Fingerprint Sensor Testing Using Force Feedback Control

Viktor Johansson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6014
ISSN 0280-5316

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

Printed in Sweden by Tryckeriet i E-huset
Lund 2016

Abstract

Testing and validation of fingerprint sensors with human labor is time consuming, costly and lacks the speed, accuracy and repeatability required for sufficient results. The thesis investigates the possibility of an automated solution using an industrial robot with force feedback control. Several force feedback controllers are implemented and evaluated. The best controller is implemented together with a graphical user interface and programming of an industrial robot manipulator. The setup can successfully test multiple sensors and display pictures taken with a fingerprint sensor, together with the applied force.

Acknowledgements

I would like to thank Fingerprint Cards and the Department of Automatic Control, LTH, for the opportunity to work on this thesis. Further I would like to thank the people working and studying in and around the thesis room and the robotics lab, for the collaboration regarding space and access to the robot. A special thanks to my advisors, Anders Robertsson at the Department of Automatic Control, LTH and Markus Andersson at Fingerprint Cards for the time, interest and help they put into my thesis.

Contents

1. Introduction **9**
- 1.1 Background . . . 9
- 1.2 Problem formulation . . . 10
- 1.3 Method . . . 10
- 1.4 Software tools . . . 11
- 1.5 Hardware . . . 13

2. Theory **19**
- 2.1 Robot frames and transformation matrices . . . 19
- 2.2 Kinematics and differential kinematics . . . 21
- 2.3 Force sensors . . . 23
- 2.4 Force measurements . . . 25
- 2.5 Controllers . . . 28
- 2.6 Modeling . . . 31

3. Methods **34**
- 3.1 Controllers . . . 34
- 3.2 System identification . . . 36
- 3.3 Simulations . . . 37
- 3.4 Summary of simulations . . . 40
- 3.5 Implementation of sensors . . . 42

4. Results **46**
- 4.1 Robot control GUI . . . 46
- 4.2 ABB Rapid Program . . . 48
- 4.3 ExtCtrl computer . . . 49

5. Conclusion and discussion **63**
- 5.1 Conclusion . . . 63
- 5.2 Models and system identification . . . 63
- 5.3 Force controllers . . . 64
- 5.4 Force sensors . . . 65
- 5.5 Future work . . . 67

Bibliography **68**

A. Appendix **70**
A.1 System identification script 70
A.2 6 DOF force sensor, gravity compensation script 71
A.3 Arduino code 73

Chapter 1

Introduction

1.1 Background

Advantages as increased productivity, better product quality and lowered costs are some of the reasons why the industry adapts more and more automatized solutions. Further reasons why robots gets a greater use are the robot's ability to handle more complex tasks, which leads to that an increasing market can benefit from advantages as speed, accuracy, repeatability and ability to work around the clock. The reason why robots can handle more complex tasks is the integration of different sensors. This gives the possibility to combine the exact movements of a robot with sensors as cameras and force sensors. Robots are thereby able to adapt to variations in the environment, which benefits both safety and increase the area of use. All these advantages do unfortunately come with some disadvantages. A robot has to be programmed to handle all the sensor data and refer this to the predefined trajectory. Writing the algorithm that can handle all possible cases is both time consuming and hard.

A company that has opened their eyes for the possibilities that a robot can bring is Fingerprint Card (FPC). FPC is a world leading company specializing in biometrical fingerprint sensors. They offer software and hardware solutions that enable the integration of fingerprint sensors including "smart cards", access control for mobiles, doors, safes, USB-sticks etc. Today, most of the verifications and tests of their sensors are done through human labor. These tests are repetitive and time consuming, making it an ideal task for a robot. An automated process would provide many of the advantages mentioned above, such as less deviations between the tests, shorter cycle times and higher continuity.

1.2 Problem formulation

The purpose of this thesis is to develop and examine a force feedback controller that can be used for the following purposes; to make a robot test environment for FPC fingerprint sensor modules or original equipment manufacturer (OME) customer module, in FPC lab purposes where the fingerprint sensor module is placed or mounted:

1. In a test jig.
2. On a FPC made phone.
3. On an OME phone.
4. On a vendor reference hardware board.

To perform the tests and make it more usable, a desire from FPC is a PC Windows program to control the robot that includes an API, provided by FPC, to the fingerprint sensor modules. The structure of the system should also make it possible to easily integrate new sensors and sensor modules.

The test made on these platforms will be used to:

1. Measure some form of “Key Performance Indicator” like “number of tests per second”.
2. Test different fingerprint sensor surfaces.
3. Test modules mounted in different hardware environments.
4. Test different FPC algorithms.
5. Test for fake finger detection and liveness detection.
6. Test different angles, pressures and fingerprint sensors.

1.3 Method

In order to solve all the problems mentioned in Section 1.2, a theoretical study is needed. The topics that will be covered are:

1. Robot kinematics.
2. Frames and translations.
3. Force control.
4. Force sensor calibration methods.

5. Price and performance of different setups.

6. Software tools to use for force control, communication, GUI etc.

A summary of these topics is presented on Sections 1.4, 1.5 and 2.1-2.6.

1.4 Software tools

Many different software tools are utilized in this thesis. A compilation of all the software tools used are presented in the following section.

1.4.1 ExtCtrl/Opcom

ExtCtrl is a program developed at LTH. The program has a graphical interface called Opcom, Figure 1.1, that makes it possible to read and modify data from the robot controller's main computer and modify this data before it is send to the axis computer, Section 4.3.1. The data includes current position, velocity, position reference and velocity reference, which is some of the data that will be utilized in this thesis. For a full description see Section 4.3.3. To add additional functionality it is possible to interface with other external programs e.g., a program that reads force measurements from a force sensor [Blomdell et al., 2010].

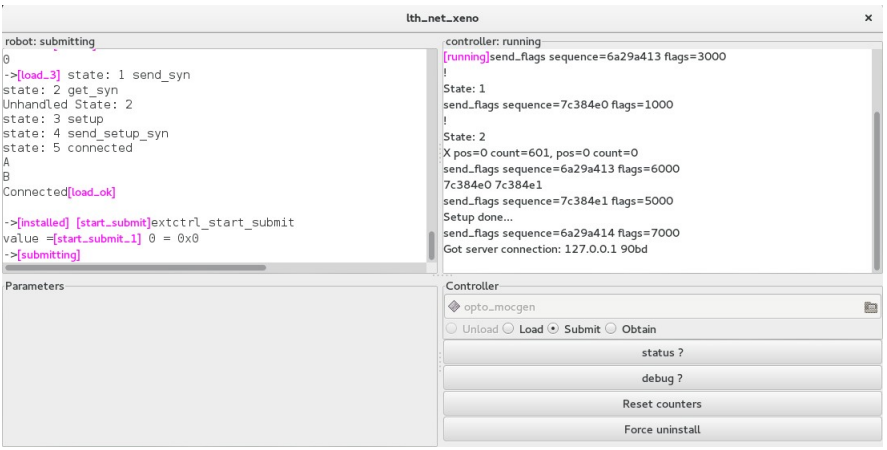


Figure 1.1 The Opcom interface.

1.4.2 Matlab/Simulink

Matlab is a powerful mathematical high level programing software. In addition to ordinary text based programing, Matlab also provides several extensions. One of these extensions is called Simulink. Simulink is a block diagram environment for

simulation and model based design. It supports simulations, automatic code generation using real time workshop (RTW), and can be used together with Matlab algorithms as well as many other text based programing languages like C#/C++, Java and Python. All the simulations and controllers in the thesis are programmed using Simulink. The Real Time Workshop extension is used to generate C-code from the Simulink-model. This code can then be used by the Opcom program. A library with Simulink-blocks containing predefined kinematics called ExtCtrl is also utilized, Figure 1.2 [Mathworks, 2016a].

