Modelling and Control of a Parallel Kinematic Robot

Kristofer Rosquist

Department of Automatic Control

Msc Thesis
ISRN LUTFD2/TFRT--5929--SE
ISSN 0280-5316

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

© 2013 by Kristofer Rosquist. All rights reserved.
Printed in Sweden by Media-Tryck
Lund 2013

Abstract

This Master's Thesis shows how to model the kinematics and forward dynamics for the parallel kinematic robot from ABB, *IRB340 FlexPicker,* and how to implement the control using software and hardware from B&R Automation. With conventional methods the forward kinematics, inverse kinematics and the kinematics for the velocities and accelerations of the robot are modelled. The forward dynamics, calculating the generated motor torques, of the robot is modelled with MapleSim and code generation of this model, to be implemented in the software from B&R Automation and used online during control, is used. In the software from B&R Automation, B&R Automation Studio, the kinematic model, path trajectories for the TCP with polynomial functions, an interface between the software and a PC and programs for communicating with the drives are also implemented together with a main program. To simplify, analyze and calculate equations Maple is used.

Experiments using the generated torque are made for comparison with the model and to improve it. The model is improved by measuring the friction torque and the torque caused by the inertia of the motor, a gear and one robot arm.

Because of the parallel structure of the robot the generated dynamic equation consists of complex differential equations which can overload the PLC during solving. To ensure stability of the dynamic model a second dynamic model with constraint stabilization is generated for use during slower movements. Despite the risk of starvation of the PLC the robot is moved at 10 g acceleration with good result comparing calculated and actual torque. Without using the desired feedforward from the dynamic equations, but instead using a simpler less dynamic feedforward method, the robot is accelerated at 16 g.

Acknowledgements

I want to thank the department of Automatic Control at LTH for having me at the department the entire duration of my thesis. I especially want to thank my supervisor, Anders Robertsson, who gave tips and guidance during the work. Also thanks to Klas Nilsson at the Department of Computer Science who gave some advice during one phase where a problem kept me from continuing with the work.

I also thank those from B&R Automation who answered my questions about B&R Automation Studio and the hardware by mail.

In the end I also would like to thank my family and friends. With the huge support from a few I managed to create this thesis which I am pleased with. Thank you!

Table of Contents

1 Introduction

1.1 Motivation

The ABB FlexPicker robot is a pick-and-place robot which is able to move at high accelerations with very high precision thanks to its parallel structure with three arms that have very light-weight forearms that connect together at the tool centre point of the robot. For moving the robot, position control will be needed for the motors moving the arms, and a dynamic model is needed for good performance. The dynamic model, to calculate the needed torque in the motors for feed forward control, is especially needed to be able to move the FlexPicker at the high accelerations it is designed for. In this thesis MapleSim is the tool that is used to investigate how a dynamic model and kinematic models for the robot can be created.

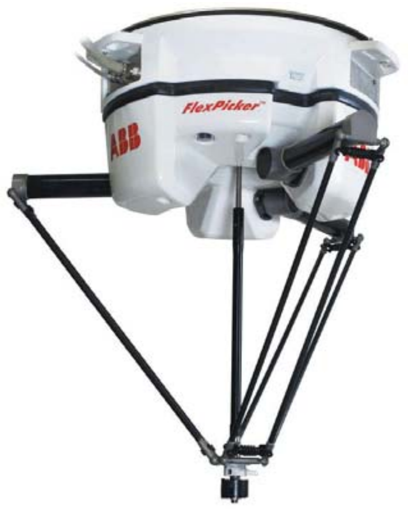


Figure 1.1 IRB 340 ABB FlexPicker robot [1].

There are many works on how the dynamic model of a parallel robot can be created. They are based on numerical methods like Newton-Euler classical procedure, Lagrange equations and virtual work [8], Jourdain's principle of virtual power [9] or a so called virtual spring approach [10] to mention a few. Used in several applications are also methods where the movement of the forearms of the robot are neglected in the model [30]. In this thesis a simulation of a parallel robot in MapleSim is used [13] and modified to create the forward dynamics of the robot not neglecting movements of any arms. By including all parts and having the equations in their algebraic form it is hoped that the solution will be more accurate than with the other methods and that the number of calculations can be kept limited for the model to be able to be used in a real-time application. Unfortunately the equations in the dynamical model are kept in algebraic form only until export, when the Jacobian in the model is converted to a numerical form and when a numerical differential equation solver is used.

By exporting the model with MapleSim the C-code for the created dynamic model will be generated. How to use this in the drives will then be investigated by implementing the dynamic and kinematic models and creating a program for movement of the robot in B&R Automation studio on a PC. The software is actuated with hardware from B&R; a PLC controller and B&R ACOPOS 1045 drives. Furthermore Maple will also be used together with MapleSim to analyze, calculate, and simplify equations.

1.2 Notations

Variables and parameters used in the thesis

l_1	Horizontal distance between two parallel arms
l_{13}	Vertical distance from base to the universal joint of 4th robot arm (Figure 1.1)
l_3	Length of the robot's forearms
l_4	Length of the robot's upper arms
l_{4r}	The vertical distance from the upper arms to the connection of the parallel arms.
l_{4c}	The distance to the mass-center for the upper arm.
R_A	Radius of the base plate centre to one of the three upper arms.
R_B	Radius of the travelling plate center to one of the three forearms
α_1	Rotation around the vertical axis for arm attached to motor 1.
α_2	Rotation around the vertical axis for arm attached to motor 2.
α_3	Rotation around the vertical axis for arm attached to motor 3.
θ_1	Angle of the first upper arm attached to the gear of motor 1.
θ_2	Angle of the second upper arm attached to the gear of motor 2.
θ_3	Angle of the third upper arm attached to the gear of motor 3.
θ_4	Angle of the third upper arm attached to the gear of motor 4.
θ_{m1}	Angle of motor 1.
θ_{m2}	Angle of motor 2.
θ_{m3}	Angle of motor 3.
θ_{m4}	Angle of motor 4.
β_1	Angle for the forarms as defined in Figure 4.7.
β_2	Angle between upper arm and forearms as defined in Figure 4.6 and Figure 12.1.
m_U	Mass of the robot's upper arms.
m_{l3}	Mass of the robot's forearms.
I_{l4xx}	The moment of inertia around the axis of rotation for the upper arms.
I_{l3zzxx}	The moment of inertia for forearms around x-axis and z-axis.
I_{motor}	The inertia of the motor connected to arms 1-3.
r_G	The gear ratio of the gears of arms 1-3.
ω_0	Zero speed for a motor.
ω_1	Low speed for the motor defined in Chapter 4.4.
ω_2	Medium speed for the motor defined in Chapter 4.4.
ω_3	High speed for the motor defined in Chapter 4.4.

2 Software and Hardware

2.1 The FlexPicker

The robot in this thesis is an IRB 340 ABB FlexPicker and except from some minor differences (different types of springs and different load at the tool centre point) it looks like Figure 1.1. The robot has four degrees of freedom, where the first three is a translational motion of the tool centre point. The tool centre point is located at the lower part of the robot and is moved by using the three in the picture horizontal arms, which in this thesis are called upper arms, and they are connected to three gears connecting to the motors at the base. The motors actuate the rotation of the upper arms which at the end are connected to two light-weight parallel arms with spherical joints. Two springs are located between every pair of parallel arms, one at the top and one at the bottom (Figure 2.1). At the bottom of these forearms there is a connection to the travelling plate which consists of a triangular metal plate with a hole in the centre allowing the fourth robot-arm a passage. This fourth arm can be used to rotate the tool centre point located below the travelling plate, and hereby implementing its fourth degree of freedom, and is actuated by a motor at the centre of the base. The arm is a telescopic arm which allows it to extend to be able to follow the translational motion of the travelling plate.

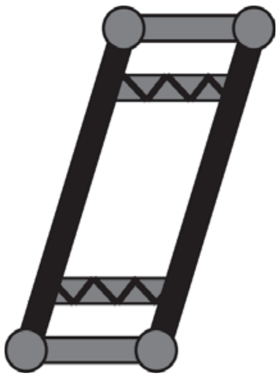


Figure 2.1 Visualization of a forearm-pair of the FlexPicker robot with springs.

2.2 MapleSim

MapleSim is a physical modelling and simulations tool that uses symbolic computation [2]. This means that all the equations in the model are kept in their fully mathematical form without any approximations when the model is created and exported to any possible application. The user could then change any parameters of the model in the application without needing to recreate and export the model from MapleSim again.The symbolic equations are also available for analysis in MapleSim and they are simplified to reduce computation time. To reduce computation time even further before exporting the model optimized code generation is used [2].

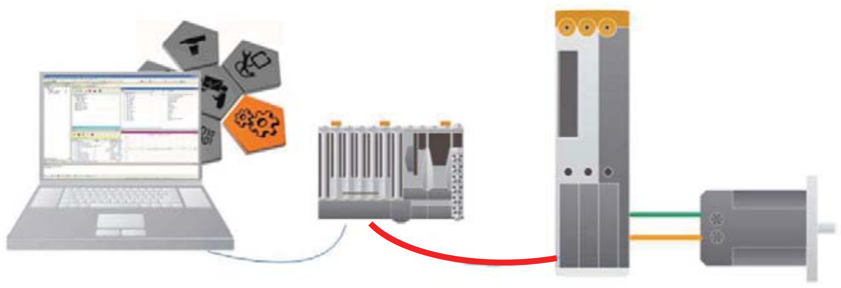


Figure 2.2 B&R hardware and software used to drive a motor [3].

2.3 Drive Design

The different components needed to drive one motor of the robot, sometimes referred to as an axis, are shown in Figure 2.2. There are four components: A PC, a PLC, a drive and a motor.

2.3.1 PC and PLC

To the left is a Windows PC with the software B&R Automation studio installed. On this software the application is programmed and the program together with the hardware configuration is transferred to the PLC, the next component. The PLC, programmable logic controller, is of the type X20CP1486 and from this the communication to the drive takes place through POWERLINK, the red cable in the figure. How the data is transferred over the POWERLINK to the drive from the user application is controlled by the NC Manager on the PLC. Parameters and commands are sent from the controller to the servo drive and status data is sent from the servo drive to the controller.

2.3.2 Servo drives and motor

The servo drive used is the B&R ACOPOS 1045 (third component in Figure 2.2), and to be able to calibrate it with different types of motors and applications two sets of parameters have to be set in Automation studio and then sent and saved on the drives:

NC Init Module - Basic axis parameters used by the NC manager to initialize the axis reference when starting up the controller on the drive [3]. For example the limits for positions, speeds and acceleration and the control parameters for the controller on the drive.

NC Parameter Tables – Specific axis parameters. Here parameters for the motor and other needed specific parameters can be specified.

In this thesis three drives are used which are all connected to the POWERLINK. The three drives are connected to the three motors, in the same way as one motor is connected to the drive to the right in the figure. The connection is represented with a green line for the angle measurement sensor (a resolver is used). The resolver is used to measure actual position of the motor. The orange line represents the current from the drive that actuates the movement of the motor.

3 Kinematics

The first part of making the model of the robot is to make the kinematic model. The kinematics is a geometric description of the robot, and there are two types, the forward kinematic and the inverse kinematic. The forward kinematics determines the position and orientation of the tool center point given the values for the actuated joint angles of the robot. The inverse kinematics determines the values of the actuated joint angles given the position and orientation of the tool center point. Both the forward and the inverse kinematics and the equations are from [4][5] where no calculation on a fourth arm is made. For the fourth arm the angle of the tool centre point and the angle for the fourth arm are the same which means no calculation is needed to get either orientation of the tool centre point or the actuated joint angle for the fourth arm. However to get the forward kinematics and the inverse kinematics the following assumptions are made:

- The travelling plate (Figure 3.1) which is connected to the 6 forearms is always parallell to the base which means that the pairs of parallel rods can be replaced with single rods in the calculations without having any effect on the kinematics [6].
- The revolute joints at the baseplate which are actuated by motors are symmetrically placed on a circle, 120° between them. Also the joints between the three forearms and the travelling plate are placed symetrically along a circle. To simplify calculations for forward kinematics the travelling plate can then be replaced by a point P which the forarms are connected to (Figure 3.2) [6].

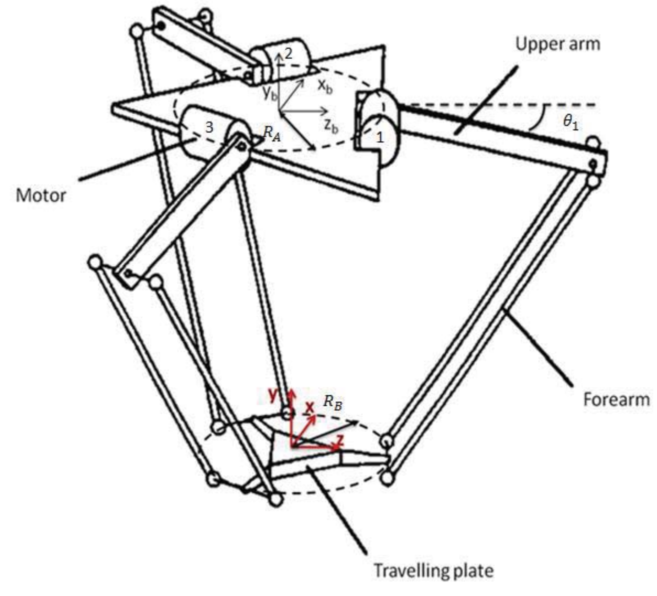


Figure 3.1 Simplified picture of FlexPicker without the fourth arm and with the definition of the coordinate system by placing the origin at the base-plate [7].

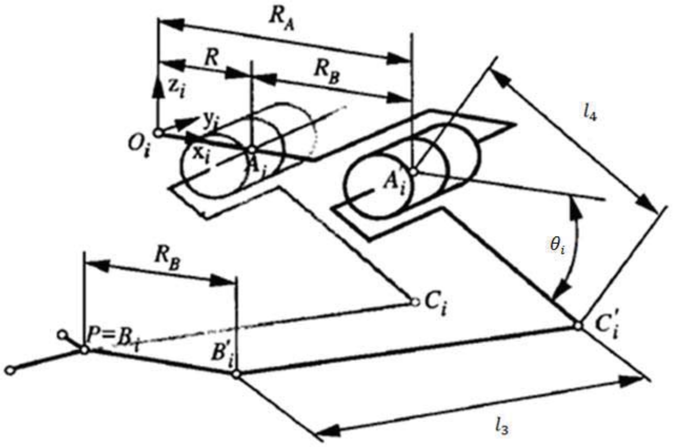


Figure 3.2 Simplification of the geometrics of a robot arm [7].

3.1 Forward and inverse kinematics

Instead of calculating the position of the tool centre point of the robot the position of the centre of the travelling plate is described. The tool centre point is a small distance along the y-axis below the travelling plate since the travelling plate is parallel to the base.

3.1.1 Forward kinematics

By using the simplification that all three forearms meet at one point *B*, called *P* in Figure 3.2, each arm is moved according to Figure 3.2 along the x-axis, which means along the z-axis in the coordinate system used in this thesis from the definition in Figure 3.1. The index *i=1,2,3* indicates the three robot arms. The coordinates for the virtual elbow in this simplification is then:

$$c_i = \begin{bmatrix} 0 \\ -l_4 \sin(\theta_i) \\ R + l_4 \cos(\theta_i) \end{bmatrix} \tag{1}$$

where $R = R_A - R_B$. In general this is only valid for arm 1 but the coordinates for the other virtual elbows, described in the base frame *{R}* (origin at the base-plate in Figure 3.1), obtained by multiplying with a rotation matrix with rotation around the y-axis:

$$C_i = {}^{R}_{y}R_i c_i \tag{2}$$

where

$${}^{R}_{y}R_i = \begin{bmatrix} \cos(\alpha_i) & 0 & \sin(\alpha_i) \\ 0 & 1 & 0 \\ -\sin(\alpha_i) & 0 & \cos(\alpha_i) \end{bmatrix} \tag{3}$$

The matrix describing all three virtual elbow points is given by:

$$C = [C_1 \quad C_2 \quad C_3] = \begin{bmatrix} {}^{R}_{y}R_1 c_1 & {}^{R}_{y}R_2 c_2 & {}^{R}_{y}R_3 c_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix}^T$$

$$= \begin{bmatrix} 0 & -z_2/\tan(\frac{\pi}{6}) & z_3/\tan(\frac{\pi}{6}) \\ -l_4 \sin(\theta_1) & -l_4 \sin(\theta_2) & -l_4 \sin(\theta_3) \\ \cos(\alpha_1)(R + l_4 \cos(\theta_1)) & \cos(\alpha_2)(R + l_4 \cos(\theta_2)) & \cos(\alpha_1)(R + l_4 \cos(\theta_3)) \end{bmatrix}^T \tag{4}$$

The dependency from the x-values and z-values comes from that the rotation angle around the y-axis between the first and the second arm is $\alpha_2 = 2\pi/3$ and thereby $2\pi/3 - \pi/2 = \pi/6$ radians from the second arm to the positive x-axis (Figure 3.3). The angle from the third arm to the negative z-axis is $-\pi/6$.

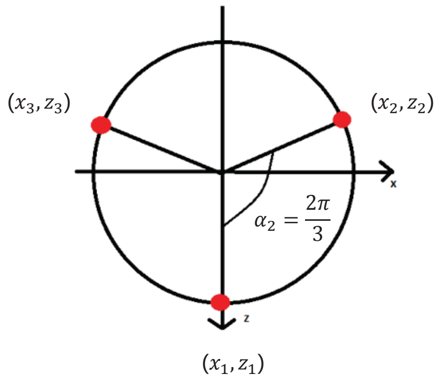


Figure 3.3 Drawing of the robot seen from above.

From these three points three spheres can be created with the points as centre points and with radius l_3. These spheres will have two intersection points where the point with negative y-coordinate is the position of the travelling plate and thereby the solution to the forward kinematics.

From the equation for a sphere

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \tag{5}$$

The three constraint equations that need to be solved are:

$$\|BC_i\|^2 = {l_3}^2 \tag{6}$$

Written with equation (5):

$$x^2 + (y - y_1)^2 + (z - z_1)^2 = {l_3}^2 \tag{7}$$

$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = {l_3}^2 \tag{8}$$

$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = {l_3}^2 \tag{9}$$

Let:

$$w_i = x_i^2 + y_i^2 + z_i^2 \tag{10}$$

With (7) subtracted by (8) rewritten with (10) becomes

$$x_2x + (y_1 - y_2)y + (z_1 - z_2)z = \frac{(w_1 - w_2)}{2} \tag{11}$$

And (7) subtracted by (9) rewritten with (10) becomes

$$x_3x + (y_1 - y_3)y + (z_1 - z_3)z = \frac{(w_1 - w_3)}{2} \tag{12}$$

By using the equations (11) and (12), x and z can be eliminated to get:

$$x = a_1y + b_1 \tag{13}$$

$$z = a_2y + b_2 \tag{14}$$

with

$$a_1 = 1/d \cdot [(y_2 - y_1)(z_3 - z_1) - (y_3 - y_1)(z_2 - z_1)]$$

$$b_1 = -1/(2d) \cdot [(w_2 - w_1)(z_3 - z_1) - (w_3 - w_1)(z_2 - z_1)]$$

$$b_2 = 1/(2d) \cdot [(w_2 - w_1)x_3 - (w_3 - w_1)x_2]$$

$$d = (z_2 - z_1)x_3 - (z_3 - z_1)x_2$$

By substituting the equation (13) and (14) in (7) a second order equation for y is obtained:

$$({a_1}^2 + a_2^2 + 1)y^2 + 2(a_1 + a_2(b_2 - z_1) - y_1)y + \left(b_1^2 + (b_2 - z_1)^2 - y_1^2 - {l_3}^2\right) = 0 \tag{15}$$

Since by definition the travelling plate has a negative y-coordinate the negative solution for y is chosen. From this a solution for x and z is obtained by inserting the solution for y in (13) and (14), respectively.

