joint angular velocities as function of time on the real robot.

6.3.2 Trajectory following

A trajectory from A to B is shown in Figure 41, where

$$A = \begin{bmatrix} 15 \\ -10 \\ 25 \end{bmatrix} \text{ [cm]} \tag{6.12}$$

and

$$B = \begin{bmatrix} 25 \\ 15 \\ 40 \end{bmatrix} \text{ [cm]}. \tag{6.13}$$

In Figure 41, a straight line (dashed) between A and B is also plotted. The calculated x-, y- and z-coordinates as function of time is shown in Figure 42, and in Figure 43 the error of the tool position coordinates is shown. Furthermore, the velocity of the trajectory is set to maximum, 10 cm/s. In Figure 44, the velocity as function of time along the trajectory is shown, and the maximum speed reference is plotted as a dashed line.

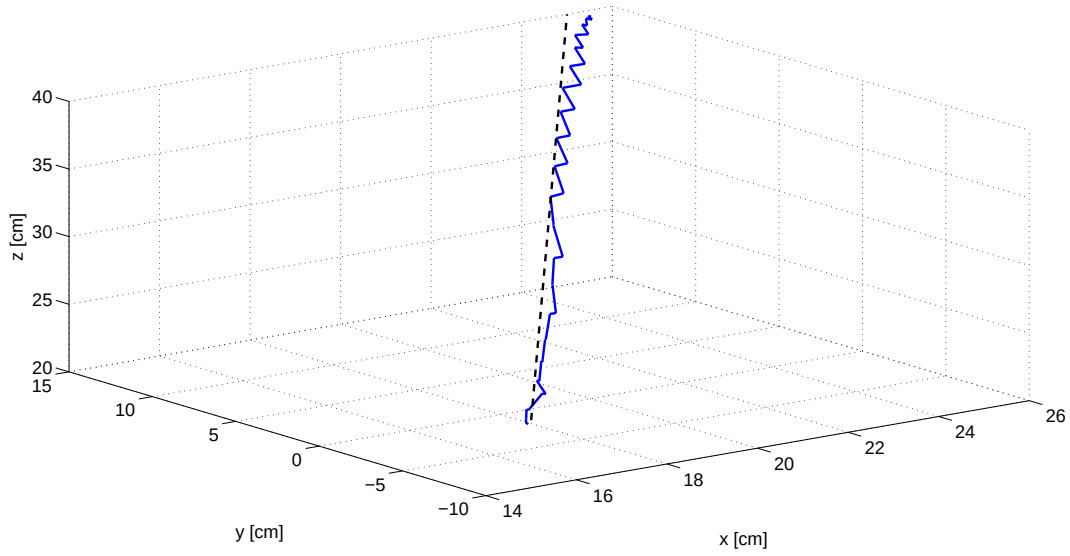


Figure 41: A trajectory from A to B.

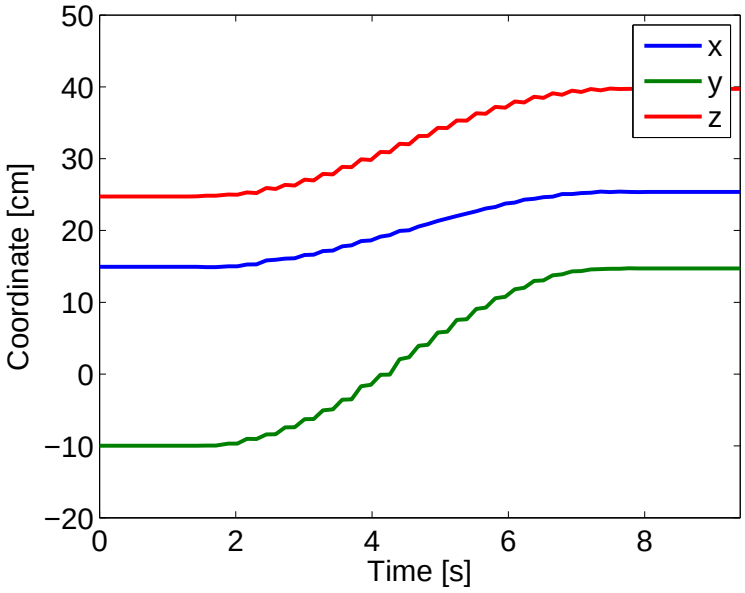


Figure 42: Calculated coordinates as function of time along the trajectory..

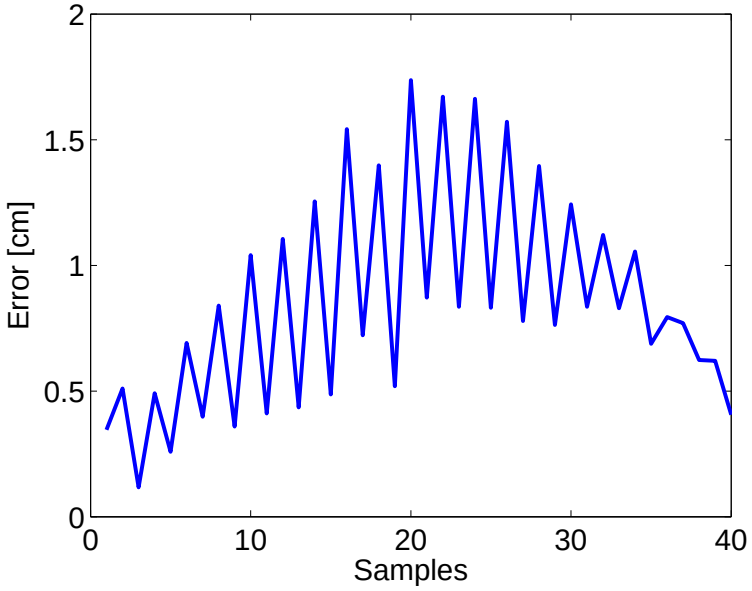


Figure 43: Error as function of samples for the tool position coordinates.

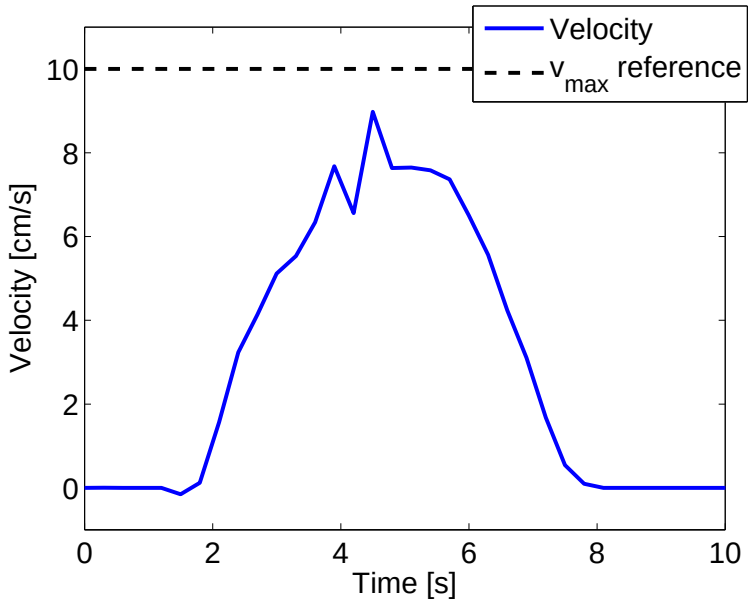


Figure 44: Velocity as function of time along the trajectory.

In Figure 45, the x-, y- and z-coordinates for the tool position are plotted separately. Both measured and simulated data are plotted, and the points A and B are the same as before.