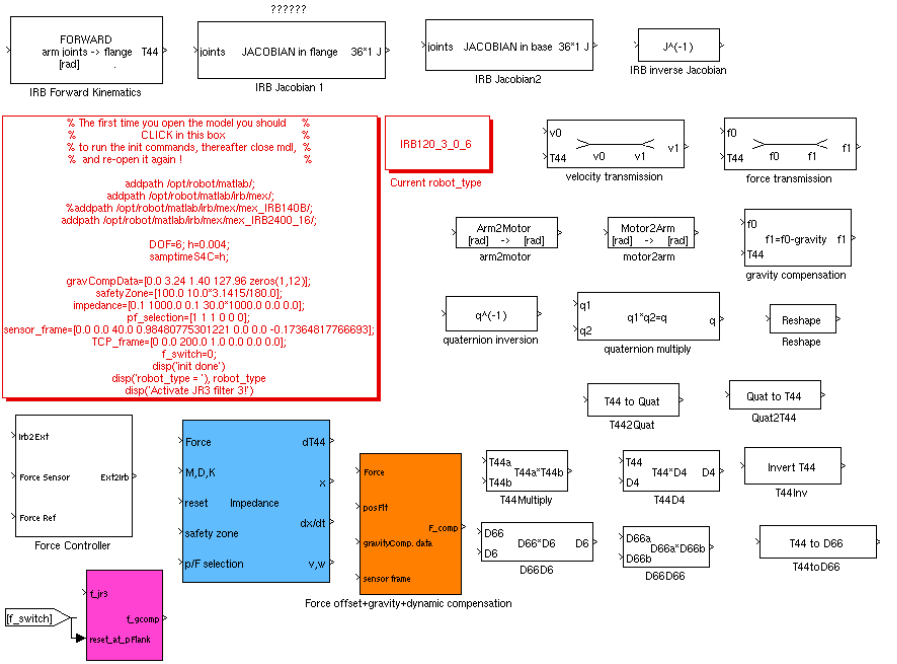


Figure 1.2 Blocks from the ExtCtrl library.

1.4.3 RobotStudio

RobotStudio is a powerful simulation program and editor for ABB robots, which allows the user to fully simulate a complete automated system together with an ABB robot, controller and complete environments. This is referred to as offline programing and is of great value for the user as this will lead to less downtime and therefore less production loss [*RobotStudio® It's just like having the real robot on your PC!*]

1.4.4 Rapid

Rapid is a software language developed by ABB for use in their robot controllers. Rapid is developed to make a simple interface to their robots and can be programmed online using their teach pendant, Figure 1.5, or offline using their robot system development tool RobotStudio.

The communication between the main controller and the force control program will be implemented using Mocgen instructions. Mocgen is developed at ABB but configured to be used with the ExtCtrl library, Section 1.4.2. This enables the velocity and position controller from ABB to interact and hand over control to the force controller.

1.4.5 ABB PC SDK

ABB PC SDK is a standard development kit from ABB that makes it possible to communicate with the robot via Ethernet and includes functionality as scanning the networks for controllers, sending data and starting execution of programs.

1.4.6 FPC's sensorAPI and Module Test Tool

Module Test Tool is a program that is developed and used by FPC and their customers to test and verify fingerprint sensors. SensorAPI is an api library developed by FPC that makes it possible to communicate with their sensor modules and includes functionality as *wait for finger detection*, *capture image* etc.

1.4.7 Arduino 1.6.9

Arduino 1.6.9 is an integrated programing environment (IDE) created to program Arduino compatible boards including the Arduino Uno used in this project. It runs on Windows, Mac OS X and Linux. The environment is written in Java and based on Processing and other open source software [*Arduino 1.6.9*].

1.4.8 Visual studio

Visual studio is a powerful integrated programing environment (IDE) developed by Microsoft. It supports development for mobile apps using IOS, Android or Windows, web apps, cloud apps, Windows apps and games, Microsoft Office development, Node.js development environment, Virtual C++, Python and .Net. It supports many different software languages including C/C++, VB.NET, C# and F#. Visual studio is used to develop the graphical interface and utilize ABB PC SDK and FPCs sensorAPI [*Application Development*].

1.5 Hardware

The hardware researched, tested and used are presented in Section 1.5. The hardware consists of robots, force sensors, fingerprint sensors, Arduino Uno board and fingerprint sensor modules.

1.5.1 Robots

The robot used in this thesis is the ABB IRB120, this due to availability and size. The ABB IRB120 is, in Section 1.5, compared to other robots that could be used for the same purpose.

ABB IRB120 ABB IRB120 is a small standard six axis industrial robot, Figure 1.3. It has an repeatability of +-0.01 mm, a max payload of 3 kg and a reach of 580 mm. The drawback of this robot is that it needs a safety zone and is thereby not designed to interact with humans [Keijser, 2012].

Figure 1.3 ABB IRB120 [Keijser, 2012].

ABB YuMi The ABB YuMi is a new dual-arm robot from ABB, Figure 1.4. The YuMi robot is developed to interact with humans and can sense when an external force is applied. Each arm has seven degrees of freedom (DOF) and has a max payload of 0.5 kg and a reach of 559 mm. Other advantages include lead through programming, ESD certification, integrated controller and no need for safety cages or zones. Due to the design with two arms it is possible to run two tests at the same time and thereby increase productivity [Crowther, 2015].

IRC5 IRC5 is the controller that comes with an ABB robot. ABB has several different versions of this controller. The ABB YuMi robot has a built in IRC5 controller and the ABB IRB120 comes with a compact controller, Figure 1.5. The IRC5 includes two controllers. The main controller calculates position and velocity reference from the trajectory specified in RAPID code. The motor controller or axis controller, as it is also called, takes the position reference, velocity reference and torque reference as inputs and calculates and applies the correct torques to the motors. A Flexpendant is also included with the IRC5 cabinet. From this interface one is able to jog and program the main computer using a touchscreen, joystick and some tactile buttons, Figure 1.5 [ABB, 2010].

Figure 1.4 ABB YuMi [Crowther, 2015].

Figure 1.5 IRC5 and Flexpendant [ABB, 2010].

Universal robot UR3 The UR3 robot, Figure 1.6, is a lightweight robot that, just as ABB YuMi robot, is developed to work and interact with humans. This results in the same advantages as for the YuMi robot, but it is designed as a regular six axis industrial robot. The max payload is 3 kg and it has a reach of 500 mm. The total weight of the robot is 11 kg, which can be compared to the IRB120 that has a weight of 25 kg. The UR3 robot has an repeatability of +-0.1 mm, which compared to the ABB IRB120's repeatability of +-0.01 mm, is not that impressive, but using force control this may not affect as much [Universal Robots, 2015].

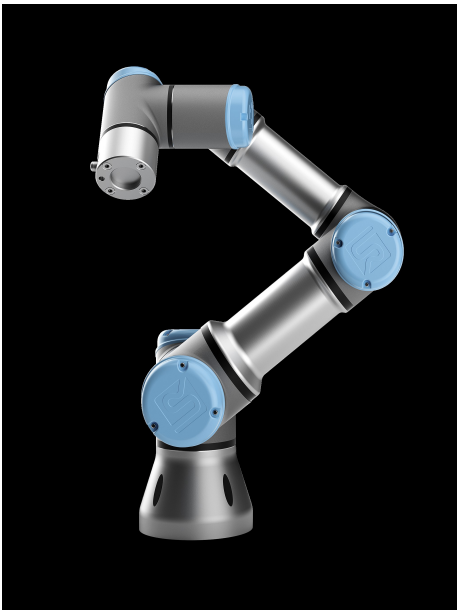


Figure 1.6 Universal robot's UR3 [Universal Robots, 2015].

1.5.2 Force sensors

Three different force sensors are tested in this thesis and a brief presentation of the sensors is done in this section.