3.1.2 Inverse Kinematics

Since the robot's first upper arm by definition only rotates in the Y-Z – plane (revolute joint) a circle with radius l_4 will then be the area this is restricted to move within. The joint on the platform (${B'}_1$) is a universal joint and can move in any direction and the movement restriction for the forearm relative to the joint (${B'}_1$) is a sphere with centre at the joint and radius l_3. The intersection of this sphere and the YZ-plane is a circle with the centre in (${B''}_1$) with known radius. The point (${B''}_1$) is the projection of point (${B'}_1$) to the YZ-plane. One intersection of this circle and the circle created by the upper arm is the point (${C'}_1$). The second point can be dismissed as a solution since this configuration of the angle of the upper arm and the angle of the forearm is not possible for the robot. From this point (${C'}_1$) the rotation angle (θ_1) can be calculated. The travelling plate's position is

$$P = B = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{16}$$

The coordinates for the universal joint for the first arm is then

$$B'_1 = \begin{bmatrix} x \\ y \\ z + R_B \end{bmatrix} \tag{17}$$

The vector ($B'_1B''_1$) from the joint to the Y-Z plane in parallel to the x-axis is:

$$B'_1B''_1 = \begin{bmatrix} x \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

The coordinates for B''_1:

$$B''_1 = \begin{bmatrix} 0 \\ y \\ z + R_B \end{bmatrix} \Rightarrow B''_1C'_1{}^2 = (y_{c'1} - y)^2 + (z_{c'1} - (z + R_B))^2 \tag{19}$$

See Figure 3.4. Since $B''_1C'_1$ is an orthogonal projection of $B'_1C'_1$ to the YZ-plane, with B''_1 as projection of the point B'_1 , $B'_1B''_1C_1$ is a right-angled triangle:

$$(B'_1C'_1)^2 = (B'_1B''_1)^2 + (B''_1C'_1)^2 \tag{20}$$

$$(B'_1C'_1)^2 = l_3{}^2 \Rightarrow x^2 + (y_{c'1} - y)^2 + (z_{c'1} - (z + R_B))^2 = l_3{}^2 \tag{21}$$

From Figure 3.4:

$$(z_{c'1} - R_A)^2 + y_{c'1}{}^2 = l_4{}^2 \tag{22}$$

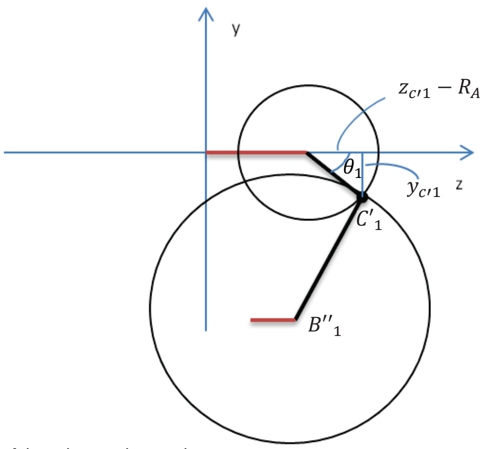


Figure 3.4 The projection of the robot on the yz-plane.

Putting equation for ${y_{c'1}}^2$ from equation (22) into equation (21) gives a solution for $y_{c'1}$ in the form:

$$y_{c'1} = a + bz_{c'1} \tag{23}$$

where $a = (x^2 + y^2 + (z - R_B)^2 + l4^2 - l3^2 - {R_A}^2)/(2y)$, $b = (R_A - z)/y$. And for $z_{c'1}$ a second degree equation has to be solved and the largest solution for $z_{c'1}$ is chosen. This is chosen because if the other solution would be chosen the angle between the upper arm and the forearm would be too large (see the other intersection of the circles in Figure 3.4). The angle θ_1, which is defined positive for negative $y_{c'1}$, is then calculated:

$$\theta_1 = \arctan\left(\frac{-y_{c'1}}{z_{c'1} - R_A}\right) \tag{24}$$

But if $z_{j1} - R_A < 0$ i.e. θ_1 is more than $\pi/2$, the solution found will not be the one above $\pi/2$ but instead the solution -π from the correct one. So π is added to the solution in these cases.

To get the inverse kinematics for the other angles the same principle is used, but for it to work one have to change coordinate system to a coordinate system where the two other arms respectively only moves in one plane. This coordinate change is applied to the travelling plate position by use of the rotation matrices. The coordinate change is:

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} = {}^R_yR_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = [x\cos(\alpha_i) + z\sin(\alpha_i) \quad 0 \quad -x\sin(\alpha_i) + z\cos(\alpha_i)]^T \tag{25}$$

3.2 The Jacobian matrix

These calculations are from [7] and starts by using the same simplifications as for the forward kinematics. Three constraint equations for the robot are

$$\|BC_i\|^2 - {l_3}^2 = 0 \tag{26}$$

for *i*=1,2,3

This constraint means the lengths of the forearm are always constant. If the vector BC_i is represented by s_i the Euclidian norm can be rewritten as $s_i^T s_i$. And equation (26) can be rewritten:

$$s_i^T s_i - l_3^2 = 0 \tag{27}$$

First time derivative:

$$s_i^T \dot{s}_i + \dot{s}_i^T s_i = 0 \tag{28}$$

Since this product is commutative this can be rewritten to:

$$s_i^T \dot{s}_i = 0 \tag{29}$$

$$\dot{s}_i = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} - {}^{R}_{y}R_i \begin{bmatrix}0\\ -l_4\cos(\theta_i)\\ -l_4\sin(\theta_i)\end{bmatrix}\dot{\theta}_i = \dot{B} - b_i\dot{\theta}_i \tag{30}$$

with $b_i = {}^{R}_{y}R_i \begin{bmatrix}0\\ -l_4\cos(\theta_i)\\ -l_4\sin(\theta_i)\end{bmatrix}$.

With equation (30) inserted in (29) and rewritten in matrix form for all three arms:

$$\begin{bmatrix}s_1^T\\ s_2^T\\ s_3^T\end{bmatrix}\dot{B} - \begin{bmatrix}s_1^T b_1 & 0 & 0\\ 0 & s_2^T b_2 & 0\\ 0 & 0 & s_3^T b_3\end{bmatrix}\dot{\theta} = \begin{bmatrix}0\\0\\0\end{bmatrix} \tag{31}$$

Rewritten:

$$\dot{B} = \begin{bmatrix}s_1^T\\ s_2^T\\ s_3^T\end{bmatrix}^{-1}\begin{bmatrix}s_1^T b_1 & 0 & 0\\ 0 & s_2^T b_2 & 0\\ 0 & 0 & s_3^T b_3\end{bmatrix}\dot{\theta} = J\dot{\theta} \tag{32}$$

Here the Jacobian matrix *J* is identified as the matrix transforming the joint velocities to travelling plate velocity. It depends both on the position of the travelling plate and the joint angles.

3.3 Velocity and acceleration kinematics

The expression (32) gives the velocity for the travelling plate, given the angle-velocities but the opposite is needed in the application. If the Jacobian is square and invertible the angle-velocities are given by:

$$\dot{\theta} = J^{-1}\dot{B} \tag{33}$$

Using Maple to Calculate J and J^{-1} and with simplification of the result gives

$$J^{-1} = \begin{bmatrix}J_1^{-1}\\ J_2^{-1}\\ J_3^{-1}\end{bmatrix} \tag{34}$$

where $J_i^{-1} = \frac{[(-x+\sin(\alpha_i)R+\sin(\alpha_i)l_4\cos(\theta_i)) \quad -(y+l_4\sin(\theta_i)) \quad (-z+\cos(\alpha_i)R+\cos(\alpha_i)l_4\cos(\theta_i))]}{l_4(\sin(\alpha_i)\sin(\theta_i)x-\sin(\theta_i)R+\cos(\theta_i)y+\cos(\alpha_i)\sin(\theta_i)z)}$, for *i*=1,2,3

The Jacobian is squared and also invertible as long as the denominators are not zero, so called singularity-free. This will not happen when keeping the robot within its working area.

To get the angle acceleration of the actuated joints, the equation (33) is differentiated in Maple (result in Appendix 14.5).

3.4 Motor angles

Since the robot arms are connected to the motors via gears, the joint angles, θ_i, described so far are not the actual angles for the motor. These angles are multiplied with the corresponging gear ratio, r_G for three first arms, to give the motor angles θ_{mi}.

4 Dynamics

To find the dynamic model of a multibody system there are in general three methods that are used; the Newton-Euler classical procedure, Lagrange equations with its multipliers and methods based on virtual work [8]. To derive the dynamic solution Jourdain's principle of virtual power can also be used [9]. However when dealing with a closed-loop mechanical system with closed kinematic chains (4.5.1) the solution is much more difficult than open chain mechanical systems [10]. Here the forward dynamic model of the robot is constructed in MapleSim which calculates the equation using graph theory approach [11][12]. One advantage of this method to create the equations of motions is that it allows separation of the equations how components interact and how they are connected which then leads to a computationally efficient set of symbolic equations. With some simplifications and conversion to ordinary differential equations by MapleSim the dynamic model is exported to C-code with the B&R connector application in the program. Implementation in a program in B&R Automation studio makes it possible to solve these differential equations simultaneously as controlling the robot.

4.1 MapleSim model

The forward dynamic model in MapleSim is constructed with rotation angles, θ_i, as inputs and torque at the four motors as outputs. The input angles are defined positive as in the previous definition in the inverse- and forward kinematics. The base frame for the coordinates, *{R},* is also the same as previously (Figure 3.1).

There is another MapleSim model [13] which is made in a similar way. This model used to design the robot DeltaBot™, a robot based on cable-actuated design. One thing the models have in common is to model the spherical joint connection between the upper arm and the forearm with revolute joints instead of spherical joints (other possibilities for the joints were tried but MapleSim couldn't generate the model in these cases). Otherwise some simplifications are made compared to the real robot (Figure 1.1):

- Instead of the double parallel arms in each forearm of the robot, single arms are used.
- No friction or elasticity is used in the model.

The overview of the model in MapleSim can be seen in Figure 4.1, where the blue filled arrows are the angle-inputs, rectangles with number 1 are subsystems for the three arms and number 2 is a subsystem for the middle, telescopic, arm. They are all connected to the travelling plate, which is represented by a point mass (black sphere in the figure). To the right is a fixed frame representing the origin of the global coordinate system and between the mass and the frame is a block that limits the travelling plate to translational movements in relation to the global coordinate system. And to the right are the torque values sent as output of the model.

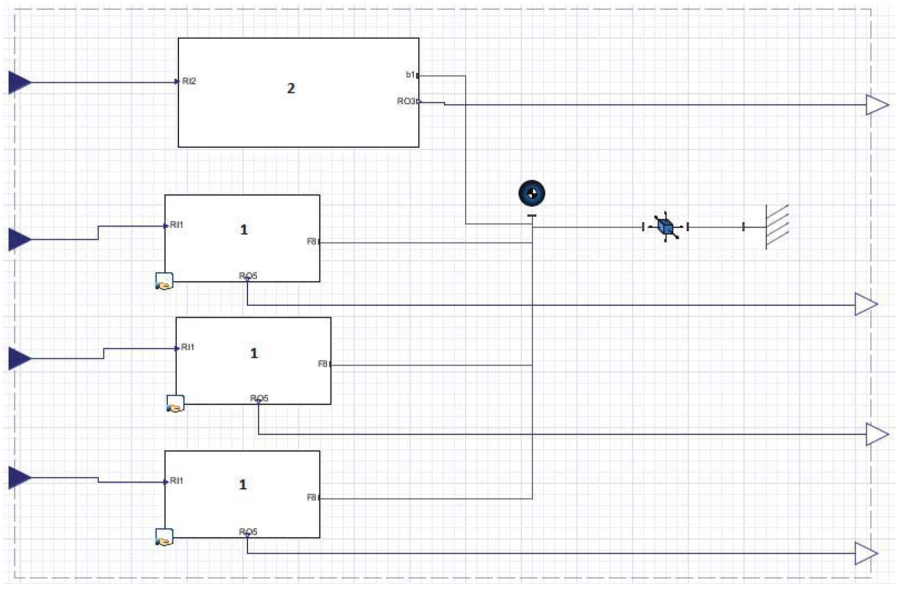


Figure 4.1 Overview of the forward dynamic model created in MapleSim with inputs and outputs.

4.1.1 First three robot arms

Figure 4.2 shows how the first three arms of the robot are created (represented by block 1 in Figure 4.1). They are made exactly the same way except for that they have one parameter α_i that differs. This parameter is used as before to decide which position and direction the arms are attached to the fixed frame at the x,z-plane (down left in Figure 4.2) with help of the rotation matrix (3) (see also $A_i{}'$ in Figure 3.2). From the left the input position (full blue triangle) is first multiplied with the gear ratio (white triangular gain-block) to get the actual reference-angle for the motor. This position is then differentiated to get angle velocity and acceleration in the next component. The input signal is converted from a Real-valued signal to a flange type, which is used to model rotation. This component thereby forces the output angle to follow the input signal. The torque that is needed to do this is extracted out of the model by the sensor (the circle with an arrow) and sent to the output (white triangle to the right). Connected to the sensor is an inertia component. This is connected to an ideal gear component and another intertia component. The first inertia component is to model the inertia of the motor and the inertia of the smaller wheel of the gear and the second one is to model the bigger wheel of the gear. The ideal gear component is modelled without any friction and since the inertia of the gear is unknown these values are set to zero.

The multibody system for the arm, with masses, torques etc, is modelled below the components for the actuator as described earlier. To the left is a fixed frame that decides starting position and rotation (α_i) which connects to the revolute joint that is rotated by the actuator around one axis. The upper arm in Figure 4.3 is attached to this joint.

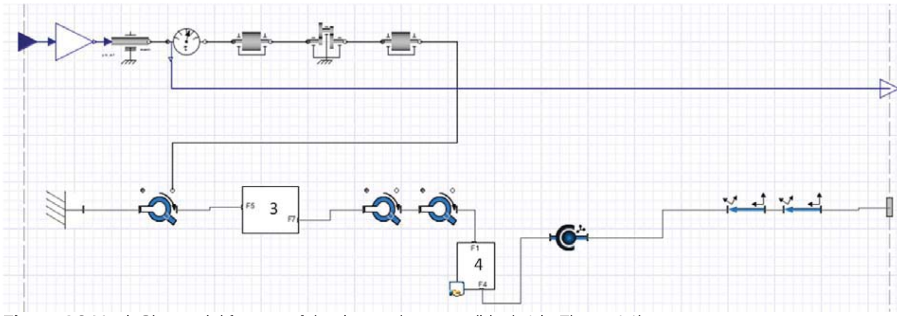


Figure 4.2 MapleSim model for one of the three robot arms (block 1 in Figure 4.1).

The mass is placed at the position of the masscenter of the upperarm between two massless rods. The upper arm also have a vertical rod since the robot joints of the parallell arms (modelled as singel arms) are not connected in the center of the upper arm but a distance down, l_{4r}, modelled by this rod (this differs compared to the kinematic model (Appendix 14.3)). The upper arm is then connected to two revolute joints that allow rotation for the forearm (Figure 4.4) around its local x, and z-axis* but no rotation around the y-axis (which is along the length of the forearm). These two joints model the actual spherical joint of the robot. The two masses on the right connected to the rod are representing springs that for the real robot is connecting the two parallel forearms. The lower part of the forearm is connected to a spherical joint which is connected to rods of total length R_B. The direction of these depends on the parameter α_i and they are connected to the travelling plate.

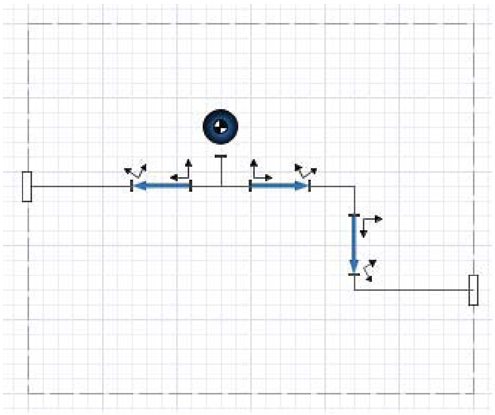


Figure 4.3 MapleSim model for the upper arms of one robot arm (block 3 in Figure 4.2).

*Each point mass has a local x,y,z-axis and the direction and lengths of the rods are in this model described in these local coordinate systems.

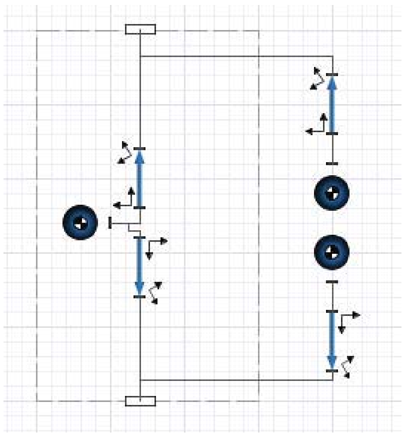


Figure 4.4 MapleSim model of the forearms of the robot (block 4 in Figure 4.2).

4.1.2 The fourth arm

The fourth and middle arm of the robot is attached a little bit higher up than the other arms (Figure 1.1). To this fixed frame, which is placed at the distance l_{13} above the base-frame, three revolute joints are connected (Figure 4.5). The first joint allows rotation around the y-axis and is actuated by the motor which is modelled in the same way as the other arms. The other two revolute joints model the universal joint of the robot that allows rotation around its x- and z-axis. The next part is the two parts of the arm that allows extension of the arm. They are modelled in the same way, with two rods and a mass at their respective mass-center. In between them is a prismatic joint (the blue object between the two white subsystems). Next are two revolute joints to model the other universal joint of the robot which connects to another subsystem, also modelled as the others. This connects to the travelling plate of the robot. The connection to the travelling plate is made through another revolute joint with rotation around the y-axis. Because of this connection the middle arm follows the translation of the travelling plate but can still rotate. To the end of this arm the tool center point with a load is connected. This is also modelled as a rod in the vertical downward direction connected to a mass at the end.

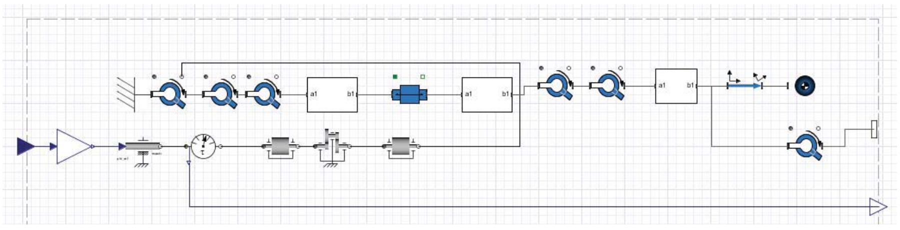


Figure 4.5 The middle rotating robotarm (2 in Figure 4.1).'

4.2 Initial values

In MapleSim the choice can be made which variables that should be used when generating and simulating the dynamic model. It can for example be joint angle positions or mass-coordinates and the initial values of these can be forced to a value, estimated or ignored. The initial values are needed for solving the differential equations in the model and in this particular model all mass-positions are ignored. Only joint angles positions are used and exactly which can be found in Appendix 14.2.

MapleSim automatically calculates these initial values for the angle positions and dynamic variables, but since the model will be exported to another program a solution how to calculate these initial values when a parameter, for example a mass, is changed for the robot has to be found.

The solution is to look into all the dynamic and kinematic equations in MapleSim and solve them with the following restrictions:

- The robot is not moving.
- The robot is in *home position*, which is the position when all actuated robot arms are at angle zero.

With these restrictions not only the angles of the upper arms are zero but also as all the angles for all joints in the fourth robot arm together with β_1 (one of the revolute joint angles between the upper arms and the forearms, defined in Figure 4.7) will be zero. The only revolute joints that don't have an angle of zero are then the other revolute joints between upper arms and forearms, defined as β_2, (Figure 4.6). After simplification in the equations by setting the correct angles to zero all of the dynamic variables except nine will be zero. These nine are three for each arm: The torque for the upper arm which is connected to the motor and forces in y and z-direction at the spherical joint. The three variables for each arm are independent of the variables for the other arms which means these equations can be solved separately with Maple's function *solve*. Unfortunately the three equations of each arm depend on β_2. However, an expression for β_2:

$$\beta_2 = \arctan\left(\frac{|y| - l_{4r}}{R_A + l_4 - R_B}\right) \tag{35}$$

can be found from Figure 4.6.

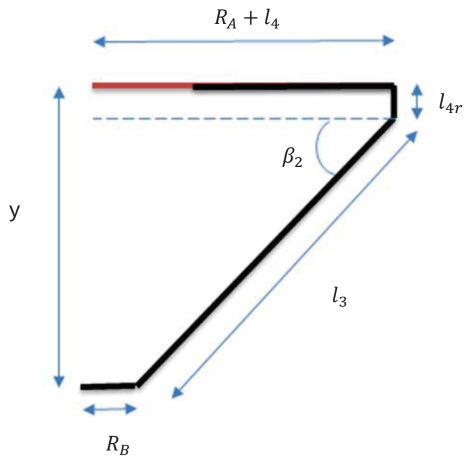


Figure 4.6 Definition of β_2 for a robot arm.