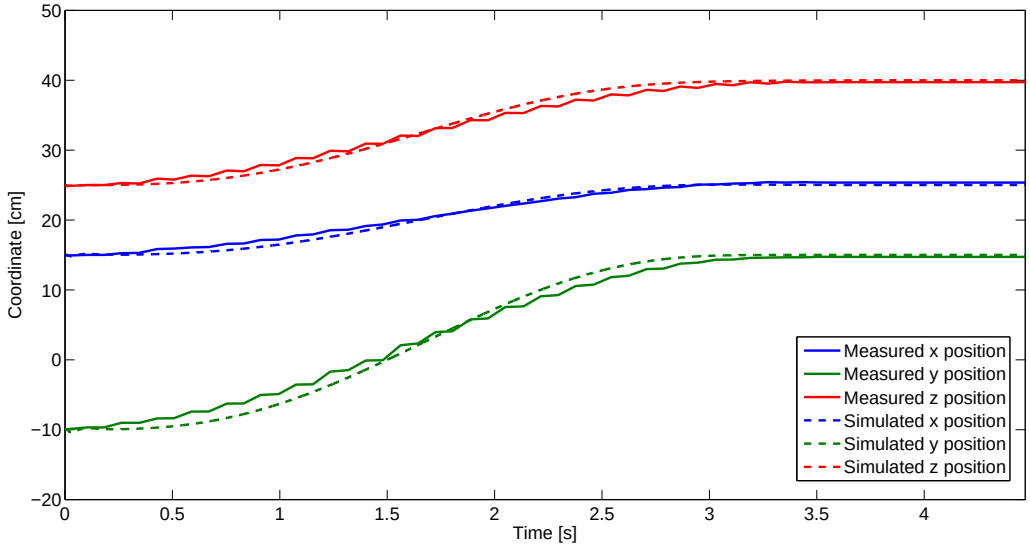


Figure 45: Coordinates as function of time along the trajectory.

7 Discussion and conclusions

In this section the results from Section 6 will be discussed. Also, deficiencies found on the current setup with the robot and hardware will be discussed, as well as suggestions of improvements for future development.

7.1 System identification

Several experiments were carried out with limited success. After discussion with both the customer, the orderer and the advisor it was agreed that the stiffness and damping coefficients were going to be too difficult to get accurate and this will be explained why below. The focus of the system identification became then to identify the friction.

The System Identification Toolbox was not able to give a good fit for the models of the flexibilities, and the loss function was larger than 1000. It should be pointed out that the exact same method was carried out using input-output data from an ABB IRB 1400 robot, which was taken from the control course TSRT62 – Modellbygge och Simulering given at ISY, Linköping University. The toolbox was able to fit a model with a loss function smaller than 1. This indicates that there is nothing wrong with the method used but more likely that there is something wrong with the hardware configuration. Why the method failed for the current configuration is most likely due to two reasons.

1. *Sampling*
   The data received from Labview is sampled at a rate of around 10 Hz and this is too low. The point of doing these experiments is to excite the eigenfrequencies of the system and see how they behave. For the ABB robot in the article [13], the first notch frequency is around 14.5 Hz which is much higher than what is possible to notice. According to the Nyquist frequency it is possible to notice frequencies in the signal up to half the sample rate, which in this case is around 5 Hz. The sample rate is even further not even constant and changes around 8-14 Hz. This makes it even more difficult for the identification.

2. *Backlash*
   The gearboxes have very large backlashes and in some positions of the robot, the backlash is around 5 degrees. The backlash is not taken into consideration in the model and this makes it difficult for the identification.

Eventually, the friction parameters were decided. From Figure 26, 27 and 28 it is clear that a two parametric model has the best fit.

7.2 Simulation versus reality

First some comments about the simulation results. For joint control it is easy to obtain good step responses with a simple PD controller and a feed forward loop, see Figure 30. Without a feed forward loop, however, the second and third joint will be highly affected by the gravitational pull, as seen in Figure 29. Especially joint 2 will be affected by the gravitation, since the second arm has a large mass. As seen in Figure 29, the angle for joint 2 leaves its initial value of 0° and causes a large stationary error. One may add an integral part to get rid of the stationary error, but it takes some time for the integral part to sum up to the right value. A feed forward loop needs a model of the robot dynamics, but this model will contain model errors. In Figure 31, the model is simulated with a

model error in the feed forward loop. As seen in the figure, the feed forward will give a larger torque than needed, since the model "thinks" the second arm is heavier than it really is. Therefore an integral part is added in this case, to handle the stationary error. Even with this rather big model error, the joint control becomes much better with a feed forward loop than the case with no feed forward at all, compare Figure 29 with Figure 31.

It is not easy to compare the simulation results with data from the robot, since the robot has a control loop already implemented in hardware, which differs from the one in the simulation model. Furthermore, the robot can not be run in open loop so it is not possible to compare with the flexible dynamic model. Also notice the PID parameters in Figure 36, 37 and 38. Since no feed forward loop can be implemented on the robot, the proportional gain has to be set higher than in the simulation model. Furthermore, the derivative gains are much larger than in the simulation model. These gains have to be this large to avoid oscillations from the motors in the settling phase of the step. On the real robot, an angular velocity trajectory is also implemented in hardware, which gives the user the possibility to choose the angular velocity over a joint movement. In Figure 39, the angular velocity is set to 60°/s, which makes the risetime larger compared to the simulated results. See also Figure 40 for a plot of the angular velocities. The joint control works well but one thing that could have made it more precise is to reduce the backlash in the gears. As seen in figure 39 there is a stationary error on the target angle. The derivative gain is set to a very high value for all of the axis as seen in Figure 36, 37 and 38. This is to reduce the oscillations on the robot.

The trajectory in Figure 41 is similar to the simulated one in Figure 32, with exception for its choppy appearance. This depends of the data logging in Labview, when data is logged the interface and also the robot is stuttering forward. This makes the velocity profile in Figure 44 wrong compared to the simulated profile in Figure 35. However, when data is not logged, the velocity profile seems correct and the robot does not stutter. Because of the problems with the data logger, the time axis of the calculated coordinates in Figure 45 have been scaled to make a comparison with the simulated results possible. In this figure, it seems like reality and simulation agree. The trajectory could have been smoother if Labview could check for reference signals more often. The fact that the sampling time is not constant makes it difficult to generate a vector with reference angles of the correct length. As of right now there are a lot more elements in the reference vector compared to the amount of times Labview checks for a reference. If the sampling time were constant these two could have been the same. This could have reduced the number of calculations made and maybe Labview would have run smoother.

7.3 Possible improvements

There are several problems with the hardware which makes it difficult to control the robot. There is a considerably large backlash, about 5 degrees, both in the gears and to some extent in the motor itself. This makes it hard to get accurate angle positions. For axis 1, the joint became very oscillatory at every stop. Since the force of gravity is parallel to axis 1 it does not not affect the outcome of the angle. Therefore the same step in joint angle can give different results. Axes 2 and 3 are affected by gravity and will be pushed down. This will for example make it almost impossible to reach exactly 90 degrees for axis 2. This problem can be fixed by reducing the backlash.

The motor that was originally at axis 2 kept blowing a fuse which turned out to be because it could not generate enough torque to lift the arm. That problem was solved by changing the motor for axis 2 and 3 so that the faulty motor is now at the end of the arm, at axis 3. The motor that is now at axis 2 is not working very well and is very oscillatory in the

end position for some angles close to 90 degrees.