JR3 The JR3 force/torque sensor is a standard industrial force/torque sensor, Figure 1.7. It is able to measure force along three axes and torque around each axis giving the possibility to measure force and torque in all directions. The advantages of this kind of sensor is that it is possible to do gravity compensation, Section 2.4, and reconstruct forces in directions outside its own coordinate system e.g., at the tool tip. The JR3 sensor uses strain gauge technology, Section 3.5 [JR3, 2016].

Figure 1.7 JR3 force/torque sensor.

Optoforce The Optoforce sensor used in this thesis, Figure 1.8, can measure force in three Cartesian directions. The sensor measures forces using infrared light by detecting small deformations in the shape of the surface of a halfsphere. A cross section of this sensor is showed in Figure 2.7. The sensor is delivered uncalibrated: a calibration is therefore needed, Section 3.5 [Optoforce, 2016].

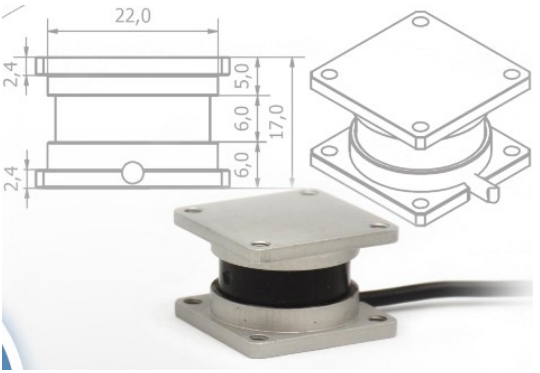


Figure 1.8 Optoforce force sensor [Optoforce, 2016].

Own developed sensor A sensor is also constructed by taking the strain gauges from an ordinary kitchen scale, Figure 1.9. This sensor is designed to be placed underneath a fingerprint sensor and is thereby only required to measure the force in one direction.

1.5.3 Finger print sensors and sensor modules

FPC5832 The FPC5832 is a small sensor module developed by Fingerprint Cards for testing and demo purposes. It reads a fingerprint sensor via an SPI channel and

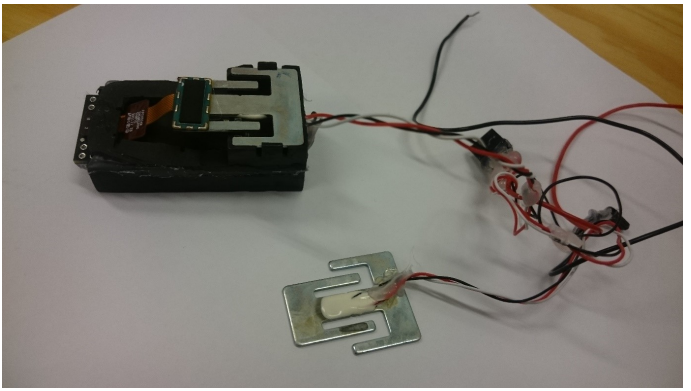


Figure 1.9 Own developed force sensor with an integrated fingerprint sensor and sensor module.

communicates to a computer via USB using FPC's sensorApi. FPC provides many different sensors with different designs and functionality. Some of these sensors are shown in Figure 1.10.

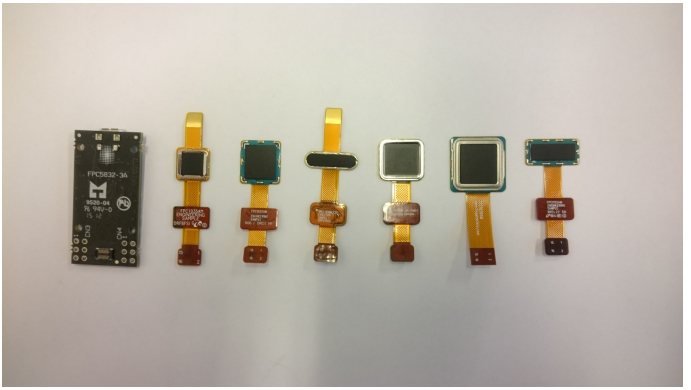


Figure 1.10 FPC5832 (left) and some of FPC's sensors.

Chapter 2

Theory

The theory in the following chapter is based on [Johansson, 2015], [Johansson et al., 2015], [Stolt, 2015], [Vougioukas, 2001], [Nikoleris, 2015], [Spong et al., 2006] and [Freidovich, 2014].

2.1 Robot frames and transformation matrices

Translation between different Cartesian coordinate frames is something extremely important when working with robots. This enables one to work with many objects, sensors etc. An example of this can be seen in Figure 2.1, displaying a robot, a work object and a camera. Using a transformation matrix one can e.g., get the position in camera coordinates and translate these coordinates to the robot's frame. To simplify this type of calculations one can use the homogeneous transform. This transform between frames is done using the following matrix:

$$\left[\begin{array}{ccc|c} & R_{3x3} & & T_{3x1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.1}$$

where R is a 3x3 rotation matrix and T is the transformation vector i.e., the vector describing distance in x-, y- and z-direction between the different frames.

A rotation matrix can be derived in many different ways. One of the most used approaches is to use the Euler angles. The Euler angles can be derived by first rotating the current Cartesian frame around its z-axis, then around the new frame's y-axis and lastly, rotating around the z-axis in the newest frame. This is referred to as Euler angles according to ZYZ-rotations. A visualization of this can be seen in Figure 2.2 and a mathematical representation of this can be derived and results in the rotation matrix 2.2, where s equals sine and c equals cosine.

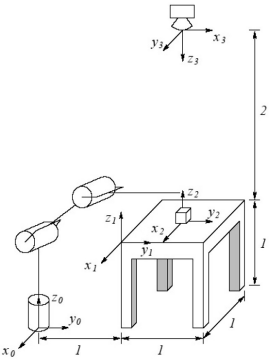


Figure 2.1 Example of different frames [Nikoleris, 2015].

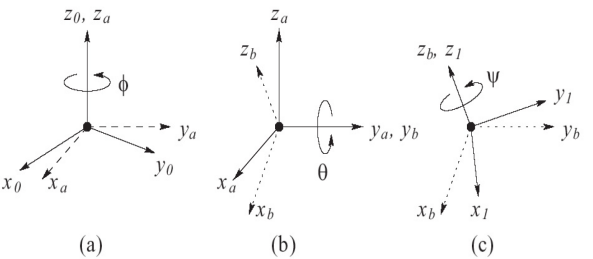


Figure 2.2 Euler angles (ZYZ) [Nikoleris, 2015].

$$\begin{bmatrix} c(\varphi)c(\theta)c(\psi)-s(\varphi)s(\psi) & -c(\varphi)c(\theta)s(\psi)-s(\varphi)c(\phi) & c(\varphi)s(\theta) \\ s(\varphi)c(\theta)c(\psi)+c(\varphi)s(\psi) & -s(\varphi)c(\theta)s(\psi)+c(\varphi)c(\psi) & s(\varphi)s(\theta) \\ -s(\theta)c(\psi) & s(\theta)s(\psi) & c(\theta)) \end{bmatrix} \quad (2.2)$$

Applying this information to the setup in Figure 2.1, one will derive the following translation matrices from the different frames to the robot frame.

$$H_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} H_2^0 = \begin{bmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & 1.5 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} H_3^0 = \begin{bmatrix} 0 & 1 & 0 & -0.5 \\ 1 & 0 & 0 & 1.5 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

where H_1^0 is the translation from the table, H_2^0 from the work object and H_3^0 from the camera. By multiplying one of these matrices with a vector containing the

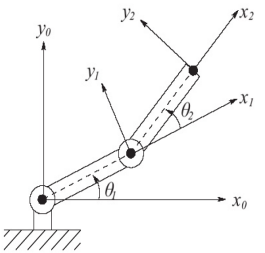
position of an object in the chosen frame and a one, $[x, y, z, 1]^T$, one will get the position of the object in the robot frame.

To get the translation from the robot base frame to one of the other frames, one can simply take the inverse of the translation from the other frames to robot base frame.