4.3 Inertia

As was previously explained the inputs in the MapleSim model are angle positions for the arms and outputs are torque needed at the motors to follow these inputs. The parameters needed to create a dynamic model are the masses, the position of the mass centre of gravity and the inertia at the mass centre for every body part around its local axis. The inertia, position of the mass centre and mass for the upper arms are given but for the forearms and the fourth robot arm they have to be measured and calculated. The inertia for the travelling plate and the springs connecting two parallel rods of the forearms are set to zero since these parts are not rotating. The travelling plate is not rotating at all and the springs are not rotating around their local axis (they follow the rotation of the forearms).

4.3.1 Forearms

The inertia for each point mass is the moment of inertia around the local axis and when recalculating the inertia at the centre of gravity for the single arms from the double parallel arms the following is made which mean a significant simplification of the model is made without any loss in the correctness of the calculations of the torques.

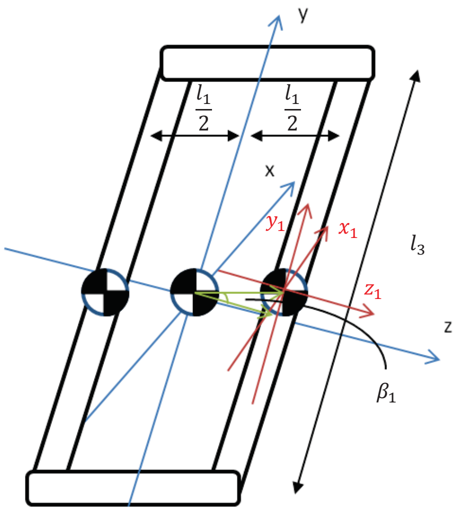


Figure 4.7 The two forearm rods and the replacement rod mass centres.

The two rods are replaced with a rod with its centre of mass directly between them along the axis parallel to the horizontal parts of the forearms (Figure 4.7). The mass for the replacement rod equals the sum of the two rod-masses and to calculate the moment of inertia at the z-axis and x-axis for the replacement rod the superposition principle and parallel axis theorem, also called Steiner's Theorem [14], for mass moment of inertia can be used:

Moments of Inertia for the double arms are calculated with the assumption that they are solid thin rods [14]:

$$I_{l3zzxx} = \frac{m_{l3} {l_3}^2}{12} \tag{36}$$

The moment of inertia around the y-axis can be neglected since the arms don't rotate around this axis.

To calculate the inertia around the x-axis at the mass centre of the replacement rod both the parallel-axis theorem and superposition principle are used. If one of the rods is considered, the moment of inertia for its local x-axis (x_1) is known, and this axis is parallel to the x-axis for the replacement rod, with the perpendicular distance to this axis $l_1/2 \cdot \cos(\beta_1)$ (Figure 4.7). The parallel-axis theorem gives that the inertia around the x-axis of the mass centre of the replacement rod for one of the two rods is

$$I_{l3zzxx} + m_{l3}\left(\frac{l_1}{2}\cos(\beta_1)\right)^2 \tag{37}$$

Since the moment of the inertia around the x-axis is independent on the x-coordinate the contribution of the moment of inertia around the x-axis for each rod can be added together to get the total by using the superposition principle.

By using the same reasoning for the z-axis, with the only difference that the perpendicular distance to the z-axis for the centre rod from the z-axis of the two other rods are $l_1/2 \cdot \sin(\beta_1)$, the inertia around the z-axis of the center rod is

$$2I_{l3zzxx} + 2m_{l3}\left(\frac{l_1}{2}\sin(\beta_1)\right)^2 \tag{38}$$

So far this single arm model is exactly as accurate as if a double arm model would be used, but a small simplification that means a little less accurate model is made by setting $\beta_1 = 0$.

With this assumption, the inertia tensor matrix, as defined in [15] for the replacement rod, is

$$I = \begin{pmatrix} 2I_{l3zzxx} + 2m_{l3}\left(\frac{l_1}{2}\right)^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2I_{l3zzxx} \end{pmatrix} \tag{39}$$

4.3.2 Telescopic arm

The fourth actuated robot arm consists of two parts, one connected to the base and one connected to the travelling plate that slides over the first. The first part has its mass centre in the centre of the rod and it is modelled as solid when estimating the inertia. The second part has its mass centre close to the bottom end and is modelled as two parts; two cylinders with an inner and outer diameter, but the smaller part at the bottom have a smaller inner diameter. Both mass centres of the parts are placed in each centre and the masses for each part are calculated using centre of mass equation, by also knowing the total weight from using a scale. This is done by balancing it and then measure the distance from the two parts mass centre to the balancing point. The inertia at this second parts mass centre is then calculated from the mass centres of the two small parts, using parallel axis theorem for the inertia around x- and z- axes. The inertia around the y-axis is neglected since this part does not rotate around this axis since it is connected to the travelling plate through a joint for which the inertia also can be set to zero.

4.4 Friction

Torque in the robot caused by friction is modelled as two parts: One constant part that only depends on the direction of rotation, Coulomb friction torque τ_c, of the motor and another part that is piecewise linear dependent on the speed (Figure 4.8). For moving in the opposite direction the friction is modelled in exactly the same way except the sign of the torque is changed and for zero velocity zero torque is chosen. The intervals are between the motor velocities $\omega_0 = 0s^{-1}, \omega_1 = 200s^{-1}, \omega_2 = 400s^{-1}, \omega_3 = 628s^{-1}$. These speeds are chosen since some torque losses at these speeds are already given in the available motor data but the assumption is made that speed dependent torque losses is more than given by these values. Therefore torque measurements and calculations are made at ω_1 and ω_2 motor speed with only an upper arm connected to the motor. At the speed ω_3, no experiments can be done without reaching the maximum allowed torque at the motor, but instead the increase in torque losses from ω_2 to ω_3 is assumed to be the same as the increase in the torque losses in the motor data. Measurements are also done to get the constant part of the friction model.

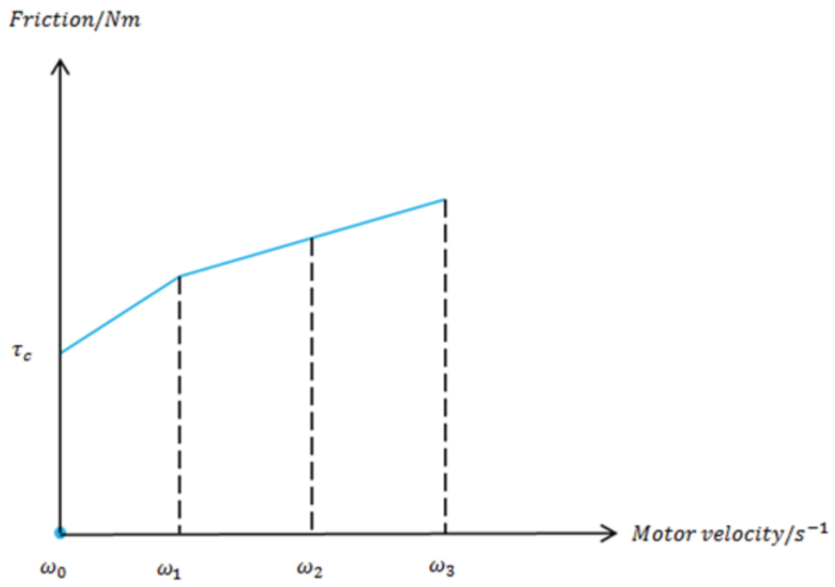


Figure 4.8 Piecewise linear speed dependent friction model.

4.5 Dynamic model export

4.5.1 Kinematic constraints

Another name for the state variables used in a multibody system are generalized coordinates [20]. As described before they can be chosen in different ways in MapleSim, by guessing or ignoring the initial conditions, but even if all are ignored MapleSim chooses the coordinates for you so that sufficient coordinates are used to describe the model.

The generated model used has 10 generalized coordinates and their corresponding 10 generalized velocities from the remaining revolute joints. These coordinates can be either independent, as in robots with open kinematic chains, or dependent as for this robot which has closed kinematic chains. They are closed since the robot arms are connected and movement in one arm affects the angles for some joints in the other arms. In other words, the angles and angle velocities for the joints chosen depend on each other and can be written as a set of equations. These equations, kinematic constraints, could be used to find the inverse kinematic of the robot in a similar way (Appendix 14.4) that have been done by P. Gossens and T. Richard [16] but the ordinary differential equations that the dynamic response consists of can only be solved together with these constraints (in contrary to open kinematic chains with independent generalized coordinates) and together they make the whole dynamic model including a set of differential algebraic equations [17]. To include the constraint equations in the dynamic model a set of reaction forces are enforcing these constraints [18]. These forces are internal forces that don’t produce any work, since no extra energy can be put into the system except from the external forces. These are the forces that in Chapter 4.2 are described as dynamic variables.

The differential algebraic equations, DAE, have to be converted into ordinary differential equations, ODE, for MapleSim to be able to solve. The goal is to use the model in B&R Automation studio which is possible by exporting it to ANSI-C code with MapleSim’s function B&R- connector. A number of different choices of ODE solvers can be chosen to be used in the C-code, where the simplest one is used, which is an Euler solver. During the automatic code-generation simplification on the equations and optimization on the code is performed which results in very fast real-time ANSI-C code that can be used in the application on B&R Automation studio.

4.5.2 Stabilizing methods

The way to convert the DAEs to ODEs is to replace the position constraints with acceleration constraints which are integrated simultaneously as the dynamic ODEs. However during numerical integration in the C-code the numerical errors can accumulate which leads to violation of the constraint equations and which further can lead to the error increasing at an exponential rate [19]. Solutions to this problem are Baumgarte Constraint Stabilization [20] or constraint projection of the solution on the position constraint. The method of constraint projection is used. This method projects the solutions each integration step from the ODEs on to the position constraint using the constraint projection routine as described on MathWorks website [21]. When having the constraint projection active on a model the integration time increases, but the model will be stable. For this reason two dynamic models are created from MapleSim; one model with no stabilization that is used short times for fast movements of the robot and one with constraint projection for slower movements.

4.5.3 B&R connector generated programs

The fact that MapleSim has the function that enables you to add your own equations which then can be optimized, simplified and generated into ANSI C-code is taken advantage of to make two other programs for B&R Automation studio. In total the four programs are:

- *Dynamic Model 1*. Joint angle, angle velocity and angle acceleration for the four robot arms connected to the motors via the gears is used as input and torque for the motors as output.
- *Dynamic Model 2*. The same as *Dynamic Model 1* except that this has projection on the constraint equations included in the model.
- *Jacobian*. The inverse kinematics for velocity and acceleration, in other words the equations that calculates the speed and acceleration for the robot arms given the the angle position of the arms, travelling plate position, speed and acceleration. The output of this program is the source of input for speed and acceleration for the two dynamic models.
- *Initial*. The program to calculate the initial state variables and dynamic variables in the Dynamic models.

5 Path and trajectory generation

To be able to move the robot it needs to have a path to follow and for this purpose two different kinds of paths are created: One to change the position from a stationary point to another stationary and the other is to change the velocity from a constant velocity to another constant velocity. But not only is the position from the path generation needed but also velocity and acceleration. For this purpose a trajectory calculation is made which gives position, velocity and acceleration. The trajectory is created with polynomial functions based on methods described in [15] with the difference that the end time of the movement is unknown and is calculated by setting a value for the maximum acceleration that should be reached during the movement (Chapter 5.1.1). The trajectories are implemented in a computer program with a specific time step, τ (Chapter 8.2).

5.1 Change position

The path is created using a 5th order polynomial function for the position of the travelling plate X, previously denoted B. The reason is the movement have three initial conditions and three end conditions that have to be fulfilled, from start time, t_0, to end time, t_f:

$$X(t_0) = 0$$

$$\dot{X}(t_0) = 0$$

$$\ddot{X}(t_0) = 0$$

$$X\left(t_f\right) = d$$

$$\dot{X}\left(t_f\right) = 0$$

$$\ddot{X}\left(t_f\right) = 0$$

where $d \geq 0$ is the desired distance for the movement. This will mean there are six equations with six unknown constants$(a_{0,}a_1, a_2, a_3, a_4, a_5)$, which are solvable as long as $t_f > 0$ [15].

$$X(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{40}$$

$$\dot{X}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{41}$$

$$\ddot{X}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \tag{42}$$

$$t = t_0, t_f$$

Without loss of generality we can set start time $t_0 = 0$. This gives the solution for the constants:

$$a_0 = 0$$

$$a_1 = 0$$

$$a_2 = 0$$

$$a_3 = \frac{10d}{{t_f}^3}$$

$$a_4 = -\frac{15d}{{t_f}^4}$$

$$a_5 = \frac{6d}{{t_f}^5}$$

5.1.1 End time (time-optimal)

The start time is known but the end time is not. One choice is to make the move to the end position as fast as possible where the limiter for the movement will be the maximum acceleration. To find the maximum acceleration, Acc, (42) is differentiated with respect to time and the calculated constants are inserted in this expression and set to zero:

$$\frac{60d}{{t_f}^3} - \frac{360d}{{t_f}^4}t + \frac{360d}{{t_f}^5}t^2 = 0 \tag{43}$$

Solving this for *t* gives the times with maximum and minimum acceleration, these times inserted in acceleration equation (42) gives the maximum acceleration and the end time:

$$Acc = \frac{10d\sqrt{3}}{3{t_f}^2} \Longrightarrow t_f = \sqrt{\frac{10d\sqrt{3}}{3Acc}}$$

5.2 Change velocity

The velocity change ignores the position and thereby the velocity polynomial, $V(t)$, only have four conditions:

$$V(t_0) = v_s$$

$$\dot{V}(t_0) = 0$$

$$V(t_f) = v_f$$

$$\dot{V}(t_f) = 0$$

where v_s is the current velocity and v_f is the desired end velocity and $t_0 = 0$. These four conditions mean that a 3rd degree polynomial is enough:

$$V(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{44}$$

The solution will give

$$a_0 = v_s$$

$$a_1 = 0$$

$$a_2 = \frac{3(v_f - v_s)}{{t_f}^2}$$

$$a_3 = -\frac{2(v_f - v_s)}{{t_f}^3}$$

And as for the change position the end time have to be calculated from the max acceleration with the result

$$t_f = \frac{3|v_f - v_s|}{2Acc}$$

The paths are created discretely and the change in position each time-step can be approximated by a numerical integration of the velocity:

$$\mathrm{X(t)} \approx \Delta\mathrm{X(t)} = \mathrm{V(t)} \cdot \tau \tag{45}$$

5.3 Implementation

The paths are implemented in the program to create different reference paths for the travelling plate of the robot. One path is created for the robot to move from the current position, position A, to a desired position, position B, in a straight line. The distance between these points in space is calculated and saved as the value *d*. The fifth order polynomial function can then be used but since this function is only in one dimension it is multiplied with the direction vector. The direction vector is a unit vector for the direction between position A and B. Of course this calculated position is a position change of the travelling plate and is added to the current position. The second velocity polynomial function is used for two movements: move the robot at a constant speed along a line in space, or at a constant angular velocity in a circle. The polynomial is used only when accelerating up to a constant speed or angular velocity, or change the speed or velocity. For the first application the direction vector and desired velocity is set by the user and the path is created and multiplied with this direction vector. Of course here is also the current position of the manipulator added to the created path for the position. In the second application the change in velocity from the velocity polynomial is added as a change in the current angle velocity of the robot. The angle is measured from the z-axis and the circle is created on the zx-plane. These polar coordinates need to be transformed back to Cartesian coordinates for the inverse kinematics and Jacobian.

6 The controller

6.1 Control principle for one motor

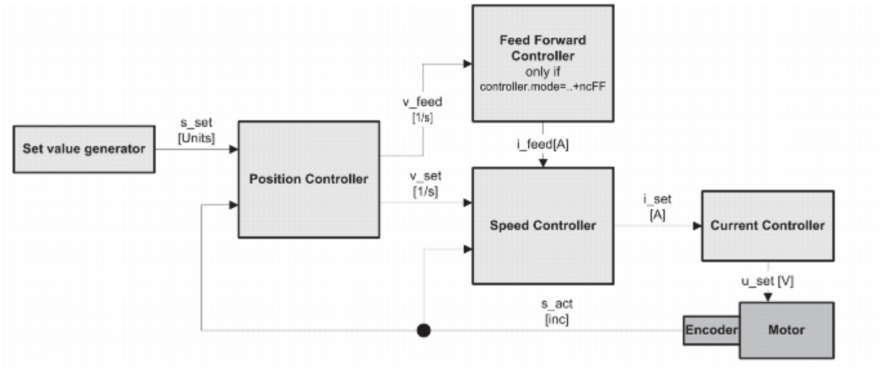


Figure 6.1 Block diagram of the control-loops for control of a motor [22].

The principle of the B&R ACOPOS controller is shown in Figure 6.1. The controller is a cascade controller, were the *s_set* comes from a set value generator from the drive. The only sensor information from the motor is the position. This position is fed as input to the position controller, where it is subtracted with the *s_set* to get the position error in the position controller (Figure 6.2). The *s_act* is also differentiated and subtracted with the output of the position controller, *v_set*, to get the velocity error which is used in the speed controller (Figure 6.3). The output of the speed controller is fed to the current controller which sets the appropriate voltage on the motor. Both the position controller and the speed controller are simple PI-controllers with parameters that can be changed in Automation studio but the parameters of the Current controller cannot be set but is instead calculated from given motor parameters [22].

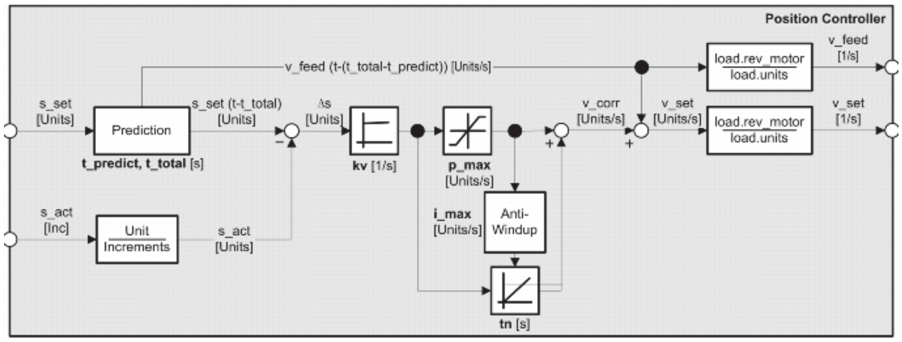


Figure 6.2 The position controller in Figure 6.1 [22].

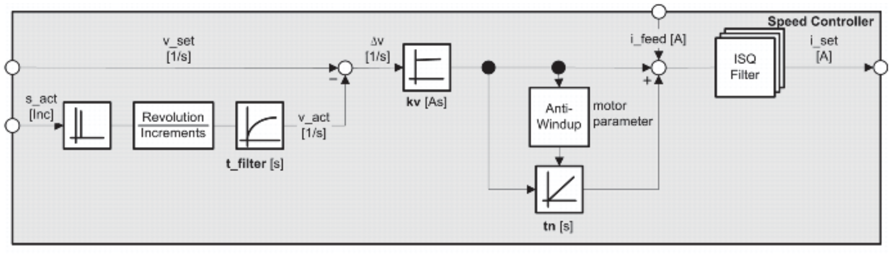


Figure 6.3 The speed controller in Figure 6.1 [22].

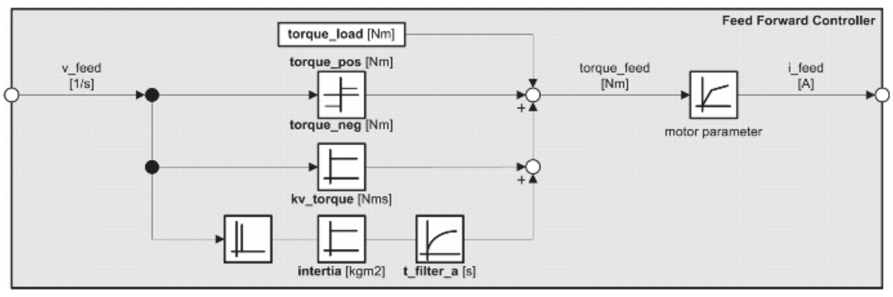


Figure 6.4 The feed forward controller in Figure 6.1 [22].