Another problem is the National Instruments hardware. The NI 9054 requires the software NI SoftMotion [7] which does most of the control on its own. It has a built in PID loop for control of the motors and it is not possible to disable it. This makes the system identification very difficult since it is not possible to run the open loop system. The initial plan was to have a feed forward, but that is not possible with the SoftMotion software. It would be recommended to exchange them to cards that support Matlab, such as xPC Target [16] instead of the National Instruments equipment. All control could then be done in Matlab, instead of Labview. The connection to the robot is very unstable and it is not uncommon to lose the connection for no apparent reason. This problem should be fixed if the hardware is replaced. Labview is not an optimal program to use for control of the robot. The trajectory planning is done in Matlab and there were some issues getting it in to Labview. There is a function that you can use to add a Mathscript in to the Labview interface, but the trajectory planner is too long so Labview lagged out and lost the connection to the robot if it was ran more than once. Because of this, the program has to be restarted every time the trajectory is to be switched. Several of the functions that were originally used in the trajectory planner had to be changed because they were not supported in the Mathscript. Because of this, Labview and the current hardware setup is not recommended to control an industrial robot.

References

[1] National Instruments, *NI Labview 2010* http://www.ni.com/labview/, 2010.

[2] The Mathworks Inc., *Matlab* http://www.mathworks.se/products/matlab/index.html, 2010.

[3] The Mathworks Inc., *Simulink* http://www.mathworks.se/products/simulink/index.html, 2010.

[4] Faulhaber, *Faulhaber 3242G024BX4 Brushless DC Motor*, http://www.faulhaber.com/uploadpk/EN_3242_BX4_Encoder_DFF.pdf.

[5] Faulhaber, *Faulhaber Type BLD 5603 Servo Amplifiers*, http://www.faulhaber.com/uploadpk/EN_BLD5603-06_im_MIN.pdf.

[6] National Instruments, *NI 9514 C Series Servo Drive Interface with Encoder Feedback*, http://sine.ni.com/nips/cds/view/p/lang/en/nid/206348.

[7] National Instruments, *LabVIEW NI SoftMotion Module*, http://sine.ni.com/nips/cds/view/p/lang/en/nid/14234.

[8] National Instruments, *NI cRIO-9022 Real-Time Controller with 256 MB DRAM, 2 GB Storage*, http://sine.ni.com/nips/cds/view/p/lang/en/nid/206760.

[9] Craig, John J., *Introduction to robotics: Mechanics and control*. Pearson Education, 3rd Edition, 2005.

[10] The Mathworks Inc., *Symbolic Math Toolbox*, http://www.mathworks.se/products/symbolic/index.html, 2010.

[11] Bruno Siciliano, Oussama Khatib, *Springer handbook of robotics*, 2008.

[12] M. W. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, 1st Edition, 2005.

[13] M. Östring, S. Gunnarsson, M. Norrlöf, *Closed-loop identification of an industrial robot containg flexibilities*, 2003.

[14] The Mathworks Inc., *System Identification Toolbox*, http://www.mathworks.se/products/sysid/index.html, 2010.

[15] Gällsjö, Anders and Ingeström, Victor, *User manual - Modeling and control of an industrial robot*, Department of Electrical Engineering, Linköping University, 2011.

[16] The Mathworks Inc., *xPC Target*, http://www.mathworks.se/products/xpctarget/supported-hardware/index.html.

A Forward kinematics

```
function position = forward(th1,th2,th3,l1,l2,l3)
%FORWARD Calculates the tool position.
%   pos = forward(th1,th2,th3,l1,l2,l3) is the position of the tool
%   in room coordinates. Input arguments are the joint angles and
%   the arm lengths.
T_a_b = [cosd(th1) -sind(th1) 0 0;sind(th1) cosd(th1) 0 0;0 0 1 0;0 0 0 1];
T_b_c = [cosd(th2) -sind(th2) 0 0;0 0 -1 0;sind(th2) cosd(th2) 0 l1;0 0 0 1];
T_c_d = [cosd(th3) -sind(th3) 0 l2;sind(th3) cosd(th3) 0 0;0 0 1 0;0 0 0 1];
T_d_e = [1 0 0 l3;0 1 0 0;0 0 1 0;0 0 0 1];

T_a_e = T_a_b*T_b_c*T_c_d*T_d_e;

Pe = [0;0;0;1];

Pa = T_a_e*Pe;

position = Pa(1:3);

end
```

B Inverse kinematics

```
function [angles] = inverse(x,y,z,l1,l2,l3)
%INVERSE Calculates the joint angles.
%   angles = inverse(x,y,z,l1,l2,l3) are the angles of the joints.
%   Input arguments are the room coordinates for the tool and the
%   arm lengths.

%Help variables
S = sqrt(x^2+y^2);
L = sqrt(S^2+(z-l1)^2);
%phi = acos(X/L);

% theta_1 in right quadrant [0 < theta_1 < 360]
if y < 0
    th1 = atan2(y,x);
else
    th1 = atan2(y,x);
end
th_1_deg = th1*180/pi;


% Calculation of angles
alpha_1 = atan2(z-l1,S);
alpha_2 = acos((l2^2-l3^2+L^2)/(2*l2*L));
q1 = alpha_1 + alpha_2;
q2 = acos((l2^2+l3^2-L^2)/(2*l2*l3));

% Alternative 1 (th_1 turned towards point)
th_2_deg = q1*180/pi;
```

```
th_3_deg = q2*180/pi-180;

% Alternative 2 (th_1 turned towards point)
th_1_deg_2 = th_1_deg;
th_2_deg_2 = (alpha_1-alpha_2)*180/pi;
th_3_deg_2 = -th_3_deg;

% Alternative 3 (th_1 turned away from point)
if th_1_deg <= 0
    th_1_deg_3=th_1_deg+180;
else
    th_1_deg_3=th_1_deg-180;
end
th_2_deg_3 = -180-th_2_deg_2;
th_3_deg_3 = th_3_deg;

% Alternative 4 (th_1 turned away from point)
th_1_deg_4 = th_1_deg_3;
th_2_deg_4 = 180-th_2_deg;
th_3_deg_4 = -th_3_deg;


%%%%%%%%%%%%%%%%%%%% Adjustment so that -90 < theta 2 < 270
if th_2_deg > 270 || th_2_deg < -90
    th_2_deg = th_2_deg - sign(th_2_deg)*360;
end

if th_2_deg_2 > 270 || th_2_deg_2 < -90
    th_2_deg_2 = th_2_deg_2 - sign(th_2_deg_2)*360;
end

if th_2_deg_3 > 270 || th_2_deg_3 < -90
    th_2_deg_3 = th_2_deg_3 - sign(th_2_deg_3)*360;
end

if th_2_deg_4 > 270 || th_2_deg_4 < -90
    th_2_deg_4 = th_2_deg_4 - sign(th_2_deg_4)*360;
end

%%%%%%%%%%

angles = roundn(real([th_1_deg th_1_deg_2 th_1_deg_3 th_1_deg_4;...
                th_2_deg th_2_deg_2 th_2_deg_3 th_2_deg_4;...
                th_3_deg th_3_deg_2 th_3_deg_3 th_3_deg_4;...
                1 1 1 1]),-5);

% Check if robot tool is out of its range
check=forward(real(th_1_deg),real(th_2_deg),real(th_3_deg),l1,l2,l3);
check=roundn(check,-5);
x=roundn(x,-5);
y=roundn(y,-5);
z=roundn(z,-5);
```

```
if check(1)~=x || check(2)~=y || check(3)~=z
    angles(4,:) = [0 0 0 0]; %=false;
else
    index = [find(angles(1,:)<-179) find(angles(1,:)>179) ...
             find(angles(2,:)<-30) find(angles(2,:)>210) ...
             find(angles(3,:)<-135) find(angles(3,:)>135)];
    angles(4,index) = 0;
end
end
```