2.2 Kinematics and differential kinematics

2.2.1 Forward kinematics

Forward kinematics is a mathematical way of finding the position and orientation of the end effector of a robot given the joint angles. A simple example of how to implement this can be seen in Figure 2.3, which displays a two DOF manipulator and its translation matrix from base frame to end effector, where a_1 and a_2 are the lengths of the manipulator arms.



$$T_2^0 = A_1 A_2 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1 c_1 + a_2 c_{12} \\ s_{12} & c_{12} & 0 & a_1 s_1 + a_2 s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.3 Example of a two DOF manipulator and its forward kinematics expression [Nikoleris, 2015].

To be able to derive such equation and equations for more complex manipulators with arbitrary many DOF, different kinematic tools can be used. One of the most frequently used representations of robotic manipulators is the Denavit-Hartenberg convention. This method represents the kinematic chain between all robot joints by linking them together. This makes e.g., a six DOF manipulator nearly as easy to represent as a two DOF manipulator in Figure 2.3.

The Denavit-Hartenberg convention between two links is derived by entering the distance a_i, between the point where the x-axis of the new link intercepts the z-axis of the previous link and the new origin, the distance d_i, between the same point and the previous origin, the angle α_i, between the two z-axes around the new x-axis, and the angle, θ_i, between the two x-axes around the previous z-axis, Figure 2.4, into transformation matrix 2.4.

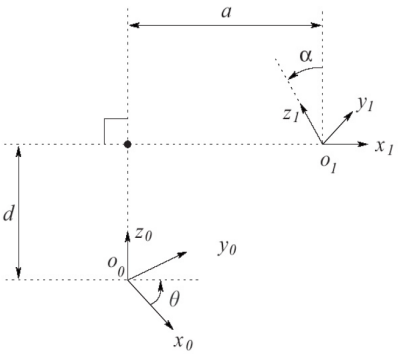


Figure 2.4 Denavit-Hartenberg convention [Nikoleris, 2015].

$$\begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

By multiplying all the matrices that link the different links together the chain is complete and one will have the position of the flange in the robot's base frame coordinates depending on the angle θ_i of each joint.

2.2.2 Inverse kinematics

Inverse kinematics is the opposite of previously mentioned forward kinematics i.e., a way to find the values for each joint given a position and orientation. The inverse kinematics is more difficult to compute than forward kinematics, but can for certain types of manipulators with simple geometry be calculated by an analytical or geometrical solution, although in most cases the inverse kinematics are calculated with a numerical method. By looking at Figure 2.5 one can come to the conclusion that there can be multiple solutions to an inverse kinematics problem and with some manipulators it is even possible to get an infinite number of solutions, Section 2.2.4.

If there exist multiple solutions to an inverse kinematics problem it is important to discard solutions which will result in large and rapid movements. The solution closest to the current position is therefore used and a continuous motion can be achieved.

2.2.3 Velocity kinematics

Velocities of a manipulator can be expressed in either Cartesian or in joint space. The correlation between joint velocities and Cartesian velocities is described by

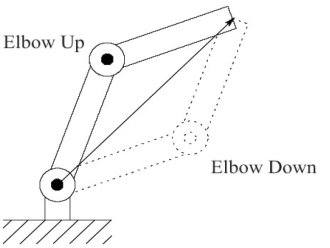


Figure 2.5 Example of multiple solutions to an inverse kinematics problem [Nikoleris, 2015].

Equation 2.5.

$$\begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = J(q)\dot{q} \tag{2.5}$$

where $\dot{p}$ is a vector with the Cartesian velocities of the manipulator in a given frame, ω is the angular velocity of the frame, J is the manipulator Jacobian and $\dot{q}$ is the vector containing the joint velocities.

2.2.4 Joint singularities

A problem when using a robot manipulator where it is possible to align two or more revolute joints around the same axis, is joint singularities. This enables the manipulator to reach the position with a desired orientation in infinitely many ways. If the robot is set to be in a singularity, the Jacobian matrix, Equation 2.5, becomes singular which makes it impossible to calculate the desired joint velocities. It is therefore important to make sure that the robot does not work in or close to joint singularities, since this can lead to unpredictable behavior.

2.3 Force sensors

In this thesis several force sensors are used and evaluated.

A force sensor can measure force in one or more directions and some sensors can also measure torque. Force sensors come in different shapes and sizes and with different designs for different purposes. Two different approaches to measure force are presented in this section.

2.3.1 Strain gauge sensors

Strain gauge sensors is the most common design in industrial applications. It works by attaching a strain foil to a metallic rod, Figure 2.6. This foil will then change

its resistivity depending on elongation and compression. To measure this small resistive change, a Wheatstone bridge with two strain gauges are often used, one for the measurements and one to compensate for temperature change. To amplify the signal from the Wheatstone bridge an instrumentation amplifier can be used. Further information about this can be found in Section 3.5.2. To get a sensor that can measure the force in each Cartesian direction and torque around each axis one can use multiple strain gauges and then through force kinematics calculate all the forces and torques.

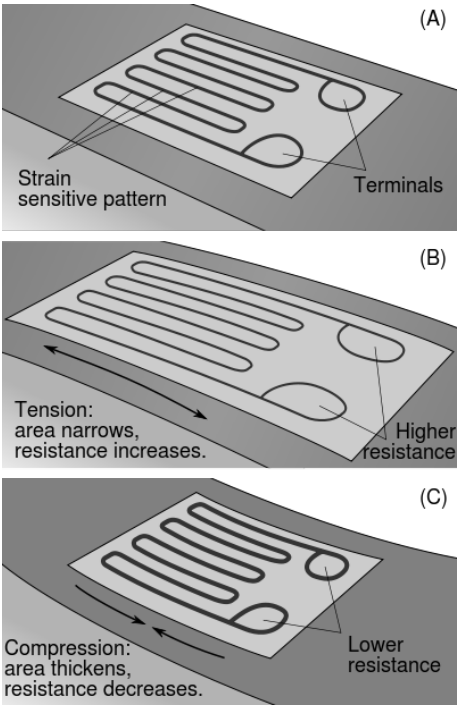


Figure 2.6 Elongation and compression of a strain gauge sensor [*Strain gauge*].

2.3.2 Optical deformation sensors

For applications where small forces and high accuracy is important one can use an optical deformation sensor. As the name suggests it measures the deformation of a rubber material using infrared light, Figure 2.7. This method can detect the smallest of changes in the rubber material and measure contact forces as small as a feather touching the surface (0.01 N). This type of sensor does also handle high overloads well, due to the fact that the sensors are not in contact with the deformation material. However there are some disadvantages as well and they are discussed in Section 5.4.

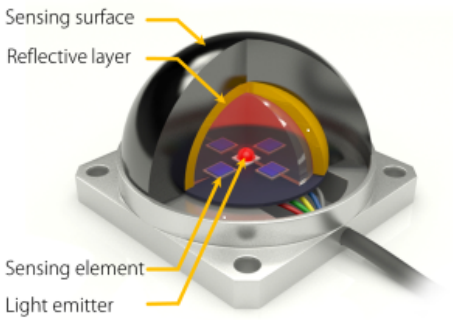


Figure 2.7 Structure of a optical deformation sensor [Optoforce, 2016].

2.4 Force measurements

The theory in Section 2.4 is based on [Vougioukas, 2001] and describes a method for gravity compensation of a 6 DOF force/torque sensor. In [Vougioukas, 2001] one can also read about the performance and evaluation of this method. In Appendix A.2 an implementation of the method, written in Matlab, is presented.