The feed forward controller (Figure 6.4) can be used to improve the performance if one knows five parameters. Two of the parameters are the load torque (torque from the weight of a mass connected to the motor) (*torque_load*) and the speed dependent torque losses (*kv_torque*). The constant torque, Coulomb torque, when moving in positive (*torque_pos*) or negative (*torque_neg*) direction respectively are two other parameters and the inertia of the motor is the last one. It works in the way that through *v_feed*, which is *s_set* differentiated and then multiplied with the *kv_torque* to get the speed dependent torque losses, differentiated again and multiplied with the inertia to get the acceleration torque and depending on the sign of *v_feed*, *torque_pos* or *torque_neg* will be chosen as the constant torque. These three torque values are added together with the load torque to get the *torque_feed*. If all the parameters are correct this value should be the same as the torque generated in the motor. To send this value as feed forward to the motor it has first to be converted to current which is done by the torque constant given in the motor parameters. The resulting *i_feed* is added to the output of the speed controller and sent to the current controller which sets the appropriate voltage to the motor.

6.2 Control principle for all motors

In the dynamic model of the robot there are three types of torques that contribute to the total torque: Gravitational torque, inertial torque and friction torque. These three are modelled and added together to the variable *torque_load* for each axis when moving the robot. Because they are added together the other parameters in Figure 6.4 are then set to zero. The set position value and the value of *torque_load* for each motor are created according to the principle in Figure 6.5: The desired position, velocity and acceleration of the travelling plate of the robot given in Cartesian coordinates are transformed into angle position, *s_set*, velocity and acceleration of each motor through the inverse kinematics for position and for velocity and acceleration. The angle position, velocity and acceleration are inputs to the dynamic model currently used and the outputs are the torques for every axis, *torque_load*, which together with angle position, *s_set*, are sent to the drives as described earlier. The actual position of each axis is measured and with the forward kinematics the actual position coordinates for the travelling plate is acquired.

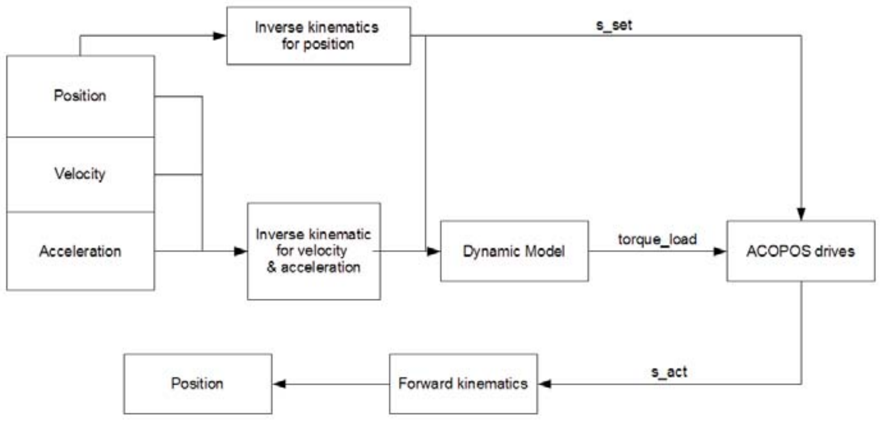


Figure 6.5 Block diagram of the principle of controlling all robot motors

7 Program Design

7.1 Move to position

The flowchart in Figure 7.1 describes the steps and the states for the program when moving a robot to a certain position and the boxes with the text “Control with *Dynamic Model 1*” or “Control with *Dynamic Model 2*”, is the control described in Figure 6.5.

STATE_WAIT – The default state when the robot is not moving. Here the robot can be in either *home position*, which is the position when the robots arms are at a horizontal angle and no dynamic model is active, or not. In *home position* the torque and other variables are initialized at the first movement of the robot, but also when returning to *home position*. When the robot is at rest in any other position than *home position*, the *Dynamic Model 2* is active.

STATE_CHECK_WORKSPACE – When the user sends a moving command, either to a specific position or to the home position, the program changes its state to this. Here every point along the path is checked so it is inside the defined workspace of the robot. If one point is outside, the program stops and waits for user to make a reset command and try another path, which means it goes back to the STATE_WAIT again. If all are inside the workspace the *Dynamic Model 2* is cancelled if it was active and Dynamic model 1 is initialized and started and the program changes to STATE_MOVE.

STATE_MOVING Here the path is created and the robot is simultaneously controlled with *Dynamic Model 1* (Figure 6.5). When the specified end-time of the movement is reached the *Dynamic Model 1* is cancelled and if the robot moved to somewhere else than to the home position, the *Dynamic Model 2* is initialized and activated.

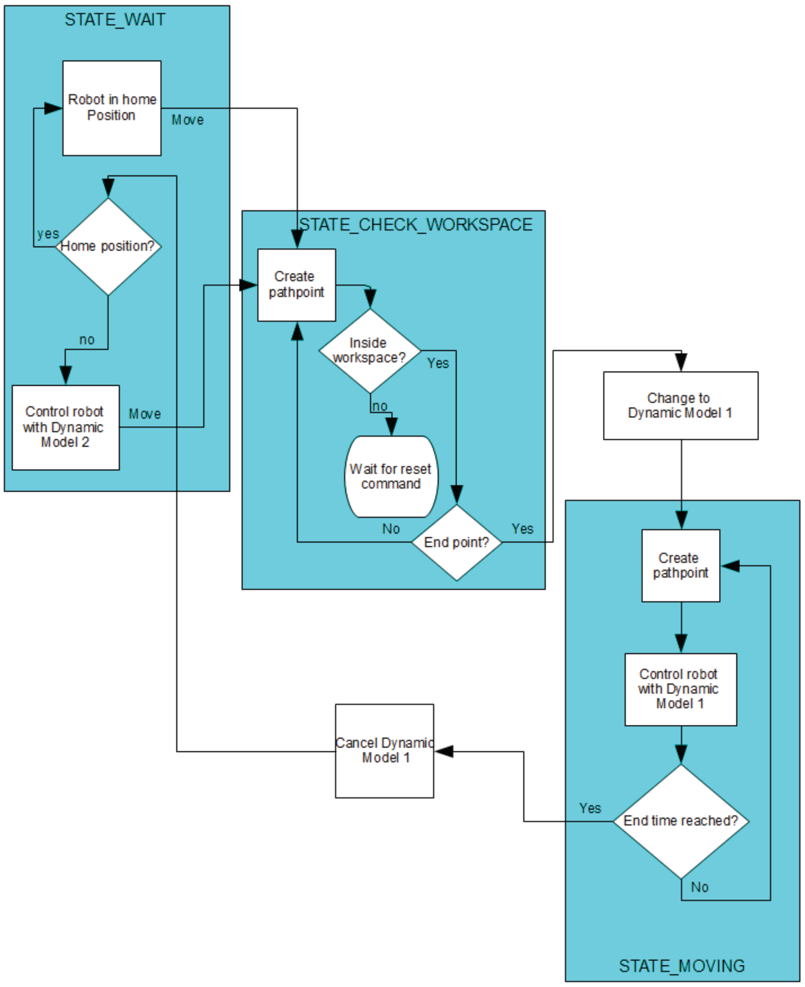


Figure 7.1 Flow-chart over the states of the program when moving to a position.

7.2 Move at constant speed

Moving the travelling plate at a constant speed, also called jogging, is done either along a line or in a circle. The program also has three states, and depending if the desired jog-speed of the robot is bigger or smaller than the user-specified parameter *switchspeed*, *Dynamic Model 2* or *Dynamic Model 1* is used:

STATE_WAIT – The same as moving to a position; either *Dynamic Model 2* or no dynamic model is active. When a command to move to a desired jog-speed is executed a check is made if the speed is higher than *switchspeed*, and if it is, then *Dynamic Model 1* is activated. Otherwise Dynamic model 2 is activated if it wasn't already active.

STATE_REF_CHANGE – This state is used whenever the user starts a jog-movement from STATE_WAIT or changing the desired speed from STATE_JOGGING. The needed parameters to create the path are calculated here by using current jog-speed, desired jog speed and maximum acceleration. The exit out of this state is always to STATE_JOGGING.

STATE_JOGGING – In this state the robot accelerates up to or moves at a constant speed until the user sends a stop command. Or in the case when moving in a circle it sends a stop command after it moved a whole circle. Changing the desired speed will also make the program leave this state. For these three alternatives the new program state will become STATE_REF_CHANGE. A check is also made in this state so that the state changes to STATE_WAIT when the robot has stopped.

8 Automation Studio

8.1 Communication

The connection between the PC, with Automation Studio, and the PLC is done with an Ethernet-cable. The IP-address for the computer is set to 192.168.0.2 and for the PLC 192.168.0.1. To connect to the interface (Chapter 8.3) on the PLC, which is used to control the program, a VNC-viewer is used with the password "c" for control.

The communication to the PLC and the NC manager (Chapter 2.3.1) from the computer can be made in different ways:

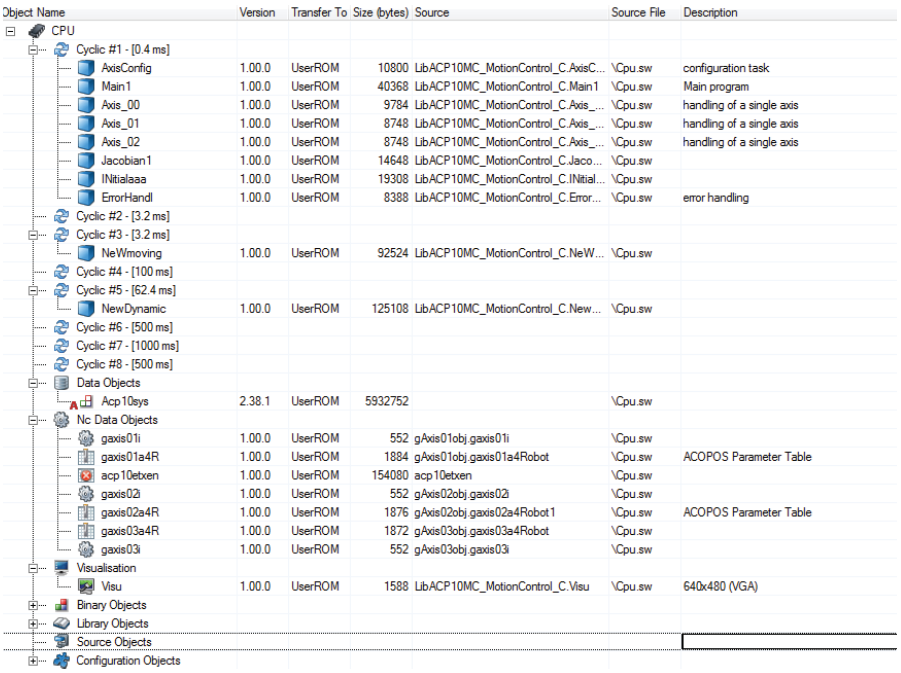
- Direct control of an axis in automation studio with ACP10 software, a B&R specific interface, also called NC Test.
- Through PLCopen motion control function blocks.

There are many function block used in the program, some are PLC Open standard functions but there are also some B&R specific functions which are operated in the same ways as the standard blocks [3]. Some of the blocks used in the application are so called cyclic function blocks. These blocks are used every cycle of the task class the program is in (see Chapter 8.2) and they are:

- MC_BR_ReadCyclicPosition, reads the actual position of the motor axis.
- MC_ReadActualTorque, reads the actual torque at the axis.

- MC_BR_CyclicWrite, can be used to write to any variable on the drive, but is used to write the calculated torque from the models to the variable *torque_load* for the feed forward control.
- MC_BR_MoveCyclicPosition, writes the reference value for position, *s_set*, to the axis.

8.2 Threads



Object Name	Version	Transfer To	Size (bytes)	Source	Source File	Description
CPU						
Cyclic #1 - [0.4 ms]						
AxisConfig	1.00.0	UserROM	10800	LibACP10MC_MotionControl_C.AxisC...	\Cpu.sw	configuration task
Main1	1.00.0	UserROM	40368	LibACP10MC_MotionControl_C.Main1	\Cpu.sw	Main program
Axis_00	1.00.0	UserROM	9784	LibACP10MC_MotionControl_C.Axis_...	\Cpu.sw	handling of a single axis
Axis_01	1.00.0	UserROM	8748	LibACP10MC_MotionControl_C.Axis_...	\Cpu.sw	handling of a single axis
Axis_02	1.00.0	UserROM	8748	LibACP10MC_MotionControl_C.Axis_...	\Cpu.sw	handling of a single axis
Jacobian1	1.00.0	UserROM	14648	LibACP10MC_MotionControl_C.Jaco...	\Cpu.sw	
INitialaaa	1.00.0	UserROM	19308	LibACP10MC_MotionControl_C.INitial...	\Cpu.sw	
ErrorHandl	1.00.0	UserROM	8388	LibACP10MC_MotionControl_C.Error...	\Cpu.sw	error handling
Cyclic #2 - [3.2 ms]						
Cyclic #3 - [3.2 ms]						
NeWmoving	1.00.0	UserROM	92524	LibACP10MC_MotionControl_C.NeW...	\Cpu.sw	
Cyclic #4 - [100 ms]						
Cyclic #5 - [62.4 ms]						
NewDynamic	1.00.0	UserROM	125108	LibACP10MC_MotionControl_C.New...	\Cpu.sw	
Cyclic #6 - [500 ms]						
Cyclic #7 - [1000 ms]						
Cyclic #8 - [500 ms]						
Data Objects						
Acp10sys	2.38.1	UserROM	5932752		\Cpu.sw	
Nc Data Objects						
gaxis01i	1.00.0	UserROM	552	gAxis01obj.gaxis01i	\Cpu.sw	
gaxis01a4R	1.00.0	UserROM	1884	gAxis01obj.gaxis01a4Robot	\Cpu.sw	ACOPOS Parameter Table
acp10etxen	1.00.0	UserROM	154080	acp10etxen	\Cpu.sw	
gaxis02i	1.00.0	UserROM	552	gAxis02obj.gaxis02i	\Cpu.sw	
gaxis02a4R	1.00.0	UserROM	1876	gAxis02obj.gaxis02a4Robot1	\Cpu.sw	ACOPOS Parameter Table
gaxis03a4R	1.00.0	UserROM	1872	gAxis03obj.gaxis03a4Robot	\Cpu.sw	
gaxis03i	1.00.0	UserROM	552	gAxis03obj.gaxis03i	\Cpu.sw	
Visualisation						
Visu	1.00.0	UserROM	1588	LibACP10MC_MotionControl_C.Visu	\Cpu.sw	640x480 (VGA)
Binary Objects						
Library Objects						
Source Objects						
Configuration Objects						

Figure 8.1 Layout on the PC on how the programs are placed in the task classes in the PLC.

On the CPU it runs several task classes, threads, with different cycle times and different priorities. Every program has an initial part and a cyclic part where the initial part is only started once when turning on the hardware.

Starting from the top the first one is *Cyclic #1*.This is the task class with the shortest cycle time, 0.4 *ms*, and the highest priority. The programs in this task class are:

- Main1, this is the main program. It has a responsibility to change the state when moving to a position and jogging, creating the path, checking workspace and calculating the forward and inverse kinematic.
- AxisConfig, this only has an initial part where each axis in the program with its corresponding parameters is set to the correct Acopos-unit which is connected to the correct motor.

- *Axis_00*, *Axis_01*, *Axis_02*, in these programs all the communication to the drives are made with function blocks. The structure is based on the program from [23].
- *Jacobian1*. The program from MapleSim to calculate angular velocity and acceleration for the axes.
- *Initialaaa*, Initialization program from MapleSim and it is used when the robot is in the home position, either by the user or called from the main program.
- *ErrorHandling*, from [23], checks each all axis for errors and stops the axis in case of error.

All programs in this cycle are executed once every cycle and in order from top to bottom. This means that when *main* creates the reference values for the axis these are sent from the programs axis programs at almost exactly the same time and the movement of the axes are thereby synchronized.

In the two task classes *Cyclic #3* and *Cyclic #5* the program for *Dynamic Model 1*, called *NeWmoving*, and the program for *Dynamic Model 2*, called *NewDynamic*, are found. In these dynamic models the joint angles and all the dynamic variables are used as outputs, in addition to the torques, and saved in global variables. The reason that all variables are used as outputs is to be able to initiate a model when changing between models using these variables. The cycle time of these task classes are chosen as small as possible but still large enough so that all calculations in the models are done within this time.

8.3 The interface

A graphical interface (GUI) has been developed for operator communication, see Figure 8.2. Navigation between the different pages is made at the bottom at the GUI. The pages are:

- *Home*, this page is for controlling the robot, and looks like Figure 8.2. To the top left the desired position and angle for the robot can be entered and the move started. In the middle are buttons to activate the use of jogging the robot or deactivate it. Under these buttons the user can change the desired acceleration for the movement or the turning of the 4th axis, but in this thesis this axis is not connected to a motor. The direction, speed, acceleration and *switchspeed* for jogging the robot can be changed to the right with a start and stop button. The buttons *circle* and *line* is for the user to decide if the robot should move in a circle or in a line. For the circle movement the speed and acceleration is angle velocity and angle acceleration in the polar coordinates but for moving along a line the speed and acceleration is the velocity and acceleration in the Cartesian coordinate system of the travelling plate. Pressing the *homing* button moves the robot to the *homing* position. *Reset* is pressed to acknowledge the error message which appears the desired path is outside the workspace for the robot. This message will appear where now “B&R Template Instant Message” is written.
- *Geometric parameters*, edit the values of the parameters of lengths, gear ratios and the working area for the robot which is used in the dynamic model.
- *Masses*, edit the values of the parameters for the different masses of the robot used in the dynamic model.
- *Inertias*, edit the values of the parameters of the inertia of the different parts of the robot used in the dynamic model.

- *Start motors*, is the page where the motors are started and stopped. They should always be started in the order 1-3* and should only be started in home position. There are also two buttons to turn on and off the breaks.

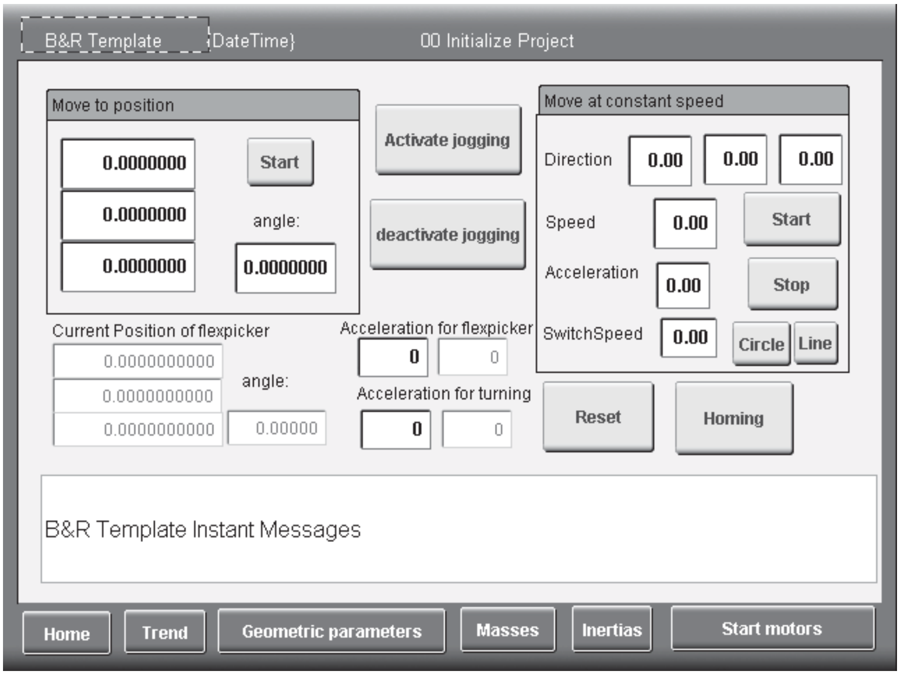


Figure 8.2 Interface to change parameters and move the robot.

***Even though the FlexPicker have breaks on the first three axes, a signal from one drive to turn off the break turns off all of them. Therefore only the first axis has break parameters, which makes it very important, to avoid moving an axis with the break on, to start the first axis first which then turns off the break for all three and then turn on the other two axes. Also important to note is that the fourth arm is not connected to any drive in the current experimental setup.

9 Experiments

These experiments, except the varying acceleration experiment (Chapter 9.4.2) and experiments with position control for the whole robot (Chapter 9.6), were done on one arm of the robot using the NC Test interface in Automation Studio. The generated path is then created with the NC Test and the feed forward parameters are used instead of the dynamic models (a more detailed description in Chapters 9.1-9.5). The experiments in Chapter 9.4.2 were done with the programmed software and applied to one arm of the robot, and in Chapter 9.6 the whole robot with the programmed software as described in the thesis was used (with the modification that the feed forward parameters were used instead of the dynamic models in Chapter 9.6.2) .