C Flexible dynamic model

```
%%%%%%%%%%%%%%%%%%%%%%
% Robot parameters %
%        &         %
% Init Simulation  %
%%%%%%%%%%%%%%%%%%%%%%

% Filter constant

alpha = 0.01;

% Gear ratio
n1 = 1/2;
n2 = 1/2;
n3 = 1/2;

% Initial conditions for the arms. Angles for motors will be calculated
% assuming no flexible torque is transferred through the gearboxes
% [rad]
qa1_0 = 0;
qa2_0 = 0;
qa3_0 = 0;
qm1_0 = 0;
qm2_0 = 0;
qm3_0 = 0;

% [rad/s]
qa1d_0 = 0;
qa2d_0 = 0;
qa3d_0 = 0;
qm1d_0 = qa1d_0/n1;
qm2d_0 = qa2d_0/n2;
qm3d_0 = qa3d_0/n3;

%%%%%%%%%%%%%%%%%%%%%%
% Robot link properties
%%%%%%%%%%%%%%%%%%%%%%
% Mass [kg]
m1 = 0.790;
m2 = 1.06;
m3 = 0.197;
```

```
% Center of gravity, expressed in the coordinatesystem attached to each
% link [m]
Gx1 = -0.0123;
Gy1 = -0.0480;
Gz1 = 0.0628;

Gx2 = 0.121;
Gy2 = -0.000800;
Gz2 = -0.0235;

Gx3 = 0.0286;
Gy3 = -0.0133;
Gz3 = 0;

% Moments and producs of inertia, all with respect to the center
% of gravity [kgm^2]
Ixx1 = 0.00253;
Ixx2 = 9.32e-4;
Ixx3 = 9.78e-5;

Iyy1 = 0.00281;
Iyy2 = 0.00600;
Iyy3 = 7.12e-4;

Izz1 = 0.000439;
Izz2 = 0.00500;
Izz3 = 6.62e-4;

Ixy1 = 4.54e-07;
Ixy2 = -2.36e-5;
Ixy3 = 1.22e-7;

Ixz1 = -0.000399;
Ixz2 = 0.00100;
Ixz3 = -7.50e-5;

Iyz1 = 5.96e-07;
Iyz2 = -9.00e-6;
Iyz3 = -9.17e-8;

% Link lengths, distance from origin to origin [m]
L1 = 0.17;
L2 = 0.255;
L3 = 0.15;

% Motor inertias [kgm^2]
Imzz1 = 6.38e-5;
Imzz2 = 6.38e-5;
Imzz3 = 6.38e-5;

% Spring coefficient in the K vector [Nm/rad]
```

```
Kplastic = 100;
Kmetal = 1000;

% Damping coefficient in the D vector [Nms/rad]
Dplastic = 1;
Dmetal = 0.1;

% Fricion coefficient in the F vector [Nms/rad]
Ffric1 = 0.08;
Ffric2 = 0.33;
Ffric3 = 0.05;

parameters = [m1,m2,m3,n1,n2,n3,Gx1,Gy1,Gz1,Gx2,Gy2,Gz2,Gx3,Gy3,Gz3,...
    Ixx1,Ixx2,Ixx3,Iyy1,Iyy2,Iyy3,Izz1,Izz2,Izz3,Ixy1,Ixy2,Ixy3,Ixz1,...
    Ixz2,Ixz3,Iyz1,Iyz2,Iyz3,L1,L2,L3,Imzz1,Imzz2,Imzz3,Kplastic,Kmetal,...
    Dplastic,Dmetal,Ffric1,Ffric2,Ffric3];

function [qa1dd,qa2dd,qa3dd,qm1dd,qm2dd,qm3dd] = dynamic(qa1d,qa2d,qa3d,...
    qa1,qa2,qa3,parameters,tau1,tau2,tau3,qm1d,qm2d,qm3d,qm1,qm2,qm3)
%DYNAMIC The dynamic flexible model.

% Loading the robot parameters from workspace
m1 = parameters(1);
m2 = parameters(2);
m3 = parameters(3);
n1 = parameters(4);
n2 = parameters(5);
n3 = parameters(6);
Gx1 = parameters(7);
Gy1 = parameters(8);
Gz1 = parameters(9);
Gx2 = parameters(10);
Gy2 = parameters(11);
Gz2 = parameters(12);
Gx3 = parameters(13);
Gy3 = parameters(14);
Gz3 = parameters(15);
Ixx1 = parameters(16);
Ixx2 = parameters(17);
Ixx3 = parameters(18);
Iyy1 = parameters(19);
Iyy2 = parameters(20);
Iyy3 = parameters(21);
Izz1 = parameters(22);
Izz2 = parameters(23);
Izz3 = parameters(24);
Ixy1 = parameters(25);
Ixy2 = parameters(26);
Ixy3 = parameters(27);
Ixz1 = parameters(28);
Ixz2 = parameters(29);
Ixz3 = parameters(30);
Iyz1 = parameters(31);
```

```
Iyz2 = parameters(32);
Iyz3 = parameters(33);
L1 = parameters(34);
L2 = parameters(35);
L3 = parameters(36);
Imzz1 = parameters(37);
Imzz2 = parameters(38);
Imzz3 = parameters(39);
Kplastic = parameters(40);
Kmetal = parameters(41);
Dplastic = parameters(42);
Dmetal = parameters(43);
Ffric1 = parameters(44);
Ffric2 = parameters(45);
Ffric3 = parameters(46);

% The mass matrix, M
MSim = [Ixx2/2 + Ixx3/2 + Iyy2/2 + Iyy3/2 + Izz1 - ...
    (Ixx3*cos(2*qa2 + 2*qa3))/2 + (Iyy3*cos(2*qa2 + 2*qa3))/2 - ...
    Ixy3*sin(2*qa2 + 2*qa3) + Gx1^2*m1 + Gy1^2*m1 + (Gx2^2*m2)/2 + ...
    (Gy2^2*m2)/2 + (Gx3^2*m3)/2 + Gz2^2*m2 + (Gy3^2*m3)/2 + Gz3^2*m3 + ...
    (L2^2*m3)/2 - (Ixx2*cos(2*qa2))/2 + (Iyy2*cos(2*qa2))/2 - ...
    Ixy2*sin(2*qa2) + (Gx2^2*m2*cos(2*qa2))/2 - (Gy2^2*m2*cos(2*qa2))/2 ...
    + (L2^2*m3*cos(2*qa2))/2 + (Gx3^2*m3*cos(2*qa2 + 2*qa3))/2 - ...
    (Gy3^2*m3*cos(2*qa2 + 2*qa3))/2 + Gx3*L2*m3*cos(qa3) - ...
    Gy3*L2*m3*sin(qa3) + Gx3*L2*m3*cos(2*qa2 + qa3) - ...
    Gy3*L2*m3*sin(2*qa2 + qa3) - Gx2*Gy2*m2*sin(2*qa2) - ...
    Gx3*Gy3*m3*sin(2*qa2 + 2*qa3), - Iyz3*cos(qa2 + qa3) - ...
    Ixz3*sin(qa2 + qa3) - Iyz2*cos(qa2) - Ixz2*sin(qa2) - ...
    Gy3*Gz3*m3*cos(qa2 + qa3) - Gx3*Gz3*m3*sin(qa2 + qa3) - ...
    Gy2*Gz2*m2*cos(qa2) - Gx2*Gz2*m2*sin(qa2) - Gz3*L2*m3*sin(qa2), ...
    - Iyz3*cos(qa2 + qa3) - Ixz3*sin(qa2 + qa3) - ...
    Gy3*Gz3*m3*cos(qa2 + qa3) - Gx3*Gz3*m3*sin(qa2 + qa3); ...
    - Iyz3*cos(qa2 + qa3) - Ixz3*sin(qa2 + qa3) - Iyz2*cos(qa2) - ...
    Ixz2*sin(qa2) - Gy3*Gz3*m3*cos(qa2 + qa3) - ...
    Gx3*Gz3*m3*sin(qa2 + qa3)- Gy2*Gz2*m2*cos(qa2) - ...
    Gx2*Gz2*m2*sin(qa2)- Gz3*L2*m3*sin(qa2), m2*Gx2^2 + m3*Gx3^2 + ...
    2*m3*cos(qa3)*Gx3*L2 + m2*Gy2^2 + m3*Gy3^2 - 2*m3*sin(qa3)*Gy3*L2 + ...
    m3*L2^2 + Izz2 + Izz3, m3*Gx3^2 + L2*m3*cos(qa3)*Gx3 + m3*Gy3^2 ...
    - L2*m3*sin(qa3)*Gy3 + Izz3; - Iyz3*cos(qa2 + qa3) - ...