2.4.1 Six DOF force/torque sensor

A six DOF force/torque sensor (FTS) generates an output vector describing force and torque in each Cartesian direction, Vector 2.6.

$$F_s = [F_x, F_y, F_z, \tau_x, \tau_y, \tau_z]^T \tag{2.6}$$

The sensor measurements do unfortunately not only consist of the desired contact force F_c. To get a more accurate representation of the force measured by a force sensor one can represent this force by Equation 2.7.

$$F = F_b + F_g + F_c + F_n + F_v \tag{2.7}$$

where F_b is the bias force, F_g is the force due to gravity, F_c is the contact force, F_n is the noise and F_v is the output due to vibrations or accelerations of mass in the chain "outside the location of the sensor". F_n can be handled using a digital filter, F_v by giving the system time to stop vibrate before reading values, F_b by e.g., jogging the robot to positions where the force in a certain direction is known to be zero and compensate with this force and F_g by gravity compensation.

2.4.2 Bias

The idea for the bias estimation is quite simple. The estimation F_b equals the error when the force in a Cartesian direction should be zero, i.e., when the FTS Cartesian

z-direction is parallel to the gravity vector (z-direction in world coordinates). This would, without a bias, give:

$$F_b^x = F_{xm} = F_b^y = F_{ym} = 0 \tag{2.8}$$

The bias terms are thereby given by the following equations:

$$F_b^x = F_{xm}, \quad F_b^y = F_{ym} \tag{2.9}$$

where F_{xm} and F_{ym} is the measured force in x- and y-direction. A rotation around the (in this case) z-axis can be performed to get a dataset. This procedure needs to be done for each of the Cartesian directions of the FTS. This generates two bias estimation data-sets in each of the Cartesian directions and an average of these measurements can be calculated and used as the F_b.

The torque bias can be calculated using the same methodology but utilizing a different set of equations i.e., the torque τ_b^x is calculated by:

$$\tau_b^x = \tau_{xm} - (r_y * F_{zm} - r_z * F_{ym}) \tag{2.10}$$

where $r = [r_x, r_y, r_z]$ is the unknown center of mass, τ_{xm} is the measured x-torque and F_{zm} and F_{ym} is the measured force in z- and y-direction. A second approach that can be implemented is to jog the FTS to two different rotations where the true forces are known to be $+F$ and $-F$. The reading of the sensors will then become $+F + F_b$ and $-F + F_b$. Adding these two measurements and dividing by two will thereby give the bias estimate.

The torque bias will in similar fashion be given by Equation 2.11.

$$\begin{aligned} \tau_{xm} &= \tau_b^x + (r_y * F_{zm} - r_z * F_{ym}) \\ \tau_{xm} &= \tau_b^x + (-r_y * F_{zm} + r_z * F_{ym}) \end{aligned} \tag{2.11}$$

where $r = [r_x, r_y, r_z]$ is the unknown center of mass, τ_{xm} is the measured x-torque and F_{zm}, F_{ym} is the measured force in z-and y-direction, respectively. The second approach can be done with multiple pairs to get a better estimate. The advantage the second approach gives is that one does not need to know the exact transformation between FTS frame and world frame.

2.4.3 Estimation of mass

The gravity vector in world coordinates can be expressed as $g_I = [0, 0, -g]^T$ where g is the local gravity acceleration. If the FTS frame rotation for the i:th measurement is R_i, the gravity vector can be expressed as $g_{si} = R_i^T * g_I$. Using the force bias estimation, the measured force values and the equation $F_i = m * g_{si}$, where m is the mass, F_i the force measurements stacked into vector and G is the gravity vector stacked into a vector. One can get a least square estimation of m from Equation 2.12.

$$\hat{m} = \frac{G^T F}{G^T G} \tag{2.12}$$

2.4.4 Estimation center of mass

The center of mass for FTS and tool is derived from the torque measurements. If one denotes the center of mass $r = [r_x, r_y, r_z]$, where r equals the distance between FTS frame to center of mass in x-, y- and z-direction respectively, one can use Equation 2.13.

$$\tau_i = mr \times g_{si} \tag{2.13}$$

The equation can be rewritten as $\tau_i = mA_i r$, where τ_i is the measured torque of the i:th rotation, m is the mass and A_i is the matrix 2.15 of the i:th rotation. By stacking the K torque measurements and the A_i matrices into a vector $T \in \mathbb{R}^{3K}$ and a matrix $A \in \mathbb{R}^{3K\times 3}$, one can solve for $\hat{r}$ using Equation 2.14.

$$\hat{r} = \frac{1}{\hat{m}}(A^T A)^{-1} A^T T \tag{2.14}$$

$$A_i = \begin{bmatrix} 0 & g_{iz} & -g_{iy} \\ -g_{iz} & 0 & g_{ix} \\ g_{iy} & -g_{ix} & 0 \end{bmatrix} \tag{2.15}$$

2.4.5 Gravity compensation

Using the previously derived values of $\hat{m}$ and $\hat{r}$ one can calculate the gravity compensation by the equation:

$$\hat{F_g} = \begin{bmatrix} \hat{m} R^T g_I \\ \hat{m}\hat{r} \times (R^T g_I) \end{bmatrix} \tag{2.16}$$

Where R defines the FTS frames current rotation in the robot frame.

2.4.6 Force torque sensors with less DOF

For a sensor that is not able to measure torque the estimation of center of mass is not possible using this method. The error this generates can be minimized if the distance between the origin of the force frame and the center of mass is as small as possible. To compensate for rotation, one can remove the offset when the experiment starts and then do this every time the tool rotates, as long as there is no contact force. To get the nominal force between the tool and a contact point one can simply use Equation 2.17 below (this method will also work with FTS).

$$F_{tot} = \sqrt{F_x^2 + F_y^2 + F_z^2} \tag{2.17}$$

where F_{tot} is the force normal to the surface.

2.5 Controllers

In this thesis the main goal of the controller algorithm is to make it easy for the user to implement it on any robot without a detailed model of the robot. To make this possible it is necessary to use external sensing, which in this case was an external force sensor. The environment the robot is working with is often stiff and in this case sensitive to high forces. This makes it a non-trivial task to design the force controller. The two main approaches of this problem is presented below.

2.5.1 Direct force control

The first and simplest approach is to use an external feedback loop that controls the force error between the measured force and a force reference. This can be done by implementing a PID-controller or a simpler PI-controller. The advantages a PI-controller has in this implementation is that one does not have a derivative gain that is sensitive to high frequency disturbance. The transfer function of a PI-controller and PID-controller, in parallel form and in Laplace domain, is described by Equations 2.18 and 2.19.

$$G_{PI} = P + I\frac{1}{s} \tag{2.18}$$

$$G_{PID} = P + I\frac{1}{s} + Ds \tag{2.19}$$

where P is the proportional gain, I is the integral gain and D is the derivative gain.

For practical implementation anti-windup and derivative filtering is often necessary. Anti-windup is used to prevent the integral part from accumulating when the output from the controller to the controlled system is saturated. This can otherwise lead to a dramatic decrease of the control performance. The derivative filter is added to prevent the derivative part from regulating high-frequency noise, this will otherwise generate a high derivative-gain amplification of the noise and possibly an unstable system. By adding a first order low-pass filter to the derivative part of the PID-algorithm in Equation 2.19, one can describe the PID-controller by Equation 2.20, where N denotes the location of the pole of the derivative filter (the pole is place at -N on the real axis) [Mathworks, 2016c], [Glad, T. & Ljung, L., 2003].

$$G_{PID} = P + I\frac{1}{s} + D\frac{Ns}{s+N} \tag{2.20}$$

Another controller that can be used as a direct force controller is a Linear-Quadratic-Gaussian controller (LQG-controller). LQG-control is one of the most fundamental ways of deriving an optimal controller. The benefits of the LQG-controller is that it allows a trade off in controller performance and control effort and takes into consideration process disturbances and measurement noise. A LQG-controller is derived from a state-space model and consists of an optimal

state-feedback gain (L) and a Kalman state estimator gain (K). The controller minimizes the cost function in Equation 2.21 [Mathworks, 2016b], [Glad, T. & Ljung, L., 2003].

$$\begin{aligned} J &= E\{\lim_{\tau\to\infty}\int_0^{\tau}[x^T,u^T]Q_{xu}\begin{bmatrix}x\\u\end{bmatrix}dt\} \\ &\text{subject to the state space equation} \\ \dot{x} &= Ax+Bu+w \\ y &= Cx+Du+v \end{aligned} \tag{2.21}$$

where the process noise w and measurement noise v are Gaussian white noises with covariance Q_{wv}. The Q_{xu} and Q_{wv} are weighting matrices to be tuned.