9.1 Torque constant verification

To verify that the torque-constant in the motor data from ABB is correct when using it in B&R drive units, an experiment is made. All parts except for the upper arms are detached and a weight is attached to the end of the arm. The theoretical torque needed to lift the arm with a 1 kg weight is calculated and added to the parameter *torque_load*. The control parameters in the position and speed controller are set to zero so that the motor is driven only by the current controller with a reference torque value as input (Figure 6.1). A test is made to see how much weight is needed to keep the arm motionless in a horizontal position with the same force needed when by hand moving the arm up as moving the arm down. The extra torque needed to move the arm is due to high static friction. If something else than the corresponding force to counterweight the 1 kg load is needed for this a scaling of the value from the motor data is needed.

9.2 Coulomb friction

To estimate this value the robot-arm was moved at a constant slow speed, first in the direction of lifting the robot arm and then moving it in the direction down. Different values of the direction dependent torque feed forward parameters were tried to get the position error to vary around zero*. The resulting torque value when lifting the arm (negative direction) should theoretically be

$$\tau_{up} = -\tau_g - \tau_c \tag{46}$$

where τ_g is the torque needed to overcome the gravitational pull from the arm and τ_c is the Coulomb friction. For moving the arm down the torque needed is

$$\tau_{down} = -\tau_g + \tau_c \tag{47}$$

The difference of (46) and (47) will then be

$$\tau_{down} - \tau_{up} = 2\tau_c \Leftrightarrow \tau_c = \frac{\tau_{down} - \tau_{up}}{2} \tag{48}$$

Since the gravitational component of the torque varies with the position of the arm both experiments were performed around angle zero.

*Since the value of the actual torque was varying too much to get an accurate value by just reading it.

9.3 Speed dependent friction torque

Measurements of the torque at two speeds, $\omega_1 = 200s^{-1}$ and $\omega_2 = 400s^{-1}$, were performed in a similar way as for Coulomb friction; by trying with different values for the torque feed forward parameter but only in negative direction. The value of the torque which gives lag error around zero when the arm is moving at constant speed around angle zero is then theoretically:

$$\tau_{up} = -\tau_g - \tau_c - \tau_{spi} \Leftrightarrow \tau_{spi} = -\tau_{up} - \tau_g - \tau_c \tag{49}$$

where τ_{spi} is the extra torque needed to compensate for the speed dependent friction at speed $\omega_i, i = 1{,}2$. τ_g is calculated theoretically from the equations in MapleSim. By measuring at these two speeds the torque can be compared to the only given data for friction which is the torque losses from the motor data. As talked about in Chapter 4.4 the torque at the third speed can’t be measured without reaching the maximum torque in the motor. This doesn’t mean that the arm can’t move at this speed but it can’t move at this speed for long enough for measurements of the torque to be made without the arm reaching its limit angles.

9.4 Acceleration experiments.

9.4.1 Constant acceleration

When the values of the friction torque are determined some tests with constant acceleration are performed and the actual torque is compared to the theoretical. These tests were made accelerating up to the two speeds, ω_1 and ω_2 for two different values of constant acceleration:

Acc1=6283 *rad/s*$^{-2}$

Acc2=12566 *rad/s*$^{-2}$

The following simplifications are made when comparing the actual torque and the theoretical:

- The speed dependent friction torque at the constant acceleration is the same as the speed dependent friction torque at the maximum speed of the movement, which means the result from the previous experiment is used.
- The gravitation torque component is the theoretically calculated at position zero of the arm.

Using the value from the Coulomb torque experiment the theoretical torque needed at constant acceleration in the negative direction is then:

$$\tau_{acc} = -\tau_g - \tau_c - \tau_{sp} - \left(I_{motor} + \frac{I_{l4xx} + {l_{4c}}^2\, m_U}{{r_G}^2}\right) a \qquad (50)$$

where a is the absolute value of the acceleration and the quotient in the parenthesis is the equivalent inertia for the upper arm at the motor.

9.4.2 Varying acceleration with generated path

Compared to the other experiments this was not performed in NC Test in Automation Studio but instead performed using a modified version of the jogging of the robot. The modification that was made was that it sends the reference position directly as angle position to one axis and to the other axis to keep zero as reference. Another modification is that the model used is a model from MapleSim modelling just one axis. The program for the model is placed in the same thread, *Cyclic #1*. Also the Jacobian is deactivated in the program since the angle velocity and angle acceleration together with the angle reference is sent to the model. There are two movements performed in this test: One movement when it starts with the motor at position zero and accelerates with maximum acceleration of 23100*rad/s*$^{-2}$ up to speed 200*rad/s*$^{-1}$ and when the motor passes 13.2 radians a stop movement is initiated. In this movement the friction model is not active, but for the next movement it is and the inertia of the motor is increased in the model to get a better result, to 0.00019*kgm²*. The motor is turned from 0 to -13.2 motor radians at maximum acceleration of *19800rad/s*$^{-2}$ and speed 132*rad/s*$^{-1}$. Both of the experiments are done with feed forward turned on with the parameters used from the tuning (see Chapter 9.5).

9.5 Tuning

Tuning of the cascade controllers is performed by tuning the speed controller while having the position controller inactive with the method from [24]. The parameter *kv* for the speed controller is set as large as possible and *tn* as small as possible without too much oscillation. Since *tn* is used in the speed controller another integrator is not needed in the position controller. When tuning the position controller a step response is needed but the maximum torque sets a limit for this so instead an extremely small change in reference position without maximizing the torque is set. This reference change is a step response of 0.0009 radians of a motor revolution.

The feed forward parameters *torque_pos* and *torque_neg* are set with the values from the results of the friction experiments. Ideally the three parameters *torque_load*, *kv_torque* and *inertia* should not be set here since they are included in the dynamic models, but to get the best performance when not using the dynamic model they are set. The load torque for the motor is something that actually changes depending on the angle of the robot arm, but a simplification is made which sets the theoretically calculated torque at angle zero for the arm as this parameter. The value used for *kv_torque* is the resulting proportional value used in the first interval in the friction model originating from the results of the friction experiments. Finally the value for the inertia is the inertia for the motor, not from the given parameters but the value from the result from the acceleration experiments that better fit the measured torque, plus the inertia for the arm recalculated to its equivalence at the motor. This inertia has the lowest lag-error when testing the performance on different movements.

9.6 Experiments with position control for the whole robot

In these movements, with all of the arms of the robot connected, the actual position of the travelling plate is compared with the set position of the travelling plate by calculating the distance between the points. Since the actual position for the travelling plate in this thesis is based on calculations from the actual angles of the motors the actual position might differ a bit in reality and need to be measured with some sensors on the robot to give a more accurate value. The delay between sending a reference position and reading a reference position is taken into account by saving the values of the reference position in a buffer. The position of the robot when in *home position* is $(x, y, z) \approx (0.0m, -0.655m, 0.0m)$.

9.6.1 With dynamic model

A movement between two points is made with the dynamic model active. The movement is done from *home position* to $(x, y, z) = (-0.1m, -0.75m, 0.3m)$ with acceleration of 80 *m/s²*. The inertia of the motor used in the model is the same as used in the second movement as described in Chapter 9.4.2.

9.6.2 Without dynamic model

Two movements are made: One movement between two positions in space and one in a circle are done with the robot with the program described in the thesis except that no dynamic model is activated. The movement between positions is done between *home* position and $(x, y, z) = (0.3m, -0.8m, 0.0m)$ with acceleration of 160 *m/s²*. The movement in a circle is done to and from $(x, y, z) \approx (0.0m, -0.7m, 0.2m)$ with angle velocity 16*s*$^{-1}$ and angle acceleration 800*s*$^{-1}$. The feed forward is used from the tuning with an increase of the value *torque_load* and *inertia* since these values are expected to be higher when connecting the full robot compared to moving only single arms.

9.6.3 Comparing with and without dynamic model

The same movement is done as in chapter 9.6.1but with an acceleration of 100 *m/s²*. It is done twice; first with the dynamic model active as in that chapter and then with the feed forward as done in Chapter 9.6.2.

10 Results

In this section the results from the experiments in Chapter 9 are summarized. The plots shown have the unit of the torques for the motors in *Nm*, distances in meters and all angles and angle velocities in the figures in *Units* where one unit is defined in B&R automation studio as a thousand of a revolution of the motor.

10.1 Torque constant

To keep the arm horizontal and have it equally resistant in both directions to external forces a multiplication of the torque constant from the motor data by three is needed.

10.2 Coulomb friction

$$\tau_c \approx 0.170Nm$$

10.3 Speed dependent torque losses

The total torque at ω_1, is larger than in theory with the torque losses from the motor data and Coulomb friction from the experiment included. The increase in torque from ω_1 to ω_2 is about the same as the increase according to the motor data.

$$\tau_{sp1} \approx 0.155Nm$$

$$\tau_{sp2} \approx 0.265Nm$$

10.4 Acceleration

10.4.1 First acceleration experiment

$$\tau_{acc1,theory} \approx -1.15Nm$$

$$\tau_{acc2,theory} \approx -2.07Nm$$

From Figure 10.1:

$$\tau_{acc1,exp} \approx -1.80Nm$$

From Figure 10.2:

$$\tau_{acc2,exp} \approx -3.20Nm$$

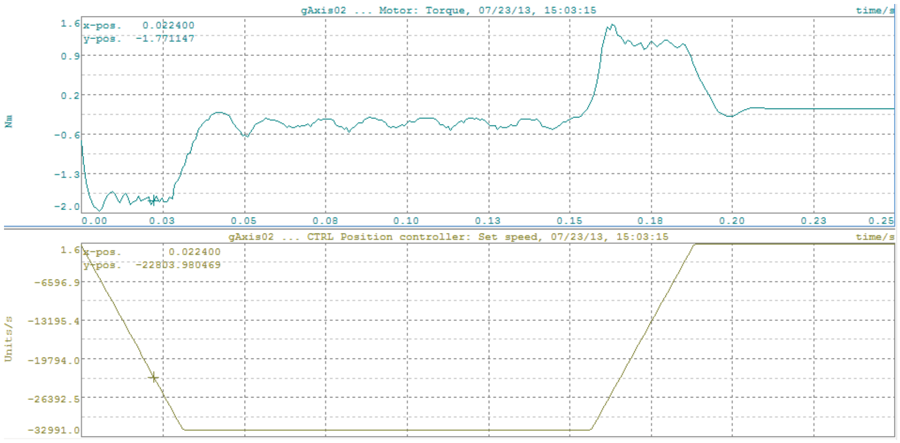


Figure 10.1 Actual torque and set speed for an axis moving with constant acceleration 6283s^{-2} up to speed $\boldsymbol{\omega_1}$.

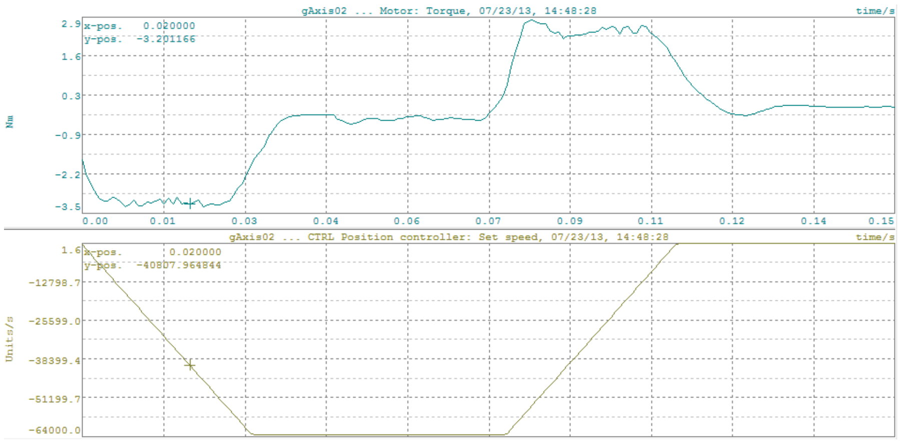


Figure 10.2 Actual torque and set speed for an axis moving with constant acceleration 12566s^{-2} up to speed ω_2.

10.4.2 Varying acceleration experiments

Results from the first varying acceleration experiment are shown in Figure 10.3 and the second in Figure 10.4. The red line is the model and the green the actual torque.

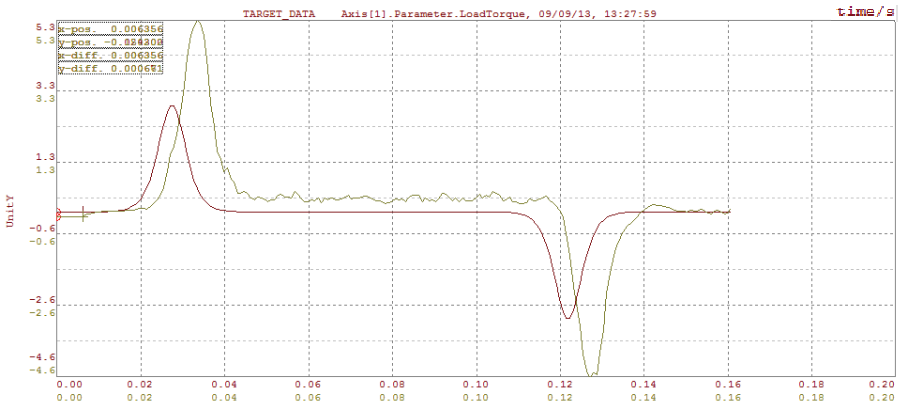


Figure 10.3 Calculated and actual torque in *Nm* for the first varying acceleration experiment

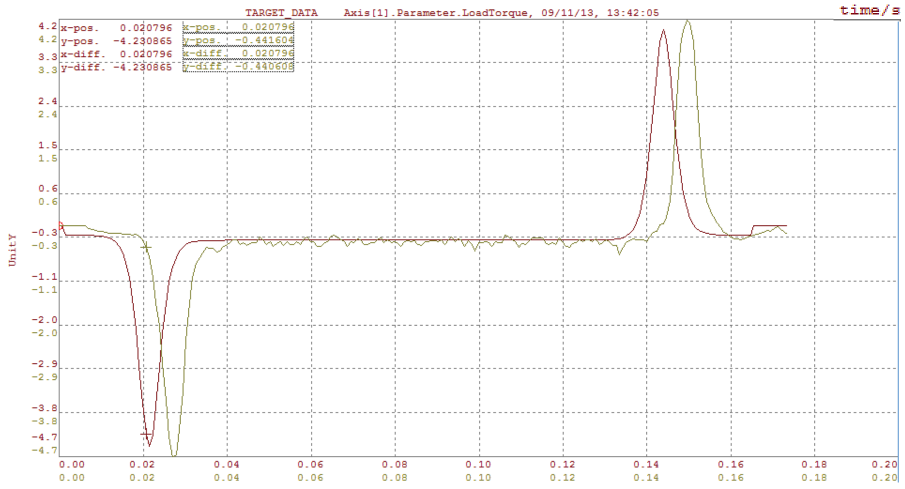


Figure 10.4 Calculated and actual torque in *Nm* for the second varying acceleration experiment

10.5 Tuning

A step response with the control parameters from the tuning, but without feed forward is shown in Figure 10.5 and lag-error together with actual torque for the second acceleration experiment is shown in Figure 10.6. In this experiment the feed forward parameters from the tuning is used.

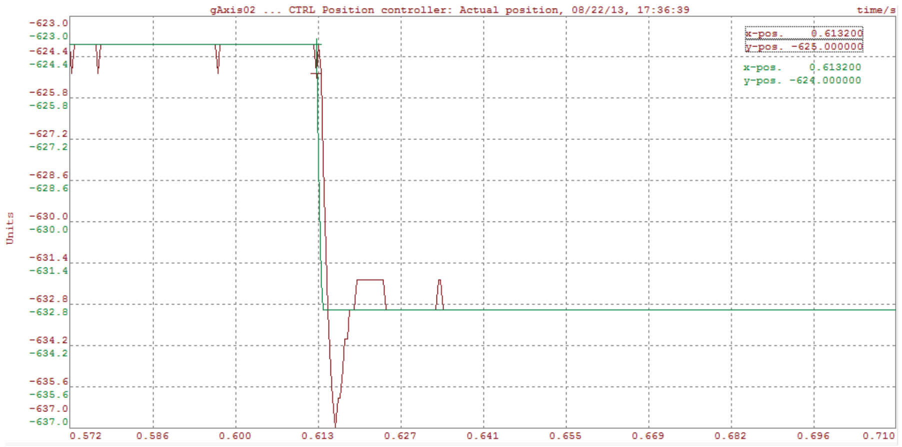


Figure 10.5 0.009 radians step response. Green is set position and red actual position.

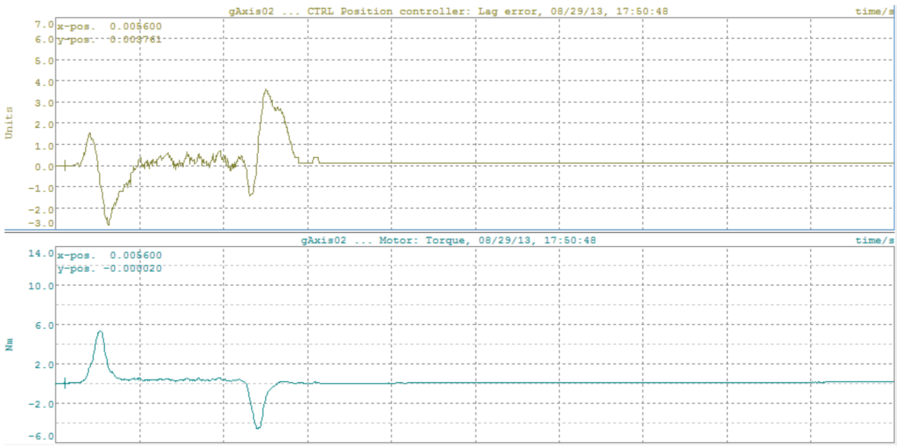


Figure 10.6 Lag error and the actual torque for the second varying acceleration experiment

10.6 Moving the robot and jogging the robot

10.6.1 With dynamic model

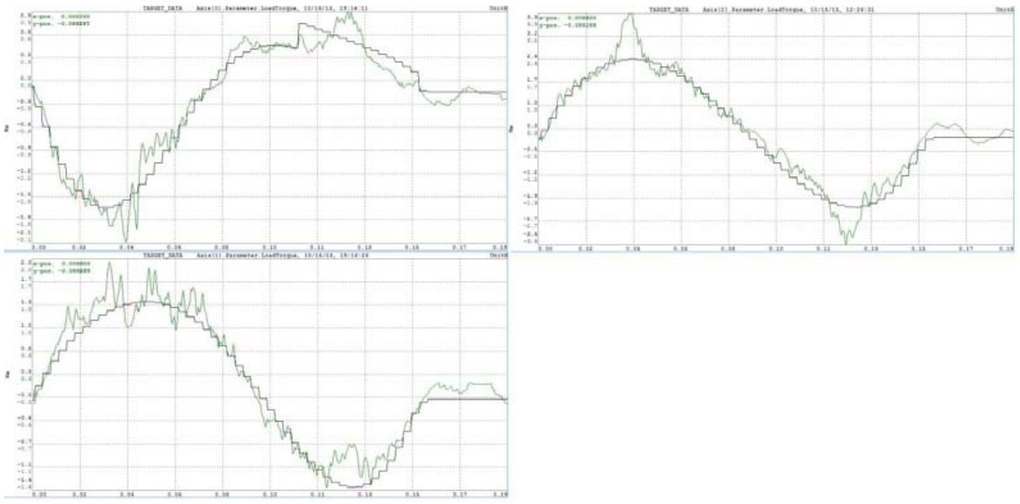


Figure 10.7 Torque from the model (black colour) and actual torque (green colour) for a movement between two positions. Axis 1 is top left, axis 2 is down left and axis 3 top right.