    Ixz3*sin(qa2 + qa3) - Gy3*Gz3*m3*cos(qa2 + qa3) - ...
    Gx3*Gz3*m3*sin(qa2 + qa3), m3*Gx3^2 + L2*m3*cos(qa3)*Gx3 + m3*Gy3^2 ...
    - L2*m3*sin(qa3)*Gy3 + Izz3, m3*Gx3^2 + m3*Gy3^2 + Izz3];

% The centrifugal and coriolis vector, C
CSim = [- qa3d*(qa1d*(Ixy3*cos(2*qa2 + 2*qa3) - ...
    (Ixx3*sin(2*qa2 + 2*qa3))/2 + (Iyy3*sin(2*qa2 + 2*qa3))/2 + ...
    (Ixz3*cos(qa2 + qa3))/2 - (Iyz3*sin(qa2 + qa3))/2 + ...
    (Gx3^2*m3*sin(2*qa2 + 2*qa3))/2 - (Gy3^2*m3*sin(2*qa2 + 2*qa3))/2 + ...
    (Gx3*Gz3*m3*cos(qa2 + qa3))/2 - (Gy3*Gz3*m3*sin(qa2 + qa3))/2 + ...
    (Gy3*L2*m3*cos(qa3))/2 + (Gx3*L2*m3*sin(qa3))/2 + ...
    (Gy3*L2*m3*cos(2*qa2 + qa3))/2 + (Gx3*L2*m3*sin(2*qa2 + qa3))/2 + ...
```

```
Gx3*Gy3*m3*cos(2*qa2 + 2*qa3)) + qa2d*(Ixz3*cos(qa2 + qa3) - ...
Iyz3*sin(qa2 + qa3) + Gx3*Gz3*m3*cos(qa2 + qa3) - ...
Gy3*Gz3*m3*sin(qa2 + qa3)) + qa3d*(Ixz3*cos(qa2 + qa3) - ...
Iyz3*sin(qa2 + qa3) + Gx3*Gz3*m3*cos(qa2 + qa3) - ...
Gy3*Gz3*m3*sin(qa2 + qa3))) - qa2d*(qa2d*(Ixz3*cos(qa2 + qa3) - ...
Iyz3*sin(qa2 + qa3) + Ixz2*cos(qa2) - Iyz2*sin(qa2) + ...
Gx3*Gz3*m3*cos(qa2 + qa3) - Gy3*Gz3*m3*sin(qa2 + qa3) + ...
Gx2*Gz2*m2*cos(qa2) + Gz3*L2*m3*cos(qa2) - Gy2*Gz2*m2*sin(qa2)) + ...
qa3d*(Ixz3*cos(qa2 + qa3) - Iyz3*sin(qa2 + qa3) + ...
(Ixz2*cos(qa2))/2 - (Iyz2*sin(qa2))/2 + Gx3*Gz3*m3*cos(qa2 + qa3) - ...
Gy3*Gz3*m3*sin(qa2 + qa3) + (Gx2*Gz2*m2*cos(qa2))/2 + ...
(Gz3*L2*m3*cos(qa2))/2 - (Gy2*Gz2*m2*sin(qa2))/2) + ...
qa1d*((m2*sin(2*qa2)*Gx2^2)/2 + m2*cos(2*qa2)*Gx2*Gy2 + ...
(Gz2*m2*cos(qa2)*Gx2)/2 + (m3*sin(2*qa2 + 2*qa3)*Gx3^2)/2 + ...
m3*cos(2*qa2 + 2*qa3)*Gx3*Gy3 + m3*sin(2*qa2 + qa3)*Gx3*L2 + ...
(Gz3*m3*cos(qa2 + qa3)*Gx3)/2 - (m2*sin(2*qa2)*Gy2^2)/2 - ...
(Gz2*m2*sin(qa2)*Gy2)/2 - (m3*sin(2*qa2 + 2*qa3)*Gy3^2)/2 + ...
m3*cos(2*qa2 + qa3)*Gy3*L2 - (Gz3*m3*sin(qa2 + qa3)*Gy3)/2 + ...
(m3*sin(2*qa2)*L2^2)/2 + (Gz3*m3*cos(qa2)*L2)/2 + ...
Ixy3*cos(2*qa2 + 2*qa3) - (Ixx3*sin(2*qa2 + 2*qa3))/2 + ...
(Iyy3*sin(2*qa2 + 2*qa3))/2 + (Ixz3*cos(qa2 + qa3))/2 - ...
(Iyz3*sin(qa2 + qa3))/2 + (Ixz2*cos(qa2))/2 - (Iyz2*sin(qa2))/2 + ...
Ixy2*cos(2*qa2) - (Ixx2*sin(2*qa2))/2 + (Iyy2*sin(2*qa2))/2)); ...
qa1d*(qa1d*((m2*sin(2*qa2)*Gx2^2)/2 + m2*cos(2*qa2)*Gx2*Gy2 + ...
(m3*sin(2*qa2 + 2*qa3)*Gx3^2)/2 + m3*cos(2*qa2 + 2*qa3)*Gx3*Gy3 + ...
m3*sin(2*qa2 + qa3)*Gx3*L2 - (m2*sin(2*qa2)*Gy2^2)/2 - ...
(m3*sin(2*qa2 + 2*qa3)*Gy3^2)/2 + m3*cos(2*qa2 + qa3)*Gy3*L2 + ...
(m3*sin(2*qa2)*L2^2)/2 + Ixy3*cos(2*qa2 + 2*qa3) - ...
(Ixx3*sin(2*qa2 + 2*qa3))/2 + (Iyy3*sin(2*qa2 + 2*qa3))/2 + ...
Ixy2*cos(2*qa2) - (Ixx2*sin(2*qa2))/2 + (Iyy2*sin(2*qa2))/2) + ...
qa2d*((Ixz3*cos(qa2 + qa3))/2 - (Iyz3*sin(qa2 + qa3))/2 + ...
(Ixz2*cos(qa2))/2 - (Iyz2*sin(qa2))/2 + ...
(Gx3*Gz3*m3*cos(qa2 + qa3))/2 - (Gy3*Gz3*m3*sin(qa2 + qa3))/2 + ...
(Gx2*Gz2*m2*cos(qa2))/2 + (Gz3*L2*m3*cos(qa2))/2 - ...
(Gy2*Gz2*m2*sin(qa2))/2) + qa3d*((Ixz3*cos(qa2 + qa3))/2 - ...
(Iyz3*sin(qa2 + qa3))/2 + (Gx3*Gz3*m3*cos(qa2 + qa3))/2 - ...
(Gy3*Gz3*m3*sin(qa2 + qa3))/2)) - ...
qa3d*(qa1d*((Gy3*L2*m3*cos(qa3))/2 + (Gx3*L2*m3*sin(qa3))/2) + ...
qa2d*((3*Gy3*L2*m3*cos(qa3))/2 + (3*Gx3*L2*m3*sin(qa3))/2) + ...
qa3d*(Gy3*L2*m3*cos(qa3) + Gx3*L2*m3*sin(qa3))); ...
qa1d*(qa1d*(Ixy3*cos(2*qa2 + 2*qa3) - (Ixx3*sin(2*qa2 + 2*qa3))/2 + ...
(Iyy3*sin(2*qa2 + 2*qa3))/2 + (Gx3^2*m3*sin(2*qa2 + 2*qa3))/2 - ...
(Gy3^2*m3*sin(2*qa2 + 2*qa3))/2 + (Gy3*L2*m3*cos(qa3))/2 + ...
(Gx3*L2*m3*sin(qa3))/2 + (Gy3*L2*m3*cos(2*qa2 + qa3))/2 + ...
(Gx3*L2*m3*sin(2*qa2 + qa3))/2 + Gx3*Gy3*m3*cos(2*qa2 + 2*qa3)) + ...
qa2d*((Ixz3*cos(qa2 + qa3))/2 - (Iyz3*sin(qa2 + qa3))/2 + ...
(Gx3*Gz3*m3*cos(qa2 + qa3))/2 - (Gy3*Gz3*m3*sin(qa2 + qa3))/2) + ...
qa3d*((Ixz3*cos(qa2 + qa3))/2 - (Iyz3*sin(qa2 + qa3))/2 + ...
(Gx3*Gz3*m3*cos(qa2 + qa3))/2 - (Gy3*Gz3*m3*sin(qa2 + qa3))/2)) + ...
qa2d*(qa2d*(Gy3*L2*m3*cos(qa3) + Gx3*L2*m3*sin(qa3)) + ...
qa3d*((Gy3*L2*m3*cos(qa3))/2 + (Gx3*L2*m3*sin(qa3))/2))];
```

```
% The gravity vector, G
g = 9.81;
GSim = [0; g*m2*(Gx2*cos(qa2) - Gy2*sin(qa2)) + ...
    g*m3*(Gx3*cos(qa2 + qa3) - Gy3*sin(qa2 + qa3) + L2*cos(qa2)); ...
    g*m3*(Gx3*cos(qa2 + qa3) - Gy3*sin(qa2 + qa3))];

% The elastic torque vector, K
K = diag([Kplastic Kmetal Kmetal]);
KSim = K*([qa1;qa2;qa3]-[n1*qm1;n2*qm2;n3*qm3]);

% The damping vector, D
D = diag([Dplastic Dmetal Dmetal]);
DSim = D*([qa1d;qa2d;qa3d]-[n1*qm1d;n2*qm2d;n3*qm3d]);

% The friction vector, F
F = diag([Ffric1 Ffric2 Ffric3]);
FSim = F*[n1*qm1d;n2*qm2d;n3*qm3d];

% The motor mass matrix, B
Bsim = diag([Imzz1/n1^2 Imzz2/n2^2 Imzz3/n3^2]);

% The torque input
tau=[tau1/n1;tau2/n2;tau3/n3];

% Calculate the angle accelerations
qadd = MSim\(-CSim-GSim-KSim-DSim);
qmdd = Bsim\(tau+KSim+DSim-FSim);
qa1dd = qadd(1);
qa2dd = qadd(2);
qa3dd = qadd(3);
qm1dd = qmdd(1);
qm2dd = qmdd(2);
qm3dd = qmdd(3);
```