To be sure that the output signal tracks the reference signal without error one can add an integrator part which creates the cost function 2.22, which is to be minimized and where Q_i is a weighting matrix for the integral part.

$$J = E\{\lim_{\tau\to\infty}\int_0^{\tau}([x^T,u^T]Q_{xu}\begin{bmatrix}x\\u\end{bmatrix}+x_i^TQ_ix_i)dt\} \tag{2.22}$$

Additional choices for a direct force controllers include a Model predictive controller (MPC-controller). An MPC-controller solves, at each sample a finite horizon optimal control problem. The first control value given by this solution is then applied to the process and the other solutions is discarded. This procedure is then repeated for each sample and the prediction horizon is shifted [Åkesson, 2006].

2.5.2 Impedance control

The second approach is to use an Impedance controller [Hogan, 1985]. The goal of this controller is to make the robot behave like a mass-spring-damped system. A control law that makes this possible is derived from Equation 2.23.

$$F = M\ddot{x}+D\dot{x}+Kx \tag{2.23}$$

where M is the mass, K is the spring constant, D is the damping constant, x is the position from where contact was made, $\dot{x}$ is the velocity, $\ddot{x}$ is the acceleration and F is the resulting force. By rewriting the equation one can derive the control law of an Impedance controller, where the control signal is an acceleration, Equation 2.24.

$$\ddot{x} = \frac{1}{M}(-D\dot{x}-Kx+F) \tag{2.24}$$

To be able to control the robot, this control signal is integrated once to get a velocity reference in tool frame and twice to get the desired position reference. The integration can be done using a discrete time integrator. This generates a sampled velocity and a position reference signal.

2.5.3 Pole placement

Pole placement design is a way to get the desired behavior of a system. Depending on where the closed loop poles of the system in the frequency domain are placed, different behavior will be established. If one looks at the different pole placements in Figure 2.8 and compare them to their respective step response one can see how the pole placement effects the system. To summarize this, poles in the right half plane give a unstable system, poles in the left half plane (LHP) a stable system. Poles on the imaginary axis give an oscillating response, poles on the real axis in the LHP will not give an overshoot on the response but has a slower rise time. The further away from the origin the poles are placed in the LHP the faster the response. The bigger angel from the real axis in the LHP the more oscillations but faster rise time. Add to this the fact that if the closed loop poles are far away from the open loop poles the system demands a high control effort and if the system has a fast response (i.e., poles with large negative value) it is more sensitive to noise and delays.

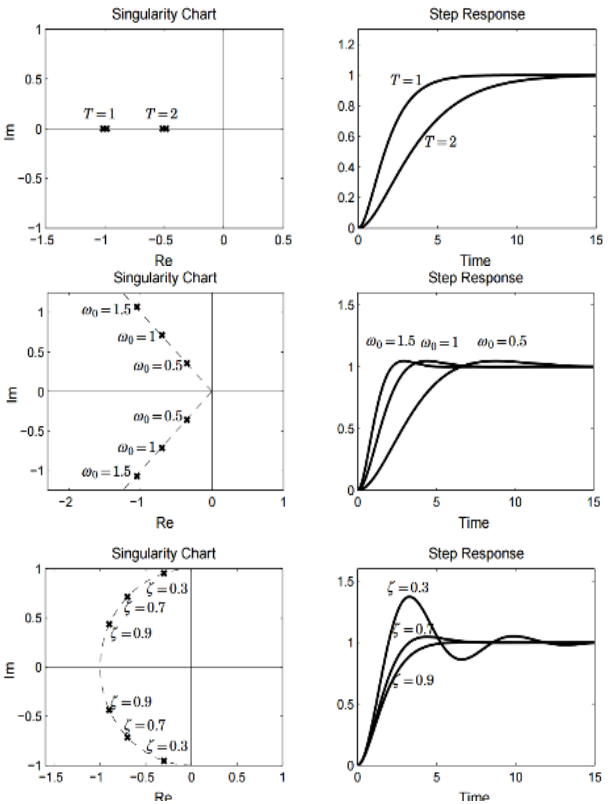


Figure 2.8 Pole placements and step responses [Hägglund, 2013].

2.6 Modeling

2.6.1 Contact model

The model of the contact force between the end effector and environment is based on a spring-damper design and the force applied to the environment is given by Equation 2.25.

$$F = K_e \Delta x + D_e \dot{x} \tag{2.25}$$

where K_e is the spring constant for the contact surface, D_e is the damping of the surface, Δx is the distance from unloaded surface to the current position and $\dot{x}$ is the current velocity. The transfer function in Laplace domain of this model from force (F) to position (Δx) can be described according to Equation 2.26.

$$X(s) = \frac{F}{(D_e s + K_e)} \tag{2.26}$$

To get an even more accurate representation of the system some noise and low pass filters can be added e.g., one filter on the velocity input of the model, one at the position input and one at the force output.

2.6.2 System identification

The model above can be used as a grey-box model, which is a model where one acquires different constants by an identification algorithm e.g., via RLS-algorithm, Equation 2.27. Another approach to model a system is by creating a so called black box model. The thing that differs the black box model from the grey box model is that instead of starting with a physical model one only considers a model that is mathematically representative of the system. The advantages of this method are that it requires less knowledge of the system and thereby, one is able to model more complex dynamics without knowing much about the dynamics.

Closed loop identification To model a robot or an unstable process one generally needs some kind of closed loop control in order to identify the model. Generally, the test signal that one would like to apply to the process input if it were a stable process is applied as reference to the closed-loop system. There are mainly two ways of performing this kind of identification; either the identification is done by deriving a model between the control signal and the process output, this is called direct identification, or the exciting signal (the reference signal to the controller) is used as the input to the system and the output is the output from system. This is called indirect identification. The controller used for the indirect identification is then compensated for to derive a model just containing the process dynamics.

There are problems equipped with closed loop identification; if too low excitation is used, only the controller dynamics might be recorded and the identified model might end up being an inverse of the controller. Another problem is that the

bandwidth of the controller affects in what frequency range identification can be made, i.e., if the process has higher bandwidth than the controller, only process dynamics within the bandwidth of the controller may accurately be identified.

Detecting non-linearity In the case of a non-linear process, it is useful to find a working range where the process can be approximated with linear models, since tools for linear models are more developed than for the non-linear ones. One way of doing this is to study the coherence spectrum for the collected input-output data. Frequencies with high coherence may be approximated using linear models, whereas frequencies with low coherence need more complex models.

Validation When a model has been identified, it is important to verify that the model actually can represent the real process. One way of verifying this is to study the residual sequence. The residual analysis gives the correlation residual for the outputs and the cross correlation between the inputs and the output residuals. This analysis is likely to be the best validation of how good the produced model is, with the exception of trying a controller based on the model. Another indication of the accuracy of the model is to compare the validation data to the data that the model generates.

2.6.3 Adaptive algorithm

To be able to interact with different materials and surfaces without the need to re-calibrate the controller an adaptive algorithm can be used. One such algorithm is Recursive Least Squares (RLS) algorithm. It estimates an online least squares solution that fit parameters to a predefined model according to the algorithm below [Johansson, 2015].

$$y = \phi^T \theta, \quad \begin{cases} \hat{\theta}_k = \hat{\theta}_{k-1} + P_k \phi_k \varepsilon_k \\ \varepsilon_k = y_k - \phi_k^T \hat{\theta}_{k-1} \\ P_k = P_{k-1} - \frac{P_{k-1}\phi_k \phi_k^T P_{k-1}}{1+\phi_k^T P_{k-1} \phi_k} \end{cases} \tag{2.27}$$

where θ contains the parameters to be estimated, $\hat{\theta}_k$ contains the parameter estimations, ϕ_k is the regressor, y the output of the system given the input ϕ, ε_k the prediction error and the matrix P_k constitutes, except for a factor σ^2, an estimate of the parameter covariance at recursion number k.