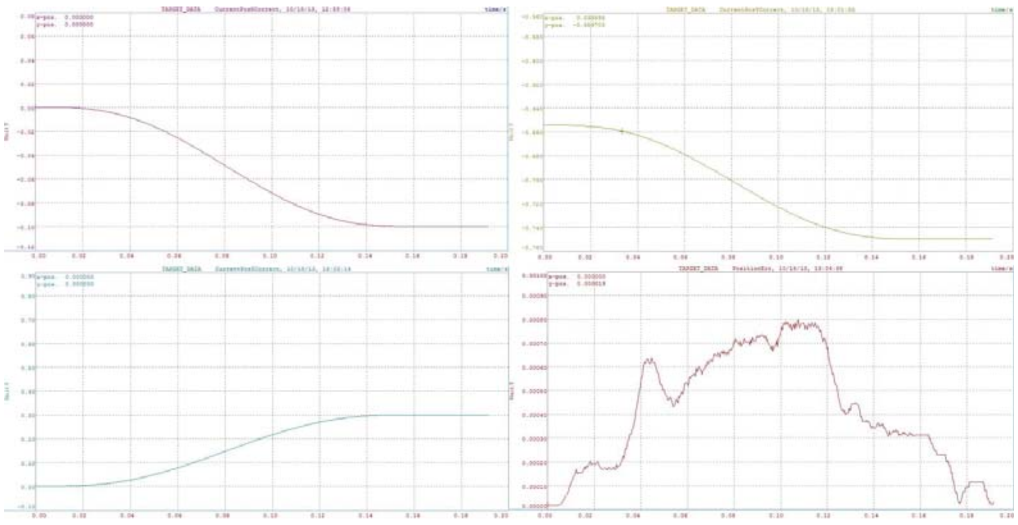


Figure 10.8 Set position and position error for the travelling plate in meters. Top left plot shows the x-position, top right shows the y-position, lower left shows the z-position and lower right shows the position error.

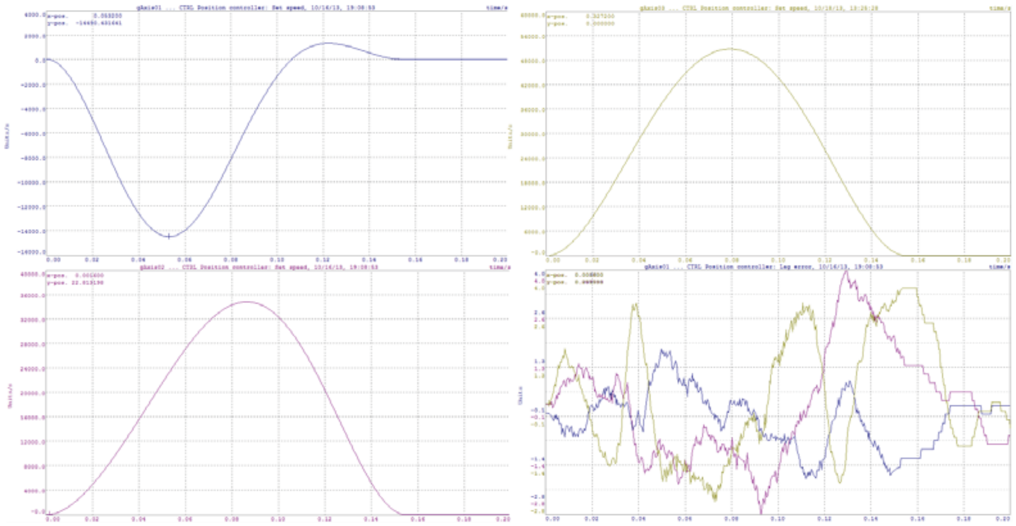


Figure 10.9 Set speed in the speed controller for axis 1 (blue), axis 2 (purple) and axis 3 (green). Lower right plot shows the angle position error for the three axes.

10.6.2 Without dynamic model

10.6.2.1 Moving to a position

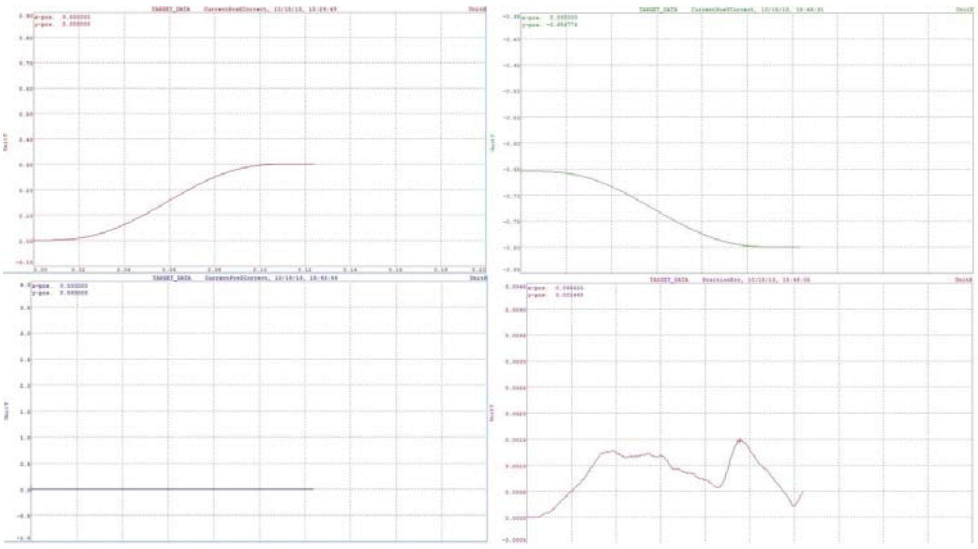


Figure 10.10 Set position and position error in meters for moving the travelling plate to a position. Top left is x-position, top right is y-position, lower left is z-position and lower right is the position error.

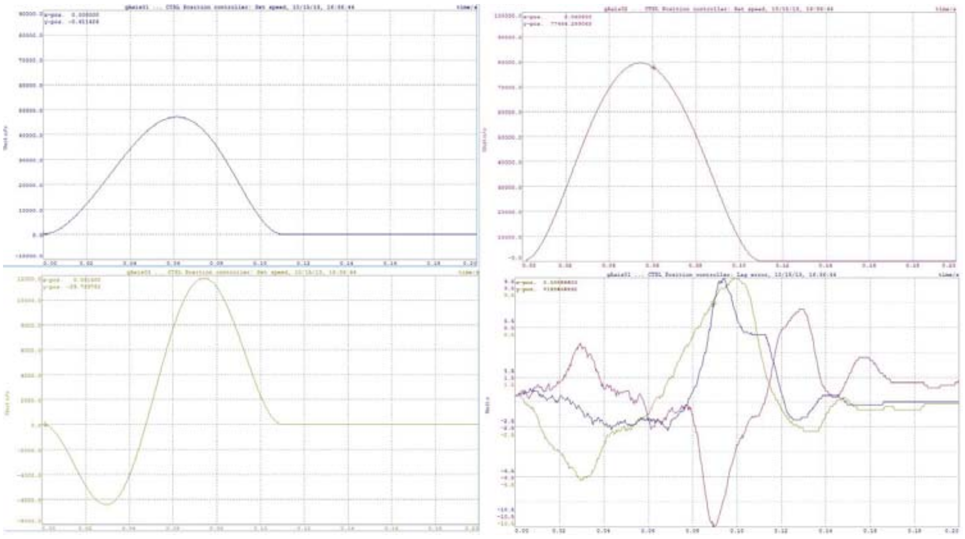


Figure 10.11 Set speed in the speed controller when moving the travelling plate to a position for axis 1 (blue), axis 2 (purple) and axis 3 (khaki). Lower right plot shows the angle position error for the three axes.

10.6.2.2 Moving in a circle

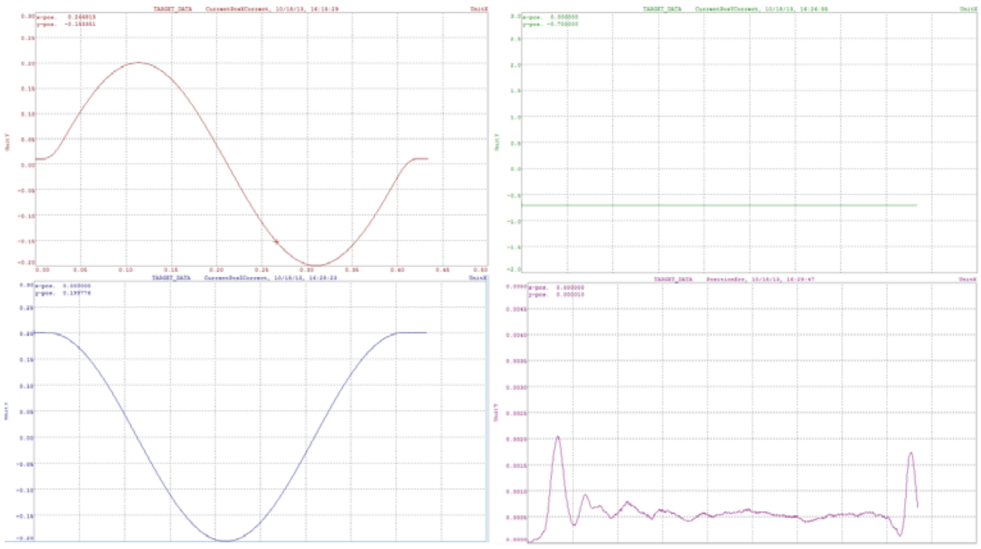


Figure 10.12 Set position and position error in meters for moving the travelling plate in a circle. Top left is x-position, top right is y-position, lower left is z-position and lower right is the position error.

10.6.3 Comparing with and without dynamic model

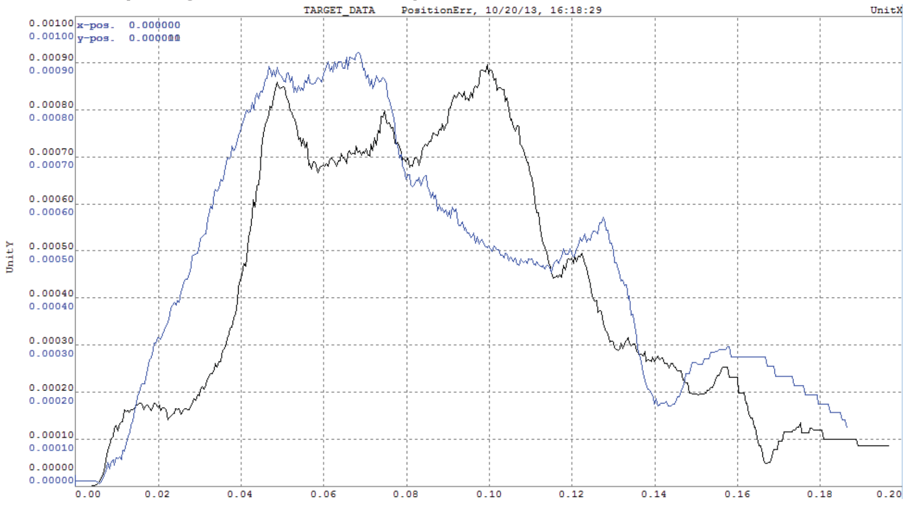


Figure 10.13 Position error for the travelling plate for moving to a position with the dynamic model (black) and without (blue)

11 Conclusions

11.1 Torque experiments

11.1.1 Torque constant

The reason the torque-constant is three times as large could be explained by that the value given in the ABB motor data is not the actual mechanical torque but electrical torque. Since the motor has three pole pairs one electrical rotation equals one third mechanical rotations.

11.1.2 Speed torque

If the same would be valid for the torque losses as for the torque constant the values should be three times as big as in the motor data, but this results in that the theoretical values for the torque are much bigger than the torque from the experiments. This is unrealistic since the total torque from the experiments should be at least as big as the speed dependent torque losses from the data. The reason for this is that the experiments are done with a gear and an arm connected to the motor which obviously can't lower the friction. The conclusion is that these values should not be multiplied with three as for the torque constant.

That the torque differences between ω_1 and ω_2 are about the same for both theory and reality is then strange since one expects more losses due to friction. However the measurements are not very exact and it can as well be that the torque is a bit more. This expected result with more speed dependent torque losses is achieved between ω_0 and ω_1.

11.1.3 Acceleration experiments

As can be seen in the constant acceleration experiments there is more than 50 % difference between the theoretical values and the experimental. In the varying acceleration experiments it is also noticed. This difference between theory and experiment could be explained that the inertia for the gear or that the efficiency for the gear is not included in the model. A gear with bearing friction can be modelled with another component in MapleSim [25].

11.2 Experiments with position control for the whole robot

11.2.1 With dynamic models

As can be seen in Figure 10.7 the torque from the model and the actual torque in the axis are very similar. The spikes in the torque in the second axis at the peak torques are there because the middle arm of the robot is shaking when moving the robot this fast, but the robot can move even faster since the maximum torque and maximum speed for the motors are not reached. One reason this is not done is that the dynamic model is not guaranteed to be stable but the main reason is that the calculations is so demanding for this dynamic model that the PLC might be overloaded (see Chapter 11.3). In Figure 10.8 position error for the travelling plate is shown, which is very small compared to if the robot would have been moved with no feed forward at all. That can be seen by deactivating the feed forward on the drives, but no results were saved by doing this.

11.2.2 Without dynamic models

In Figure 10.12 when moving the robot in a circle the position error is largest at start and stop of the movement. This could be explained that the torque needed to accelerate up to a constant angle velocity is larger than the torque needed to keep it moving in this circle with the needed circular acceleration. It could also be explained that the path created is a 3rd degree polynomial for the velocity which compared to the 4th degree for the velocity when moving between positions gives a discontinued differentiation of the acceleration. A slower increase of the acceleration might be easier for the controller to follow. Another explanation could also be that the fourth robot arm is shaking at start and stop of a movement and this shaking is a disturbance at the motors that the feed forward is not able to compensate for. The shaking is also affecting the result when moving the robot between positions as in the experiment with the dynamic model. The effect is even larger since it is moving faster in these experiments. This effect is not visible in the position error, but since the position error is based on the calculation of the angle positions of the motor the actual position error for the travelling plate is probably larger since the fourth arm of the robot is directly connected to the travelling plate. At the travelling plate the shaking should have more effect than at the motor since this vibration is transferred through the arms of the robot which could damp it. If this is true or not could only be validated by measuring the actual position of the tool centre point under the travelling plate which is the interesting point since this point is used to interact with the environment by picking up and placing objects.

11.2.3 Comparing with and without dynamic model

By looking at the position error of the travelling plate for both movements (Figure 10.13) nothing can be said which method is better. However the feed forward based on the parameters assumes that the inertia is always the same for each axis independent on the position of the travelling plate and that the speed dependent friction torque increases linear with increasing angle velocity. The dynamic model is a better description of the reality and should give a smaller position error when trying many different movements of the robot at different accelerations, to different positions and with other paths. Unfortunately, as explained before, this is not possible at all accelerations because of starvation, overloading, the PLC.

11.3 ODE-solvers

The Euler method used, is used since it is the simplest and fastest method. Compared to other methods, like Runge-Kutta methods, its region of stability is smaller [26]. If another method like Runge-Kutta would be used instead the integration time would then increase because of the several extra calculation steps which have to be made [17]. The step size has to then be increased and even though the region of stability is larger this increase of the step size might make the solver unstable. By using a third method called Implicit Euler the option could be made in MapleSim to export the full symbolic Jacobian. This is not made in the other methods where the Jacobian is made numerical before exporting. The advantage of this method is that it could be more stable since fewer approximations are made in the dynamic model when exported, but by using this method the integration time is so much larger it makes no sense in using it, especially when for much longer integration time the Euler method with projection can be used.

When using the Euler method with projection as in *Dynamic Model 2* a trade-off between step-size and number of projections have to be made. By increasing the number of projections both the output of the model and the variables in it are more likely to be close to the correct ones but at the same time these projections take time which mean that with more projections a bigger step-size have to be used. This larger step size can make the method unstable. For this reason this method is only used when moving the robot slowly to be able to keep the values from diverging and keeping the model stable. If this method would be used when moving the robot fast, both few projections together with a short step size or more projections but with a longer step size will make the differential integration in the model unstable.

11.4 Delay

Since there is a delay from sending the value from the dynamic model to the motor and then also a delay for reading the actual torque used at the motor there will be a delay visible when comparing calculated torque and actual torque. This is visible in the varying acceleration experiments (Figure 10.3, Figure 10.4), but it is not visible when moving the whole robot. The reason for this is that the dynamic model has its own delay caused by its longer cycle-time. In the result it looks that they match each other, Figure 10.7, but in fact this delay cancels the other delay when comparing them and the calculated torque should be shifted to the right to show reality.

12 Future work

12.1 Baumgarte Constraint Stabilization

When using the model without projection, the solution can diverge very fast, so something recommended is to use the Baumgarte Constraint Stabilization [20] to stabilize it. The Baumgarte Constraint Stabilization combines the position, velocity and acceleration constraints in one expression together with two parameters that has to be set. To choose good Baumgarte parameters for the model empiric tests have to be done since there are no reliable methods in choosing these parameters [27]. Unfortunately the choice of these parameters have to be done in MapleSim before exporting it, in the current version, which means that doing these empirical tests would be very time-demanding (A suggestion to include them as parameters to be set in the exported code have been suggested to Maplesoft through their support). The Baumgarte Constraint Stabilization should not be used when moving the robot slowly or when it is not moving at all since the solution is, although it is stable, diverging from the correct one which is the reason constraint projection is used instead. This was seen in some tests done and it is also shown that Baumgarte Constraint Stabilization is not always the best method [28].

12.2 Delay

Also a simple but important thing is to consider is the delay caused by calculating the dynamic model in the system. This delay is especially big when using the second dynamic model. A solution would be to send the reference value to the models but delay the sending the values to the axis a time equal to the duration of the calculation of the dynamic model. To make sure that the time for calculating the dynamic models always is the same, the calculated torque should be sent at beginning of the program.

12.3 Simplifications

Further approximations and simplifications can be made in the model to allow a shorter cycle-time when using the second dynamic model with projection, for example moving the masses for the lower-arms to the upper arms and the travelling plate and ignore their inertia as done in [29]. Another thing that simplifies the generated code a little is to not include the rotation angle around the vertical axis, α_i, but to calculate the exact position (since they are a third of a circle apart is it possible to write sine and cosine of these rotations as exact quotients). And the last thing that can be tested in MapleSim is to include the position of the travelling plate in the dynamic model, but this was tested in simulations in the program and no observable improvements to lower the integration time could be seen. Even though the position, velocity and acceleration is known for the travelling plate from the path generation, this information can't be used in the forward dynamic model in MapleSim where only the position, velocity and acceleration for the motors should be used as input. The last modification could be to increase the minimum time for the fastest thread in the CPU. This was tried but didn't seem to improve the calculation-time for the dynamic models. The problem with increased cycle-time for the shortest cycle is that if the task class in which this function block is in is larger than 0.4ms, which is the cycle time of the drive, the set value transferred from the PLC to the drive is then interpolated on the drive [22]. This could give less precise reference values.

One method to construct the dynamic model with no need of solving any differential equations is to ignore the movement of the middle arm and the forearms of the robot and place a part of these masses in the travelling plate and in the upper arm. The inertias of these arms are then ignored. This simplification has been made in [7] and been used in many applications [30]. With these simplifications all the movement of the masses can be described in the travelling plate position, velocity and acceleration and the actuated joints angle position, velocity and acceleration. The torque contribution at the actuated joints from the moving upper arm and the moving travelling plate can be calculated using the Jacobian (32) and its derivative by using the principle of virtual force in the same way as done in [31]. Another way is to use the model created in MapleSim. If the movement of the middle arm in this model is ignored and a part of its mass placed in the travelling plate the only unknown constraint variables that are left in the algebraic equations that force the model to be solved with a differential equation solver are β_1 and β_2 for every forearm, here denoted β_{1i} and β_{2i} for arm i. The solution for β_{1i}, for robot arm i, can be extracted out of the position constraints in MapleSim:

$$-l_3 \sin(\beta_{1i}) - \sin(\alpha_i)\, z + \cos(\alpha_i)\, x = 0 \tag{51}$$

With solution

$$\beta_{1i} = \arcsin\left(\frac{-\sin(\alpha_i)\,z + \cos(\alpha_i)\,x}{l_3}\right) \tag{52}$$

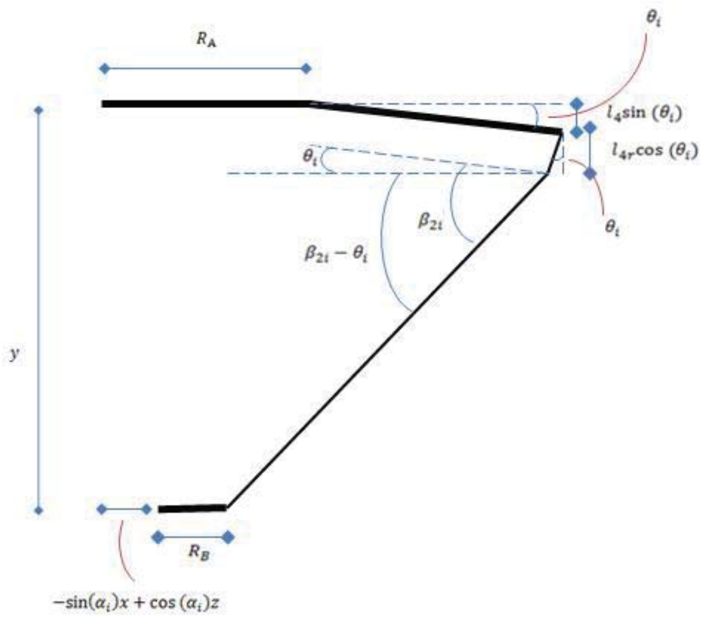


Figure 12.1 Projection of a robot arm on the yzi-plane.