D Trajectory planner

```
function angles = calctrajectory(q_i,B,vmax,N,L1,L2,L3)
%CALCTRAJECTORY Calculates a trajectory from A to B.
%
% q_i = initial angles
% B = Stop position
% vmax ~= vmean = Used for calculating the stoptime (mean velocity).
% L1,L2,L3 = Length of arms.

% Start point
A = forward(q_i(1),q_i(2),q_i(3),L1,L2,L3);
% The direction vector
S_AB = B-A;
% Length of the trajectory
L = norm(S_AB);
% Direction vector (normalized)
S_hat = S_AB/L;
```

```
% Start and stop time
ti = 0; tf = L/vmax;
% time vector
t = linspace(ti,tf,N+1);
% start vector for the quintic function
d = [0, 0, 0, 0, L, 0, 0, tf];
% Calculates the polynomial trajectory using a quintic polynomial
[AB, ~, ~, ~] = quintic(d, t);
% Positions along AB (local coords)
S_t = S_hat*AB;
% Positions along AB (global coords)
X_t = repmat(A,1,N+1) + S_t;

% Calculates the trajectory. And checks if it is ok.
% If q_i(3) = 0 there are 2 different trajectories possible. If the first
% one fails, the seccond one is checked. This is done by sending an the
% angle +- 0.3
if (q_i(3) == 0)
    q_t = chooseAngles([q_i(1:2) 0.3],X_t,N,L1,L2,L3);
    disp('111');
    if (length(q_t) ~= N+1)
        q_t = chooseAngles([q_i(1:2) -0.3],X_t,N,L1,L2,L3);
        disp('222')
    end
else
    q_t = chooseAngles(q_i,X_t,N,L1,L2,L3);
end

% Inserts q(t) & t in a struct
angles.time = t';
angles.signals.values(:,1) = real(q_t(1,:))';
angles.signals.values(:,2) = real(q_t(2,:))';
angles.signals.values(:,3) = real(q_t(3,:))';
save('anglesdata.mat', 'angles');
end

function [qd, vd, ad, a] = quintic(d, t)
%QUINTIC Computes a quintic polynomial reference trajectory.
%
% q_i = initial position
% v_i = initial velocity
% a_i = initial acceleration
% q_f = final position
% v_f = final velocity
% a_f = final acceleration
% t_i = initial time
% t_f = final time

q_i = d(1); v_i = d(2); a_i = d(3); t_i = d(4);
q_f = d(5); v_f = d(6); a_f = d(7); t_f = d(8);

c = ones(size(t));
```

```
M = [ 1 t_i t_i^2 t_i^3 t_i^4 t_i^5;
      0 1 2*t_i 3*t_i^2 4*t_i^3 5*t_i^4;
      0 0 2 6*t_i 12*t_i^2 20*t_i^3;
      1 t_f t_f^2 t_f^3 t_f^4 t_f^5;
      0 1 2*t_f 3*t_f^2 4*t_f^3 5*t_f^4;
      0 0 2 6*t_f 12*t_f^2 20*t_f^3];
%
b=[q_i; v_i; a_i; q_f; v_f; a_f];
a = M\b;
%
% qd = position trajectory
% vd = velocity trajectory
% ad = acceleration trajectory
%
qd = a(1).*c + a(2).*t +a(3).*t.^2 + a(4).*t.^3 +a(5).*t.^4 + a(6).*t.^5;
vd = a(2).*c +2*a(3).*t +3*a(4).*t.^2 +4*a(5).*t.^3 +5*a(6).*t.^4;
ad = 2*a(3).*c + 6*a(4).*t +12*a(5).*t.^2 +20*a(6).*t.^3;

end
```