2.6.4 Moving average filter

A moving average filter is a form of a finite impulse response (FIR) low pass filter that continuously takes a number of samples (a window) and outputs an average of these samples. There are many different implementations of the moving average filter where one can implement different weighting of the samples. An ordinary

moving average filter without weighting can be described by Equation 2.28.

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i-j] \tag{2.28}$$

where $y[i]$ is the i:th output, M is the size of the window and $x[i-j]$ is the $i-j$:th input.

Chapter 3

Methods

3.1 Controllers

3.1.1 Contact model

The contact model described in Section 2.6.1, is used to represent the contact force. The unknown variable is calculated using RLS described in Section 2.6.3. This results in Algorithm 3.1.

$$y = \phi^T \theta, \quad \begin{cases} y = F \\ \phi = [x \quad \dot{x} \quad 1]^T \\ \theta = [K_e \quad D_e \quad K_e x_e]^T, \end{cases} \quad \begin{cases} \hat{\theta}_k = \hat{\theta}_{k-1} + P_k \phi_k \varepsilon_k \\ \varepsilon_k = y_k - \phi_k^T \hat{\theta}_{k-1} \\ P_k = P_{k-1} - \frac{P_{k-1}\phi_k \phi_k^T P_{k-1}}{1+\phi_k^T P_{k-1}\phi_k} \end{cases} \tag{3.1}$$

where the product $K_e x_e$ is seen as a separate model parameter making the model linear.

3.1.2 Impedance controller

One of the controllers used in this thesis is an Impedance controller, Section 2.5.2. The algorithm is slightly modified to be able to act on a force reference error, instead of just the measured force. Moreover the position reference is removed due to the fact that the position is not important, which leads to the control law:

$$\ddot{x} = \frac{1}{M}(F - F_r - D\dot{x}) \tag{3.2}$$

where $\ddot{x}$ is the desired acceleration, $\dot{x}$ is the velocity, F the measured force, F_r is the force reference, D is representative of the damping and M is representative of the mass. Furthermore the pole placement design is implemented, Section 2.5.3. This is done by using Equations 2.26 and 3.2. The assumption of an ideal velocity controlled robot is made and the equation is transformed to frequency domain by Laplace transform. This gives:

$$s^2X(s) = \frac{1}{M}(F(s) - F_r(s) - DsX(s)), \quad where \quad X(s) = \frac{F}{(D_e s + K_e)} \tag{3.3}$$

Rewriting the equation results in the transfer function from F_r to F given by equation:

$$F(s) = \frac{(\frac{-D_e}{M}s - \frac{K_e}{M})}{s^2 + \frac{D+D_e}{M}s + \frac{K_e}{M}} F_r(s) \tag{3.4}$$

Using pole placement design of a second order linear time-invariant dynamic system where the poles are placed according to Figure 3.1, gives the denominator polynomial in Equation 3.5.

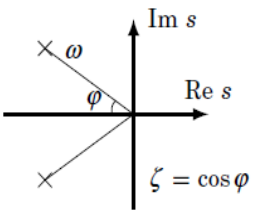


Figure 3.1 Pole placements design [Hägglund, 2013].

$$s^2 + 2\zeta\omega s + \omega^2 \tag{3.5}$$

where ω is the distance to the origin from the poles and ζ is $cos(\varphi)$ where φ is the angle to the poles.

Relating this polynomial with the denominator in the transfer function from F_r to F, gives the force control parameters in Equation 3.6.

$$M = \frac{K_e}{\omega^2}, \quad D = \frac{2\zeta K_e}{\omega} - D_e \tag{3.6}$$

where the RLS estimates of K_e and D_e can be used.

A desirable behavior of the system is to get as small overshoot as possible, which leads to the conclusion that ζ should be equal to one i.e., place the poles on the real axis. On the other hand ω, needs to be decided by testing, so noise and delays can be accounted for.

3.1.3 PID

The PID-controller used in the simulations is derived using the PID-tuner in Matlab's Control Systems Toolbox. The controller is a discrete PID-controller implemented by discretization of Equation 2.20, using the forward Euler method and tuned to be as fast as possible without getting an overshoot, giving the parameters P=1, I=0.1, D=0.05 and N=106. When implementing this controller on the real process all the parameters had to be retuned manually, giving the new parameters P=0.4, I=0.15, D=1 and N=1. The result of the final tuning is presented in Section 4.3.3.

3.1.4 LQG

The LQG is derived from the model using the same control systems toolbox in Matlab [Mathworks, 2016b]. By entering the command $lqg(model, QXU, QWV, QI, '1dof')$ an LQG-controller is derived which minimizes the cost function given by Equation 2.21 subject to the model derived by system identification, Section 2.6.2. The controller is then retuned by trial and error, due to the fact that the model does not have a physical representation. The result of the final tuning is presented in Section 3.3.2 and 3.4.

3.1.5 MPC

An MPC-controller is derived using MPCtools. MPCtools is developed by Johan Åkesson, at LTH [Åkesson, 2006]. The prediction horizon is set to 20 samples and the input horizon is set to 10 samples. The controller is then retuned by trial and error, due to the fact that the model does not have a physical representation. The result of the final tuning is presented in Section 3.3.2 and 3.4.

3.2 System identification

A system identification is done in closed loop, by running sensor program 3.1 (Section 4.3.2) three times, acquiring six different data sets representing the velocity of the robot when in contact with the surface as input and the force measurements as the output for each of the three times the sensor program 3.1 is run. The first two data input-output sets are used for the identification and the last as validation data set. An ARMAX model is derived by iterating different orders of the polynomials. The models with the highest fitting rate are then further validated to get as good of a model as possible. The algorithm can be seen in Appendix A.1.

3.3 Simulations

3.3.1 Simulation of Impedance controller

A simulation of the system is implemented in Matlab/Simulink, Figure 3.2. Thereby stability, convergence and effect of delays on the step response are evaluated for different parameter settings.

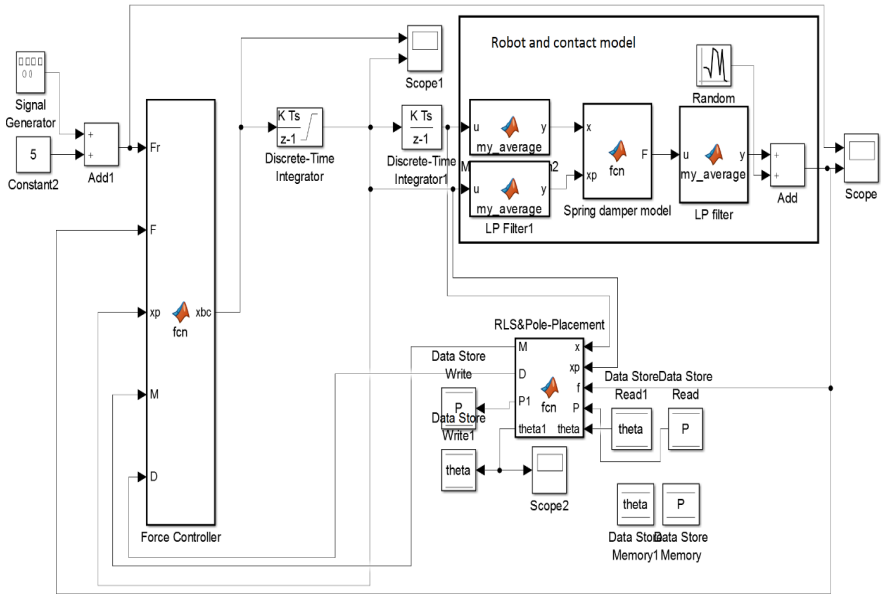


Figure 3.2 Simulation model for the Impedance controller.