The constraints equations from the model in MapleSim which include β_{2i} are more complicated but by looking at the geometry in Figure 12.1 where one robot-arm have been projected on to the YZi-plane. The Zi-plane is the plane that is specific for each arm and which the upper arm rotates in. The following equation can by hand be derived from the figure:

$$\tan(\beta_{2i} - \theta_i) = \frac{(-y - l_4\sin\theta_i - \mathrm{l}_{4\mathrm{r}}\cos(\theta_\mathrm{i}))}{R_A + l_4\cos(\theta_i) - R_B - l_{4r}\sin(\theta_i) + \sin(\alpha_i)\,x - \cos(\alpha_i)\,z} \tag{53}$$

where the minus sign before y is there since the y-coordinate is negative. The horizontal distance from the origin to the travelling plate in the picture is transformed according to (26) for the equation to be valid all robot arms. The solution to (53) is

$$\beta_{2i} = \theta_i + \arctan\left(\frac{(-y - l_4\sin\theta_i - \mathrm{l}_{4\mathrm{r}}\cos(\theta_\mathrm{i}))}{R_A + l_4\cos(\theta_i) - R_B - l_{4r}\sin(\theta_i) + \sin(\alpha_i)\,x - \cos(\alpha_i)\,z}\right) \tag{54}$$

By differentiating (52) and (54) once and twice in Maple the angle velocities and angle accelerations for these angles are given. All kinematic constraints in the model can be described with position, velocity and acceleration for the travelling plate and the actuated joints. If all the equations for the dynamic model are extracted from MapleSim, and (52), (54) and their velocities and accelerations are replacing the variables in these equations, a system of equations with only unknown dynamic variables is gained. The dynamic equations are then a set of 12 equations with 12 unknowns. 9 unknowns in the equations are the joint reaction forces and 3 are the desired torques. These can be tried to be solved with Maple function *solve* or another way. If a solution can't be found the equations can be copied to a MapleSim custom component but with this solution the need of a differential equation solver is still needed with given start values of the joint reaction forces, the dynamic variables.

12.4 Gravitational torque

If the velocities and accelerations in the dynamic model equations in MapleSim are set to zero the rest of the dynamic equations describe the dynamic model when the robot is not moving. By using the equations (52) and (54) and either find the solutions to the joint angles for the middle arm or ignore the angles by placing a part of the masses from the arm at the travelling plate (see Chapter 12.3), a solution to the gravitational component of the torque could be found by solving the equations with Maple's function *solve* as was done in Chapter 4.2. However this torque is small compared to the friction torque, and also very small compared to the torque from accelerating the robot. Unless the robot is used to lift something heavy it is unlikely that a model of only the gravitational torque would make a big difference for the control performance.

13 References

[1] ABB. http://www05.abb.com/global/scot/scot241.nsf/veritydisplay/e4712d3c88fd9240c125772e005b361b/$file/IRB%20360%20ROB0082EN_E.pdf, 3 March 2013.

[2] The advantages of symbolic computation. http://www.maplesoft.com/products/maplesim/symbolic_computation.aspx, September 17 2013.

[3] B&R training guide, The Basics of AsiM TM410. www.kongzhi.net/files/download.php?id=8346, 15 September 2013.

[4] Delta robot kinematics. http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/, 5 March 2013.

[5] Zsombor-Murray, P.J. Descriptive Geometric Kinematic Analysis of Clavel's "Delta" Robot, article, McGill University, 1 April 2004

[6] Ecorchard, Gaël and Maurine, Patrick. Self-Calibration of Delta Parallel Robots with Elastic Deformation Compensation. IEEE/RSJ International Conference on Intelligent Robots and Systems, 1283-1888, August 2-6 2005, Shaw Conference, Center Edmont, Alberta Canada.

[7] Codourey Alain. Dynamic modeling of parallel robots for Computed-Torque control. The International Journal of Robotics Research December 17, no 12 (1998): 1325-1336.

[8] Stefan Staicu. Recursive modeling in dynamics of Delta parallel robot. Robotica, Volume 27, no 2 (2009): 199 – 207.

[9] Stefan Staicu and Dan Zhang. A novel dynamic modeling approach for parallel mechanism analysis. Robotics and Computer-Integrated Manufacturing, volume 24, no 1 (2008): 167-172.

[10] Jiegao Wang, Clément M. Gosselin and Li Cheng. Modeling and simulation of robotic systems with closed kinematic chains using the virtual spring approach. Multibody System Dynamics, Volume 7, no 2 (2002): 145-170.

[11] Dynamics of Multibody Systems:Conventional and Graph-Theoretic Approaches http://real.uwaterloo.ca/~mcphee/sd652/intro_sd652.pdf, September 19 2013.

[12] MapleSim: Technological Superiority in Multi-Domain Physical Modeling and Simulation www.maplesoft.com/view.aspx?sf=7032, 29 October 2013

[13] High-speed robot, DeltaBot™, designed using Maplesoft™ technology. http://www.maplesoft.com/company/publications/articles/view.aspx?SID=100216, 6 March 2013.

[14] Nyberg, C and Liber AB. Mekanik Fortsättningskurs. Sundbyberg: Alfa Print, 2006.

[15] Spong, M.W., Hutchinson, S. and Vidyasagar M. Robot Modeling and Control. USA: John Wiley & Sons Inc, 2006.

[16] Gossens, P. and Richard, T. Using Symbolic Technology to Derive Inverse Kinematic Solutions for Actuator Control Development. http://www.maplesoft.com/Whitepapers/Mathmod2012_pgoossens_trichard_preprint_paper.pdf, 10 March 2013.

[17] Ljung, L and Glad, T. Modellbygge och simulering. 2nd ed. Lund: Studentlitteratur, 2004.

[18] Aguinaga, Iker. Multi-Body Systems, may 29 2012.
http://www.unav.es/adi/UserFiles/File/4000005502/6.Multibody.pdf, 23 September 2013.

[19] Getting Started with the MapleSim Connector for B&R Automation Studio.
http://www.maplesoft.com/documentation_center/toolboxes/MapleSim_BR_Connector_Getti
ng_Started_Guide.pdf, 20 July 2013.

[20] Dynamic Exports.
http://www.maplesoft.com/support/help/Maple/view.aspx?path=MapleSim%2FMultibody%2F
Dynamic_Exports, 21 July 2013.

[21] Constrained Nonlinear Optimization Algorithms.
http://www.mathworks.se/help/optim/ug/constrained-nonlinear-optimization-algorithms.html,
28 August 2013.

[22] B&R help explorer for Automation studio.

[23] Example program from B&R automation studio: Multiaxis.

[24] Personal communication regarding tuning with Klas Nilsson, LTH, 9 August 2013.

[25] Lossy Gear,
http://www.maplesoft.com/support/help/MapleSim/view.aspx?path=componentLibrary/1Dmec
hanics/rotational/bearingsGears/LossyGear, October 7 2013.

[26] Hairer, E. and Wanner G. Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. Springer Series in Comput. Mathematics, Vol. 14, 2nd ed. City: Springer-Verlag, 1996.

[27] Paulo Flores, Margarida Machado, Eurico Seabra and Miguel Tavares da Silva. A Parametric Study on the Baumgarte Stabilization Method for Forward Dynamics of Constrained Multibody Systems. Paper No. DETC2009-86362, 73-82. ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 4: 7th International Conference on Multibody Systems, Nonlinear Dynamics, and Control, Parts A, B and C, San Diego, California, USA, August 30–September 2, 2009.

[28] García de Jalón, J and Bayo, E. Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time challenge. New York: Springer-Verlag, 1994.

[29] Codourey, Alain. Dynamic Modelling and Mass Matrix Evaluation of the Delta Parallel Robot for Axes Decoupling Control. Proceedings of the 1996 IEEE/RS J International Conference on Intelligent Robots and Systems, Volume 3, 1211 – 1218, 4-8 Nov 1996, Senri Life Science Center, Osaka, Japan.

[30] Merlet, JP. Parallel robots. Dordrecht: Kluwer Academic Publishers, 2000.

[31] Olsson, André. Modeling and control of a Delta 3 robot. Master's thesis, ISRN LUTFD2/TFRT-5834-SE , Lund University, 2009.

[32] Inverse Kinematic Problem Solving: A Symbolic Approach Using MapleSim and Maple.
http://www.maplesoft.com/view.aspx?SF=101409/418184/InverseKinematic_W.pdf, 10 March 2013.

14 Appendix

14.1 Parameter used in the program in Automation Studio

Inertial4	Corresponding to parameter I_{l4xx}.
MassUpperarm	Corresponding to parameter m_U.
Masslowerarm	Corresponding to parameter m_{l3}.
Middleheight	Corresponding to parameter l_{13}.
RAD	Corresponding to parameter R_A.
l4center	Corresponding to parameter l_{4c}.
lmitt	Corresponding to parameter R_B.
Middle11	Length of one part of 4th actuated arm of the robot, the upper part which that is connected to the base with a universal joint (silver metal part for the central arm in Figure 1.1).
Middle11center	Distance to the mass centre from the universal joint of the part with length Middle11.
Middle12	Length of one part of 4th actuated arm of the robot, lower part, the telescopic part that slides outside the first (black part for the central arm in Figure 1.1).
Middle12center	Distance to the mass centre from the upper edge of the part with length Middle12.
Middlel2	Length of the universal joint at the end of the telescopic arm, which is connected to the travelling plate.
Middlel3	Length of the rod from the travelling plate to the tool centre point.
MassTravel	Mass of the travelling plate.
MassLoad	Mass of the load attached on the tool centre point.
MassSpring	Mass of each spring of the robot.
MassMiddle11	Mass for the part with length Middle11.
MassMiddle12	Mass for the part with length Middle12.
MassMiddle2	Mass of the universal joint with length Middlel2.
InertiaMiddle11X	The moment of inertia around x-axis for the part with length Middle11.
InertiaMiddle11Z	The moment of inertia around z-axis for the part with length Middle11.
InertiaMiddle11Y	The moment of inertia around y-axis for the part with length Middle11.
InertiaMiddle12X	The moment of inertia around x-axis for the part with length Middle12.
InertiaMiddle12Z	The moment of inertia around z-axis for the part with length Middle12.
InertiaMiddle12Y	The moment of inertia around y-axis for the part with length Middle12.
InertiaUnderX	Corresponding to parameter I_{l3zzxx}.
InertiaUnderZ	Corresponding to parameter I_{l3zzxx}.
Iloady	The moment of Inertia for the load attached on the tool centre point around the y-axis.
GearRatio	Corresponding to parameter r_G.
GearRatioAxis4	The gear ratio of the gear of the 4th actuated arm.
MotorInertia	Corresponding to parameter I_{motor}.
MotorInertiaAxis4	The inertia of the motor connected to the 4th actuated arm.
GearInertia	The inertia of the gears of arms 1-3.
GearInertiaAxis4	The inertia of the gear of the 4th actuated arm.

14.2 Generated code for the dynamic models

Here is part of the generated code for the dynamic models. First the state variables, output, input and parameters used, then configurable parameters used in *Dynamic Model 2* to set number of projections and tolerance and last the *SolverSetup* function with all the variables used in the model.

```
/**************************************************
 * Automatically generated by Maple.
 * Created On: Fri Jun 07 12:02:19 2013.
**************************************************/
#include <stdio.h>
#include <math.h>

/**************************************************
* Variable Definition for System:

* State variable(s):
*    x[ 0] = `Main.sub1.Middlearm1.R1.theta`(t)
*    x[ 1] = diff(`Main.sub1.Middlearm1.R1.theta`(t),t)
*    x[ 2] = `Main.sub1.Middlearm1.R3.theta`(t)
*    x[ 3] = diff(`Main.sub1.Middlearm1.R3.theta`(t),t)
*    x[ 4] = `Main.sub1.Middlearm1.R4.theta`(t)
*    x[ 5] = diff(`Main.sub1.Middlearm1.R4.theta`(t),t)
*    x[ 6] = `Main.sub1.Middlearm1.R6.theta`(t)
*    x[ 7] = diff(`Main.sub1.Middlearm1.R6.theta`(t),t)
*    x[ 8] = `Main.sub1.part1.R3.theta`(t)
*    x[ 9] = diff(`Main.sub1.part1.R3.theta`(t),t)
*    x[10] = `Main.sub1.part1.R5.theta`(t)
*    x[11] = diff(`Main.sub1.part1.R5.theta`(t),t)
*    x[12] = `Main.sub1.part2.R3.theta`(t)
*    x[13] = diff(`Main.sub1.part2.R3.theta`(t),t)
*    x[14] = `Main.sub1.part2.R5.theta`(t)
*    x[15] = diff(`Main.sub1.part2.R5.theta`(t),t)
*    x[16] = `Main.sub1.part3.R3.theta`(t)
*    x[17] = diff(`Main.sub1.part3.R3.theta`(t),t)
*    x[18] = `Main.sub1.part3.R5.theta`(t)
*    x[19] = diff(`Main.sub1.part3.R5.theta`(t),t)
*
* Output variable(s):
*    y[ 0] = `Main.sub1.TorqueAxis4`(t)
*    y[ 1] = `Main.sub1.torquepart1`(t)
*    y[ 2] = `Main.sub1.torquepart2`(t)
*    y[ 3] = `Main.sub1.torquepart3`(t)
*
* Input variable(s):
*    u[ 0] = `Main.sub1.MiddleInp`(t)
*    u[ 1] = `Main.sub1.Part1Inp`(t)
*    u[ 2] = `Main.sub1.Part2Inp`(t)
*    u[ 3] = `Main.sub1.Part3Inp`(t)
*
* Parameter(s):
*    p[ 0] = `Main.GearInertiaAxel4`
*    p[ 1] = `Main.GearInertia`
*    p[ 2] = `Main.GearRatioAxel4`
*    p[ 3] = `Main.GearRatio`
*    p[ 4] = `Main.Iloady`
*    p[ 5] = `Main.InertiaMiddle11X`
*    p[ 6] = `Main.InertiaMiddle11Y`
*    p[ 7] = `Main.InertiaMiddle11Z`
*    p[ 8] = `Main.InertiaMiddle12X`
*    p[ 9] = `Main.InertiaMiddle12Y`
*    p[10] = `Main.InertiaMiddle12Z`
*    p[11] = `Main.InertiaUnderX`
*    p[12] = `Main.InertiaUnderZ`
*    p[13] = `Main.Inertial4`
*    p[14] = `Main.MassLoad`
*    p[15] = `Main.MassMiddle11`
*    p[16] = `Main.MassMiddle12`
*    p[17] = `Main.MassMiddle2`
*    p[18] = `Main.MassSpring`
*    p[19] = `Main.MassTravel`
*    p[20] = `Main.MassUpperarm`
*    p[21] = `Main.Masslowerarm`
*    p[22] = `Main.Middle11`
*    p[23] = `Main.Middle11center`
*    p[24] = `Main.Middle12`
*    p[25] = `Main.Middle12center`
*    p[26] = `Main.Middle2`
*    p[27] = `Main.Middle3`
*    p[28] = `Main.Middleheight`
*    p[29] = `Main.MotorInertiaAxel4`
*    p[30] = `Main.MotorInertia`
*    p[31] = `Main.RAD`
*    p[32] = `Main.l1`
*    p[33] = `Main.l3`
*    p[34] = `Main.l4`
*    p[35] = `Main.l4center`
*    p[36] = `Main.l4r`
*    p[37] = `Main.lmitt`
*    p[38] = `Main.lspring`
*    p[39] = `Main.sub1.part3.alfa`
*    p[40] = `Main.sub1.part2.alfa`
*    p[41] = `Main.sub1.part1.alfa`
*
************************************************/

/* Configurable parameters */
#define CONITER 3.000000e0
#define CONTOL 1.000000e-05
#define INITCONITER 15
#define INITCONTOL 1.000000e-08
#define INITITER 50
#define INITTOL 1.000000e-08
#define INITWEIGHT 2.000000e+01
#define INCONTOL 1e-14



static void SolverSetup(double t0, double *ic, double *u, double *p, double *y, double h,
SolverStruct *S)
{
              long i;

              S->h = h;
              S->w[0] = t0;
              S->w[1] = Middlearm1_R1_theta;
              S->w[2] =  Middlearm1_R1_thetadot;
              S->w[3] =  Middlearm1_R3_theta;
              S->w[4] =  Middlearm1_R3_thetadot;
              S->w[5] = Middlearm1_R4_theta;
              S->w[6] =  Middlearm1_R4_thetadot;
              S->w[7] = Middlearm1_R6_theta;
              S->w[8] =  Middlearm1_R6_thetadot;
              S->w[9] =  part1_R3_theta;
              S->w[10] =  part1_R3_thetadot;
              S->w[11] = part1_R5_theta;
              S->w[12] =  part1_R5_thetadot;
```

```
S->w[13] =  part2_R3_theta;
S->w[14] =  part2_R3_thetadot;
S->w[15] =  part2_R5_theta;
S->w[16] =  part2_R5_thetadot;
S->w[17] =  part3_R3_theta;
S->w[18] =  part3_R3_thetadot;
S->w[19] =  part3_R5_theta;
S->w[20] =  part3_R5_thetadot;



//forces and torques for the middlearm joints...
S->w[21] =  var21;
S->w[22] =  var22;
S->w[23] =  Middlearm1_R2_M;
S->w[24] =  var24;
S->w[25] =  var25;





S->w[26] = part1_R4_M;
S->w[27] = part2_R4_M;
S->w[28] = part2_S1_Fx;
S->w[29] =  part2_S1_Fy;
S->w[30] =  part2_S1_Fz;

S->w[31] = part3_R4_M;
S->w[32] = part3_S1_Fx;
S->w[33] =  part3_S1_Fy;
S->w[34] =  part3_S1_Fz;

//outputs
S->w[35] =  Middlearm1_R2_M/RobotParameter.GearRatio;
S->w[36] =  part1_R4_M/RobotParameter.GearRatio;
S->w[37] =  part2_R4_M/RobotParameter.GearRatio;
S->w[38] =  part3_R4_M/RobotParameter.GearRatio;




//differentiated inputs
S->w[39]=angle4vel;
S->w[40]=angle4acc;
S->w[41]=-angle2vel;
S->w[42]=-angle2acc;
S->w[43]=-angle3vel;
S->w[44]=-angle3acc;
S->w[45]=-angle1vel;
S->w[46]=-angle1acc;


//input
S->w[47] =  Middlearm1_R2_theta;
S->w[48] =  -part1_R4_theta;
S->w[49] =  -part2_R4_theta;
S->w[50] =  -part3_R4_theta;





S->w[51] =  RobotParameter.GearInertiaAxis4;
S->w[52] =  RobotParameter.GearInertia;
S->w[53] =  RobotParameter.GearRatioAxis4;
S->w[54] =  RobotParameter.GearRatio;
```

```
            S->w[55] =  RobotParameter.Iloady;
            S->w[56] =  RobotParameter.InertiaMiddle11X;
            S->w[57] =  RobotParameter.InertiaMiddle11Y;
            S->w[58] =  RobotParameter.InertiaMiddle11Z;
            S->w[59] =  RobotParameter.InertiaMiddle12X;
            S->w[60] =  RobotParameter.InertiaMiddle12Y;
            S->w[61] =  RobotParameter.InertiaMiddle12Z;
            S->w[62] =  RobotParameter.InertiaUnderX;
            S->w[63] =  RobotParameter.InertiaUnderZ;
            S->w[64] =  RobotParameter.Inertial4;
            S->w[65] =  RobotParameter.MassLoad;
            S->w[66] =  RobotParameter.MassMiddle11;
            S->w[67] =  RobotParameter.MassMiddle12;
            S->w[68] =  RobotParameter.MassMiddle2;
            S->w[69] =  RobotParameter.MassSpring;
            S->w[70] =  RobotParameter.MassTravel;
            S->w[71] =  RobotParameter.MassUpperarm;
            S->w[72] =  RobotParameter.Masslowerarm;
            S->w[73] =  RobotParameter.Middle11;
            S->w[74] =  RobotParameter.Middle11center;
            S->w[75] =  RobotParameter.Middle12;
            S->w[76] =  RobotParameter.Middle12center;
            S->w[77] =  RobotParameter.Middlel2;
            S->w[78] =  RobotParameter.Middlel3;
            S->w[79] =  RobotParameter.Middleheight;
            S->w[80] =  RobotParameter.MotorInertiaAxis4;
            S->w[81] =  RobotParameter.MotorInertia;
            S->w[82] =  RobotParameter.RAD;
            S->w[83] =  RobotParameter.l1;
            S->w[84] =  RobotParameter.l3;
            S->w[85] =  RobotParameter.l4;
            S->w[86] =  RobotParameter.l4center;
            S->w[87] =  RobotParameter.l4r;
            S->w[88] =  RobotParameter.lmitt;
            S->w[89] =  RobotParameter.lspring;
            S->w[90] =  0.00000000000000000e+00;
            S->w[91] =  2.09439510300000010e+00;
            S->w[92] =  4.18879020400000000e+00;

            for(i=0; i<NINP; i++) S->w[i+NDIFF+NIX1-NINP+1]=u[i];
            for(i=0;i<NDIFF;i++) S->w[i+NEQ+NPAR+1]=0.0;
            if(p) for(i=0; i<NPAR; i++) S->w[i+NEQ+1]=p[i];
            if(ic) for(i=0;i<NDIFF;i++) S->w[i+1]=ic[i];
            numdiffinp(S->w,1);
            numdiffinp(S->w,1);
            fp(NEQ,S->w[0],&S->w[1],&S->w[NEQ+NPAR+1]);
            if(S->w[NEQ+NPAR+1]-S->w[NEQ+NPAR+1]!=0.0) {
                        SolverError(S,"index-1 and derivative evaluation failure");
                        return;
            }
#if INITCONITER>0
            i=Projection(S->w[0],&S->w[1],INITCONTOL,INITCONITER,NULL);
            if(i>0 && i!=3) {
                        SolverError(S,"constraint projection failure");
                        return;
            }
            fp(NEQ,S->w[0],&S->w[1],&S->w[NEQ+NPAR+1]);
            if(S->w[NEQ+NPAR+1]-S->w[NEQ+NPAR+1]!=0.0) {
                        SolverError(S,"index-1 and derivative evaluation failure");
                        return;
            }
#endif
            SolverOutputs(y,S);
}
```

14.3 Adjustments between kinematic models and dynamic model

In the kinematic models the assumption is made that the upper arms are connected directly to the forearms, but in reality and in the dynamic model the forearms are connected in the lower edge of the upper arms (Figure 14.1). To correct this, the length of the parameter l_4 is changed to l_{41} according to figure A.1.