E Functions in the simulation model

```
function [y1, y2, y3] = fcn(qa1,qa2,qa3,qa1d,qa2d,qa3d,qa1dd,qa2dd,qa3dd,parameter
% FCN Transforms arm angles to motor angles.

% Loading the robot parameters from workspace
m1 = parameters(1);
m2 = parameters(2);
m3 = parameters(3);
n1 = parameters(4);
n2 = parameters(5);
n3 = parameters(6);
Gx1 = parameters(7);
Gy1 = parameters(8);
Gz1 = parameters(9);
Gx2 = parameters(10);
Gy2 = parameters(11);
Gz2 = parameters(12);
Gx3 = parameters(13);
Gy3 = parameters(14);
Gz3 = parameters(15);
Ixx1 = parameters(16);
Ixx2 = parameters(17);
Ixx3 = parameters(18);
Iyy1 = parameters(19);
Iyy2 = parameters(20);
Iyy3 = parameters(21);
Izz1 = parameters(22);
Izz2 = parameters(23);
Izz3 = parameters(24);
Ixy1 = parameters(25);
Ixy2 = parameters(26);
```

```
Ixy3 = parameters(27);
Ixz1 = parameters(28);
Ixz2 = parameters(29);
Ixz3 = parameters(30);
Iyz1 = parameters(31);
Iyz2 = parameters(32);
Iyz3 = parameters(33);
L1 = parameters(34);
L2 = parameters(35);
L3 = parameters(36);
Imzz1 = parameters(37);
Imzz2 = parameters(38);
Imzz3 = parameters(39);
Kplastic = parameters(40);
Kmetal = parameters(41);
Dplastic = parameters(42);
Dmetal = parameters(43);
Ffric1 = parameters(44);
Ffric2 = parameters(45);
Ffric3 = parameters(46);

% The mass matrix, M
MSim = [ Ixx2/2 + Ixx3/2 + Iyy2/2 + Iyy3/2 + Izz1 - ...
    (Ixx3*cos(2*qa2 + 2*qa3))/2 + (Iyy3*cos(2*qa2 + 2*qa3))/2 - ...
    Ixy3*sin(2*qa2 + 2*qa3) + Gx1^2*m1 + Gy1^2*m1 + (Gx2^2*m2)/2 + ...
    (Gy2^2*m2)/2 + (Gx3^2*m3)/2 + Gz2^2*m2 + (Gy3^2*m3)/2 + Gz3^2*m3 + ...
    (L2^2*m3)/2 - (Ixx2*cos(2*qa2))/2 + (Iyy2*cos(2*qa2))/2 - ...
    Ixy2*sin(2*qa2) + (Gx2^2*m2*cos(2*qa2))/2 - (Gy2^2*m2*cos(2*qa2))/2 ...
    + (L2^2*m3*cos(2*qa2))/2 + (Gx3^2*m3*cos(2*qa2 + 2*qa3))/2 - ...
    (Gy3^2*m3*cos(2*qa2 + 2*qa3))/2 + Gx3*L2*m3*cos(qa3) - ...
    Gy3*L2*m3*sin(qa3) + Gx3*L2*m3*cos(2*qa2 + qa3) - ...
    Gy3*L2*m3*sin(2*qa2 + qa3) - Gx2*Gy2*m2*sin(2*qa2) - ...
    Gx3*Gy3*m3*sin(2*qa2 + 2*qa3), - Iyz3*cos(qa2 + qa3) - ...
    Ixz3*sin(qa2 + qa3) - Iyz2*cos(qa2) - Ixz2*sin(qa2) - ...
    Gy3*Gz3*m3*cos(qa2 + qa3) - Gx3*Gz3*m3*sin(qa2 + qa3) - ...
    Gy2*Gz2*m2*cos(qa2) - Gx2*Gz2*m2*sin(qa2) - Gz3*L2*m3*sin(qa2), ...
    - Iyz3*cos(qa2 + qa3) - Ixz3*sin(qa2 + qa3) - ...
    Gy3*Gz3*m3*cos(qa2 + qa3) - Gx3*Gz3*m3*sin(qa2 + qa3); ...
    - Iyz3*cos(qa2 + qa3) - Ixz3*sin(qa2 + qa3) - Iyz2*cos(qa2) - ...
    Ixz2*sin(qa2) - Gy3*Gz3*m3*cos(qa2 + qa3) - ...
    Gx3*Gz3*m3*sin(qa2 + qa3) - Gy2*Gz2*m2*cos(qa2) - ...
    Gx2*Gz2*m2*sin(qa2) - Gz3*L2*m3*sin(qa2), m2*Gx2^2 + m3*Gx3^2 + ...
    2*m3*cos(qa3)*Gx3*L2 + m2*Gy2^2 + m3*Gy3^2 - 2*m3*sin(qa3)*Gy3*L2 + ...
    m3*L2^2 + Izz2 + Izz3, m3*Gx3^2 + L2*m3*cos(qa3)*Gx3 + m3*Gy3^2 - ...
    L2*m3*sin(qa3)*Gy3 + Izz3; - Iyz3*cos(qa2 + qa3) - ...
    Ixz3*sin(qa2 + qa3) - Gy3*Gz3*m3*cos(qa2 + qa3) - ...
    Gx3*Gz3*m3*sin(qa2 + qa3),m3*Gx3^2 + L2*m3*cos(qa3)*Gx3 + ...
    m3*Gy3^2 - L2*m3*sin(qa3)*Gy3 + Izz3 ,m3*Gx3^2 + m3*Gy3^2 + Izz3 ];

% The centrifugal and coriolis vector, C
CSim = [- qa3d*(qa1d*(Ixy3*cos(2*qa2 + 2*qa3) - ...
    (Ixx3*sin(2*qa2 + 2*qa3))/2 + (Iyy3*sin(2*qa2 + 2*qa3))/2 + ...
```

```
(Ixz3*cos(qa2 + qa3))/2 - (Iyz3*sin(qa2 + qa3))/2 + ...
(Gx3^2*m3*sin(2*qa2 + 2*qa3))/2 - (Gy3^2*m3*sin(2*qa2 + 2*qa3))/2 ...
+ (Gx3*Gz3*m3*cos(qa2 + qa3))/2 - (Gy3*Gz3*m3*sin(qa2 + qa3))/2 + ...
(Gy3*L2*m3*cos(qa3))/2 + (Gx3*L2*m3*sin(qa3))/2 + ...
(Gy3*L2*m3*cos(2*qa2 + qa3))/2 + (Gx3*L2*m3*sin(2*qa2 + qa3))/2 + ...
Gx3*Gy3*m3*cos(2*qa2 + 2*qa3)) + qa2d*(Ixz3*cos(qa2 + qa3) - ...
Iyz3*sin(qa2 + qa3) + Gx3*Gz3*m3*cos(qa2 + qa3) - ...
Gy3*Gz3*m3*sin(qa2 + qa3)) + qa3d*(Ixz3*cos(qa2 + qa3) - ...
Iyz3*sin(qa2 + qa3) + Gx3*Gz3*m3*cos(qa2 + qa3) - ...
Gy3*Gz3*m3*sin(qa2 + qa3))) - qa2d*(qa2d*(Ixz3*cos(qa2 + qa3) - ...
Iyz3*sin(qa2 + qa3) + Ixz2*cos(qa2) - Iyz2*sin(qa2) + ...
Gx3*Gz3*m3*cos(qa2 + qa3) - Gy3*Gz3*m3*sin(qa2 + qa3) + ...
Gx2*Gz2*m2*cos(qa2) + Gz3*L2*m3*cos(qa2) - Gy2*Gz2*m2*sin(qa2)) + ...
qa3d*(Ixz3*cos(qa2 + qa3) - Iyz3*sin(qa2 + qa3) + (Ixz2*cos(qa2))/2 ...
- (Iyz2*sin(qa2))/2 + Gx3*Gz3*m3*cos(qa2 + qa3) - ...
Gy3*Gz3*m3*sin(qa2 + qa3) + (Gx2*Gz2*m2*cos(qa2))/2 + ...
(Gz3*L2*m3*cos(qa2))/2 - (Gy2*Gz2*m2*sin(qa2))/2) + ...
qa1d*((m2*sin(2*qa2)*Gx2^2)/2 + m2*cos(2*qa2)*Gx2*Gy2 + ...
(Gz2*m2*cos(qa2)*Gx2)/2 + (m3*sin(2*qa2 + 2*qa3)*Gx3^2)/2 + ...
m3*cos(2*qa2 + 2*qa3)*Gx3*Gy3 + m3*sin(2*qa2 + qa3)*Gx3*L2 + ...
(Gz3*m3*cos(qa2 + qa3)*Gx3)/2 - (m2*sin(2*qa2)*Gy2^2)/2 - ...
(Gz2*m2*sin(qa2)*Gy2)/2 - (m3*sin(2*qa2 + 2*qa3)*Gy3^2)/2 + ...
m3*cos(2*qa2 + qa3)*Gy3*L2 - (Gz3*m3*sin(qa2 + qa3)*Gy3)/2 + ...
(m3*sin(2*qa2)*L2^2)/2 + (Gz3*m3*cos(qa2)*L2)/2 + ...
Ixy3*cos(2*qa2 + 2*qa3) - (Ixx3*sin(2*qa2 + 2*qa3))/2 + ...
(Iyy3*sin(2*qa2 + 2*qa3))/2 + (Ixz3*cos(qa2 + qa3))/2 - ...
(Iyz3*sin(qa2 + qa3))/2 + (Ixz2*cos(qa2))/2 - (Iyz2*sin(qa2))/2 + ...
Ixy2*cos(2*qa2) - (Ixx2*sin(2*qa2))/2 + (Iyy2*sin(2*qa2))/2)); ...
qa1d*(qa1d*((m2*sin(2*qa2)*Gx2^2)/2 + m2*cos(2*qa2)*Gx2*Gy2 + ...
(m3*sin(2*qa2 + 2*qa3)*Gx3^2)/2 + m3*cos(2*qa2 + 2*qa3)*Gx3*Gy3 ...
+ m3*sin(2*qa2 + qa3)*Gx3*L2 - (m2*sin(2*qa2)*Gy2^2)/2 - ...
(m3*sin(2*qa2 + 2*qa3)*Gy3^2)/2 + m3*cos(2*qa2 + qa3)*Gy3*L2 + ...
(m3*sin(2*qa2)*L2^2)/2 + Ixy3*cos(2*qa2 + 2*qa3) - ...
(Ixx3*sin(2*qa2 + 2*qa3))/2 + (Iyy3*sin(2*qa2 + 2*qa3))/2 + ...
Ixy2*cos(2*qa2) - (Ixx2*sin(2*qa2))/2 + (Iyy2*sin(2*qa2))/2) + ...
qa2d*((Ixz3*cos(qa2 + qa3))/2 - (Iyz3*sin(qa2 + qa3))/2 + ...
(Ixz2*cos(qa2))/2 - (Iyz2*sin(qa2))/2 + ...
(Gx3*Gz3*m3*cos(qa2 + qa3))/2 - (Gy3*Gz3*m3*sin(qa2 + qa3))/2 + ...
(Gx2*Gz2*m2*cos(qa2))/2 + (Gz3*L2*m3*cos(qa2))/2 - ...
(Gy2*Gz2*m2*sin(qa2))/2) + qa3d*((Ixz3*cos(qa2 + qa3))/2 - ...
(Iyz3*sin(qa2 + qa3))/2 + (Gx3*Gz3*m3*cos(qa2 + qa3))/2 - ...
(Gy3*Gz3*m3*sin(qa2 + qa3))/2)) - qa3d*(qa1d*((Gy3*L2*m3*cos(qa3))/2 ...
+ (Gx3*L2*m3*sin(qa3))/2) + qa2d*((3*Gy3*L2*m3*cos(qa3))/2 + ...
(3*Gx3*L2*m3*sin(qa3))/2) + qa3d*(Gy3*L2*m3*cos(qa3) + ...
Gx3*L2*m3*sin(qa3)));qa1d*(qa1d*(Ixy3*cos(2*qa2 + 2*qa3) - ...
(Ixx3*sin(2*qa2 + 2*qa3))/2 + (Iyy3*sin(2*qa2 + 2*qa3))/2 + ...
(Gx3^2*m3*sin(2*qa2 + 2*qa3))/2 - (Gy3^2*m3*sin(2*qa2 + 2*qa3))/2 ...
+ (Gy3*L2*m3*cos(qa3))/2 + (Gx3*L2*m3*sin(qa3))/2 + ...
(Gy3*L2*m3*cos(2*qa2 + qa3))/2 + (Gx3*L2*m3*sin(2*qa2 + qa3))/2 + ...
Gx3*Gy3*m3*cos(2*qa2 + 2*qa3)) + qa2d*((Ixz3*cos(qa2 + qa3))/2 - ...
(Iyz3*sin(qa2 + qa3))/2 + (Gx3*Gz3*m3*cos(qa2 + qa3))/2 - ...
(Gy3*Gz3*m3*sin(qa2 + qa3))/2) + qa3d*((Ixz3*cos(qa2 + qa3))/2 - ...
```

```
    (Iyz3*sin(qa2 + qa3))/2 + (Gx3*Gz3*m3*cos(qa2 + qa3))/2 - ...
    (Gy3*Gz3*m3*sin(qa2 + qa3))/2)) + qa2d*(qa2d*(Gy3*L2*m3*cos(qa3) + ...
    Gx3*L2*m3*sin(qa3)) + qa3d*((Gy3*L2*m3*cos(qa3))/2 + ...
    (Gx3*L2*m3*sin(qa3))/2))];

% The gravity vector, G
g = 9.81;
GSim =    [0;
          g*m2*(Gx2*cos(qa2) - Gy2*sin(qa2)) + g*m3*(Gx3*cos(qa2 + qa3) ...
          - Gy3*sin(qa2 + qa3) + L2*cos(qa2)); g*m3*(Gx3*cos(qa2 + qa3) ...
          - Gy3*sin(qa2 + qa3))];

% The elastic torque vector, K
K = diag([Kplastic Kmetal Kmetal]);
KSim = K*([qa1;qa2;qa3]);

% The damping vector, D
D = diag([Dplastic Dmetal Dmetal]);
DSim = D*([qa1d;qa2d;qa3d]);

% Calculate outputs

y = MSim*([qa1dd;qa2dd;qa3dd])+ CSim + GSim + KSim + DSim;

y1 = y(1);
y2 = y(2);
y3 = y(3);
end

function [u2,u3] = fcn(qa2,qa3,parameters)
%FCN Calculates the feed forward.

g = 9.81;

m2 = parameters(2);
m3 = parameters(3);
Gx2 = parameters(10);
Gy2 = parameters(11);
Gx3 = parameters(13);
Gy3 = parameters(14);
L2 = parameters(35);

u2 = g*m2*(Gx2*cos(qa2) - Gy2*sin(qa2)) + g*m3*(Gx3*cos(qa2 + qa3) ...
    - Gy3*sin(qa2 + qa3) + L2*cos(qa2));

u3 = g*m3*(Gx3*cos(qa2 + qa3) - Gy3*sin(qa2 + qa3));

u2 = u2/2;

u3 = u3/2;

end
```