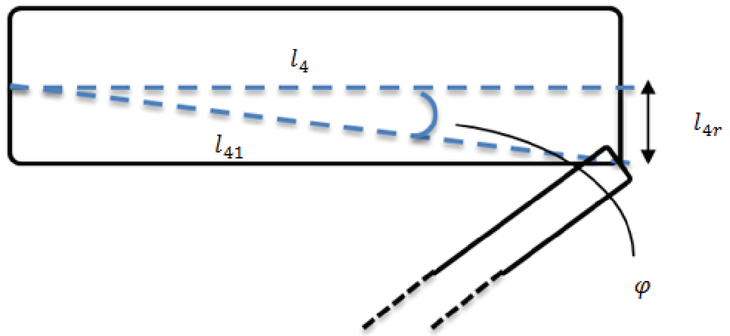


Figure 14.1 Definition of variables for the adjustment between kinematic and dynamic model.

According to:

$$l_{41} = \frac{l_4}{\cos(\varphi)} \tag{55}$$

where $\varphi = \tan^{-1}\left(\frac{l_{4r}}{l_4}\right)$

This value of l_{41} is used in the inverse and forward kinematics. But since the resulting angle from the inverse kinematics is the angle from the joint to the xz-plane, phi is added to this before sending it as an input to the dynamic model and the motors. For the forward kinematics φ has to be subtracted from the read angle of the motors to give the correct position of the travelling plate.

14.4 Inverse kinematic with MapleSim

The method described here is from [32]:

To find the inverse kinematic solution with a full model of the robot in MapleSim,with parallel forearms, is used but without the middlearm. Another difference is that the position of the travelling plate is guessed, since otherwise this position is not included in the model. By guessing the position accurately enough a successful simulation is achieved. Using MapleSims function multibody analysis the position constraints for the model can be extracted. This could also be done with the MapleSim model of the robot with singel forearms as well.

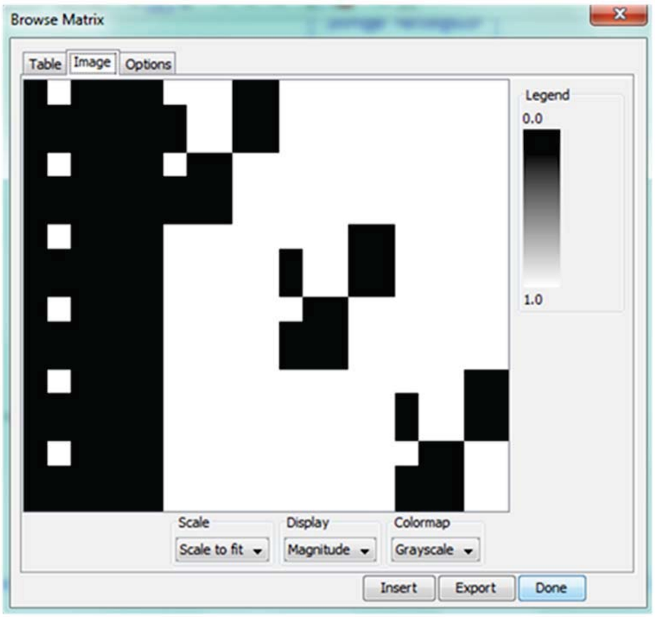


Figure 14.2 Jacobian for position constraint

By looking at the extracted equations one finds 6 groups of three equations where each group of equations only contain joint angles of one upperarm and one of the parallel forearms, the travelling plate position and rotation. This can be seen in a matrix (Figure 14.2), where the white areas are elements with value zero and the black areas are elements that differ from zero. For example the first three rows represent three constraint equations with dependency on the travelling plate position (column 1,2,3) and rotation (column 4,5,6), the revolute joint angle θ (column 7) and two angles which describes the rotation for one joint which connects the travelling plate and one rod (rotation around local z axis, β_2, and x axis, β_1). Since there is no rotation of the travelling plate the columns 4,5 and 6 are zero. Left in the three equations is the known position of the travelling plate (x, y, z), the unkown revolute joint angle θ_1, the unknown relative angles between one rod and the travelling plate for rotation around the same axis as θ_1, the local z-axis β_2, and the local x-axis β_1. The three equations are then:

$$-l3 \cdot \sin(\beta_1) - \sin(\alpha) \cdot z + \cos(\alpha) \cdot x = 0 \tag{56}$$

$$\begin{aligned}\cos(\theta) \cdot y - \cos(\alpha) \cdot \sin(\theta) \cdot z - \sin(\alpha) \cdot \sin(\theta) \cdot x - \sin(\theta) \cdot R_B + \sin(\theta) \cdot R_A \\ + \cos(\theta) \cdot l_3 \cdot \cos(\beta_1) \cdot \cos(\beta_2) - \sin(\theta) \cdot l_3 \cos(\beta_1) \cdot \sin(\beta_2) = 0\end{aligned} \tag{57}$$

$$
\begin{aligned}
-l_4 + \cos(\theta)\cdot R_B + \sin(\alpha)\cos(\theta)\cdot x + \cos(\alpha)\cdot\cos(\theta)\cdot z - \cos(\theta)\cdot R_A + \sin(\theta)\cdot y \\
+ \sin(\theta)\cdot l_3\cdot\cos(\beta_1)\cdot\cos(\beta_2) + \cos(\theta)\cdot l_3\cdot\cos(\beta_1)\cdot\sin(\beta_2) = 0
\end{aligned}
\quad \textbf{(58)}
$$

These three equations can be put in a custom component in MapleSim which then needs a guessed initial value to the arm angle to be able to find the correct solution (since there are two solutions and the arctan function need to know which quadrat the solution is in). Then three costum components can be used with the travelling plate position as input and a good guessed value as starting value for the angles to get the the motor angles.

The velocity jacobian from MapleSim can also be used to find the solution from speed for the travelling plate to velocity for the motors. It is made in a similar way with finding three equations for each arm that is solved with custom components in MapleSim. The acceleration jacobian is in MapleSim only the velocity jacobian diffirentiated.

For exact solution to the inverse kinematics these three equations could be solved for θ with Maples function solve or if the beta-angles are known (chapter 12.3) the expressions for them could be put in one of the equations and be solved. Solving these equations have been tried but gives very long expressions. To solve it with the beta-angles expressions has not been tried.

14.5 Acceleration kinematic solution

The expressions for the robot arm angle accelerations of axis 1-3 is given in Figures 14.3-14.5. The extra *1* in almost every variable have no significant meaning and the variables $l_{mitt} = R_B$ and $Rb = R_A$.

$$
\begin{aligned}
&-\Bigg(\Big(\frac{d}{dt}\mathit{x1}(t)\Big)\mathit{xdot1}(t) + \mathit{x1}(t)\Big(\frac{d}{dt}\mathit{xdot1}(t)\Big) + \Big(\frac{d}{dt}\mathit{ydot1}(t)\Big)\mathit{y1}(t) \\
&\quad + \mathit{ydot1}(t)\Big(\frac{d}{dt}\mathit{y1}(t)\Big) + \Big(\frac{d}{dt}\mathit{ydot1}(t)\Big)\mathit{l4}\sin(\theta\mathit{11}(t)) \\
&\quad + \mathit{ydot1}(t)\,\mathit{l4}\cos(\theta\mathit{11}(t))\Big(\frac{d}{dt}\theta\mathit{11}(t)\Big) + \Big(\frac{d}{dt}\mathit{zdot1}(t)\Big)\mathit{z1}(t) + \mathit{zdot1}(t)\Big(\frac{d}{dt}\mathit{z1}(t)\Big) \\
&\quad + \Big(\frac{d}{dt}\mathit{zdot1}(t)\Big)\mathit{lmitt} - \Big(\frac{d}{dt}\mathit{zdot1}(t)\Big)\mathit{Rb} - \Big(\frac{d}{dt}\mathit{zdot1}(t)\Big)\mathit{l4}\cos(\theta\mathit{11}(t)) \\
&\quad + \mathit{zdot1}(t)\,\mathit{l4}\sin(\theta\mathit{11}(t))\Big(\frac{d}{dt}\theta\mathit{11}(t)\Big)\Bigg)\Big/\Big(\mathit{l4}\big(-\sin(\theta\mathit{11}(t))\,\mathit{Rb} + \sin(\theta\mathit{11}(t))\,\mathit{lmitt} \\
&\quad + \cos(\theta\mathit{11}(t))\,\mathit{y1}(t) + \sin(\theta\mathit{11}(t))\,\mathit{z1}(t)\big)\Big) + \Bigg(\big(\mathit{x1}(t)\,\mathit{xdot1}(t) + \mathit{ydot1}(t)\,\mathit{y1}(t) \\
&\quad + \mathit{ydot1}(t)\,\mathit{l4}\sin(\theta\mathit{11}(t)) + \mathit{zdot1}(t)\,\mathit{z1}(t) + \mathit{zdot1}(t)\,\mathit{lmitt} - \mathit{zdot1}(t)\,\mathit{Rb} \\
&\quad - \mathit{zdot1}(t)\,\mathit{l4}\cos(\theta\mathit{11}(t))\big)\Big(-\cos(\theta\mathit{11}(t))\Big(\frac{d}{dt}\theta\mathit{11}(t)\Big)\mathit{Rb} \\
&\quad + \cos(\theta\mathit{11}(t))\Big(\frac{d}{dt}\theta\mathit{11}(t)\Big)\mathit{lmitt} - \sin(\theta\mathit{11}(t))\Big(\frac{d}{dt}\theta\mathit{11}(t)\Big)\mathit{y1}(t) \\
&\quad + \cos(\theta\mathit{11}(t))\Big(\frac{d}{dt}\mathit{y1}(t)\Big) + \cos(\theta\mathit{11}(t))\Big(\frac{d}{dt}\theta\mathit{11}(t)\Big)\mathit{z1}(t) \\
&\quad + \sin(\theta\mathit{11}(t))\Big(\frac{d}{dt}\mathit{z1}(t)\Big)\Big)\Bigg)\Big/\Big(\mathit{l4}\big(-\sin(\theta\mathit{11}(t))\,\mathit{Rb} + \sin(\theta\mathit{11}(t))\,\mathit{lmitt} \\
&\quad + \cos(\theta\mathit{11}(t))\,\mathit{y1}(t) + \sin(\theta\mathit{11}(t))\,\mathit{z1}(t)\big)^2\Big)
\end{aligned}
$$

Figure 14.3 Angle acceleration for axis 1.

$$
\begin{aligned}
&\Bigg(\bigg(-\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,x1(t)\Big)-\frac{1}{2}\,l4\sin\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\sqrt{3}\bigg)\,xdot1(t)+\bigg(-x1(t)+\frac{1}{2}\,\big(-lmitt\\
&\quad+Rb+l4\cos\big(\theta21(t)\big)\big)\sqrt{3}\bigg)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,xdot1(t)\Big)-\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,y1(t)\\
&\quad+l4\cos\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\bigg)\,ydot1(t)-\big(y1(t)+l4\sin\big(\theta21(t)\big)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,ydot1(t)\Big)\\
&\quad+\bigg(-\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,z1(t)\Big)+0.5\,l4\sin\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\bigg)\,zdot1(t)+\big(-z1(t)+0.5\,lmitt\\
&\quad-0.5\,Rb-0.5\,l4\cos\big(\theta21(t)\big)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,zdot1(t)\Big)\Bigg)\bigg/\bigg(l4\,\Big(\frac{1}{2}\sqrt{3}\,\sin\big(\theta21(t)\big)\,x1(t)\\
&\quad-\sin\big(\theta21(t)\big)\,Rb+\sin\big(\theta21(t)\big)\,lmitt+\cos\big(\theta21(t)\big)\,y1(t)-0.5\sin\big(\theta21(t)\big)\,z1(t)\Big)\bigg)\\
&\quad-\Bigg(\bigg(\Big(\Big(-x1(t)+\frac{1}{2}\,\big(-lmitt+Rb+l4\cos\big(\theta21(t)\big)\big)\sqrt{3}\Big)\,xdot1(t)-\big(y1(t)\\
&\quad+l4\sin\big(\theta21(t)\big)\big)\,ydot1(t)+\big(-z1(t)+0.5\,lmitt-0.5\,Rb\\
&\quad-0.5\,l4\cos\big(\theta21(t)\big)\big)\,zdot1(t)\Big)\Big(\frac{1}{2}\sqrt{3}\,\cos\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\,x1(t)\\
&\quad+\frac{1}{2}\sqrt{3}\,\sin\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,x1(t)\Big)-\cos\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\,Rb\\
&\quad+\cos\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\,lmitt-\sin\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\,y1(t)\\
&\quad+\cos\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,y1(t)\Big)-0.5\cos\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta21(t)\Big)\,z1(t)\\
&\quad-0.5\sin\big(\theta21(t)\big)\Big(\frac{\mathrm{d}}{\mathrm{d}t}\,z1(t)\Big)\Big)\bigg)\bigg/\bigg(l4\,\Big(\frac{1}{2}\sqrt{3}\,\sin\big(\theta21(t)\big)\,x1(t)\\
&\quad-\sin\big(\theta21(t)\big)\,Rb+\sin\big(\theta21(t)\big)\,lmitt+\cos\big(\theta21(t)\big)\,y1(t)-0.5\sin\big(\theta21(t)\big)\,z1(t)\Big)^{2}\bigg)
\end{aligned}
$$

Figure 14.4 Angle acceleration for axis 2.

$$
\begin{aligned}
&\bigg(\bigg(-\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,x1(t)\bigg)+\frac{1}{2}\,l4\sin\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\sqrt{3}\bigg)\,xdot1(t)+\bigg(-x1(t)-\frac{1}{2}\,\big(-lmitt\\
&\quad+Rb+l4\cos\big(\theta31(t)\big)\big)\sqrt{3}\bigg)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,xdot1(t)\bigg)-\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,y1(t)\\
&\quad+l4\cos\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\bigg)\,ydot1(t)-\big(y1(t)+l4\sin\big(\theta31(t)\big)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,ydot1(t)\bigg)\\
&\quad+\bigg(-\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,z1(t)\bigg)+0.5\,l4\sin\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\bigg)\,zdot1(t)+\big(-z1(t)+0.5\,lmitt\\
&\quad-0.5\,Rb-0.5\,l4\cos\big(\theta31(t)\big)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,zdot1(t)\bigg)\bigg)\bigg/\bigg(l4\,\bigg(-\frac{1}{2}\sqrt{3}\,\sin\big(\theta31(t)\big)\,x1(t)\\
&\quad-\sin\big(\theta31(t)\big)\,Rb+\sin\big(\theta31(t)\big)\,lmitt+\cos\big(\theta31(t)\big)\,y1(t)-0.5\sin\big(\theta31(t)\big)\,z1(t)\bigg)\bigg)\\
&\quad-\bigg(\bigg(\bigg(-x1(t)-\frac{1}{2}\,\big(-lmitt+Rb+l4\cos\big(\theta31(t)\big)\big)\sqrt{3}\bigg)\,xdot1(t)-\big(y1(t)\\
&\quad+l4\sin\big(\theta31(t)\big)\big)\,ydot1(t)+\big(-z1(t)+0.5\,lmitt-0.5\,Rb\\
&\quad-0.5\,l4\cos\big(\theta31(t)\big)\big)\,zdot1(t)\bigg)\,\bigg(-\frac{1}{2}\sqrt{3}\,\cos\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\,x1(t)\\
&\quad-\frac{1}{2}\sqrt{3}\,\sin\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,x1(t)\bigg)-\cos\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\,Rb\\
&\quad+\cos\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\,lmitt-\sin\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\,y1(t)\\
&\quad+\cos\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,y1(t)\bigg)-0.5\cos\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,\theta31(t)\bigg)\,z1(t)\\
&\quad-0.5\sin\big(\theta31(t)\big)\,\bigg(\frac{\mathrm{d}}{\mathrm{d}t}\,z1(t)\bigg)\bigg)\bigg)\bigg/\bigg(l4\,\bigg(-\frac{1}{2}\sqrt{3}\,\sin\big(\theta31(t)\big)\,x1(t)\\
&\quad-\sin\big(\theta31(t)\big)\,Rb+\sin\big(\theta31(t)\big)\,lmitt+\cos\big(\theta31(t)\big)\,y1(t)-0.5\sin\big(\theta31(t)\big)\,z1(t)\bigg)^{2}\bigg)
\end{aligned}
$$

Figure 14.5 Angle acceleration for axis 3.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* December 2013 *Document Number* ISRN LUTFD2/TFRT--5929--SE
Author(s) Kristofer Rosquist	*Supervisor* Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden Tore Hägglund, Dept. of Automatic Control, Lund University, Sweden (examiner) *Sponsoring organization*

Title and subtitle

Modelling and Control of a Parallel Kinematic Robot

Abstract

This Master's Thesis shows how to model the kinematics and forward dynamics for the parallel kinematic robot from ABB, *IRB340 FlexPicker,* and how to implement the control using software and hardware from B&R Automation. With conventional methods the forward kinematics, inverse kinematics and the kinematics for the velocities and accelerations of the robot are modelled. The forward dynamics, calculating the generated motor torques, of the robot is modelled with MapleSim and code generation of this model, to be implemented in the software from B&R Automation and used online during control, is used. In the software from B&R Automation, B&R Automation Studio, the kinematic model, path trajectories for the TCP with polynomial functions, an interface between the software and a PC and programs for communicating with the drives are also implemented together with a main program. To simplify, analyze and calculate equations Maple is used.

Experiments using the generated torque are made for comparison with the model and to improve it. The model is improved by measuring the friction torque and the torque caused by the inertia of the motor, a gear and one robot arm.

Because of the parallel structure of the robot the generated dynamic equation consists of complex differential equations which can overload the PLC during solving. To ensure stability of the dynamic model a second dynamic model with constraint stabilization is generated for use during slower movements. Despite the risk of starvation of the PLC the robot is moved at 10 g acceleration with good result comparing calculated and actual torque. Without using the desired feedforward from the dynamic equations, but instead using a simpler less dynamic feedforward method, the robot is accelerated at 16 g.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-65	*Recipient's notes*
Security classification		

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