A simulation of the system using a response to a square wave with a frequency of 0.1 Hz and amplitude of 5 N can be seen in Figure 3.3. The system has the following configuration: the poles placed at -5 on the real axis, the moving average filters with a window of 50 samples and a noise with a variance of 0.01 N. In Figure 3.4 one can observe the convergence of the RLS parameters. The parameters in the model are set to: $K = 150$, $D = 5$ and $K_e x_e = 0$, which are not obtained. This implies that noise and delays affect the modeling in a negative way, as expected. The response and convergence of the RLS-parameters without filters can be seen in Figure 3.5 and 3.6. The oscillations before and the overshoot in the first step in Figure 3.3, is caused by the fact that the RLS-parameters have not converged sufficiently yet.

The Impedance controller is also tested with the model derived during the system identification and resulted in the step responses in Figure 3.7.

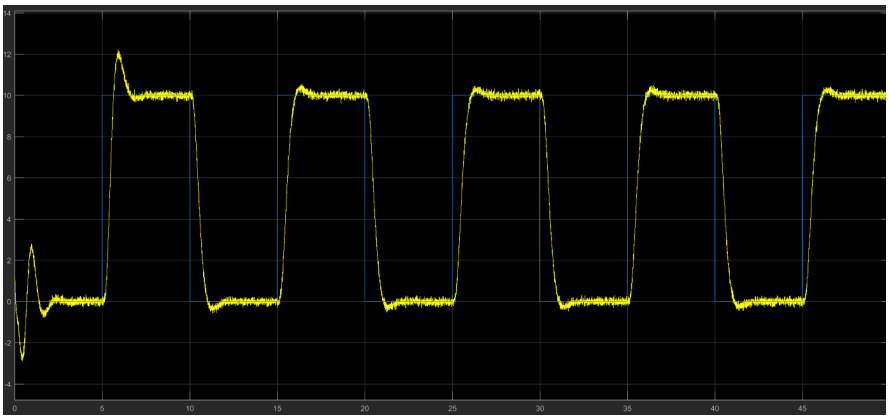


Figure 3.3 Step responses of the Impedance controller using a model with low pass filters. The data is plotted with force (N) on the y-axis and time (s) on the x-axis.

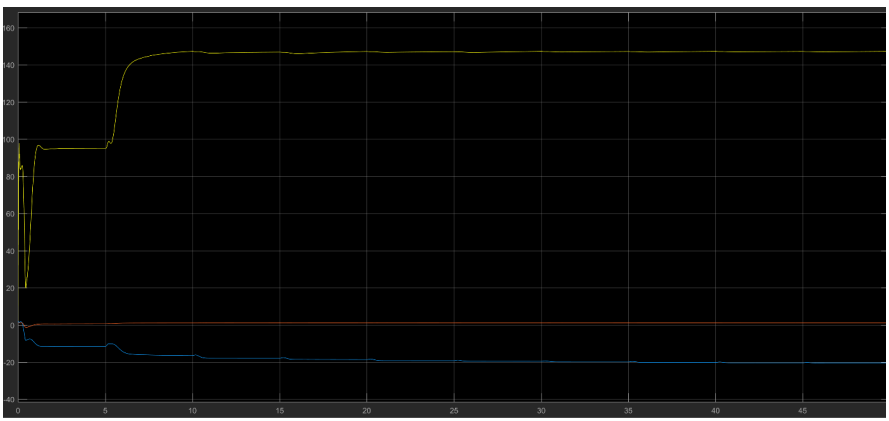


Figure 3.4 The convergence of the RLS-parameters using a model with low pass filters.

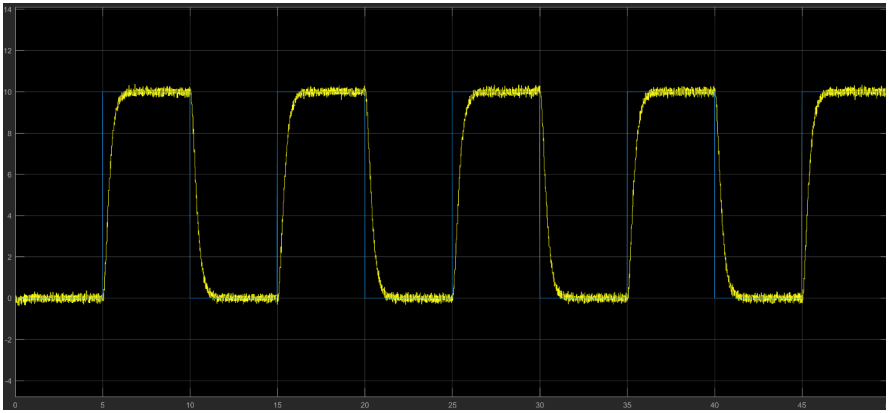


Figure 3.5 Step responses of the Impedance controller using a model without low pass filters. The data is plotted with force (N) on the y-axis and time (s) on the x-axis.

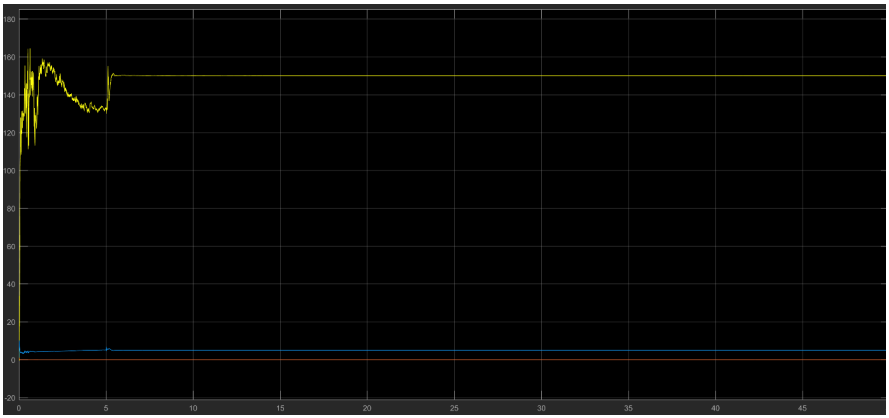


Figure 3.6 The convergence of the RLS-parameters using a model without low pass filters.

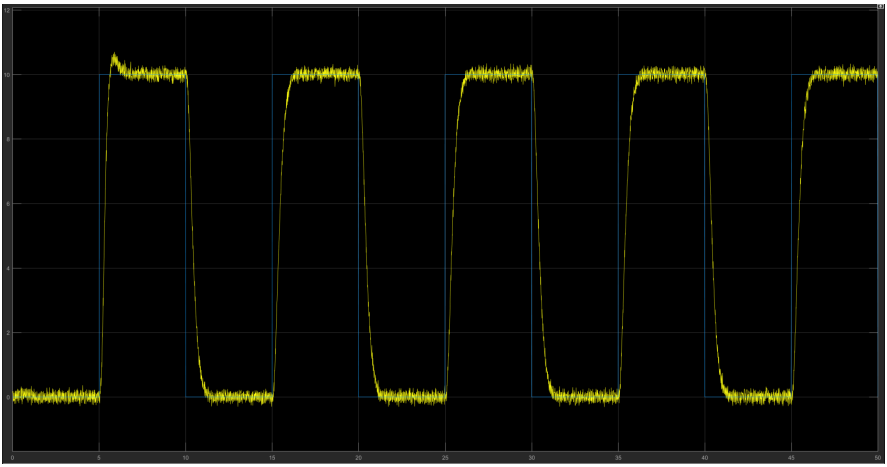


Figure 3.7 Step responses of the Impedance controller using a model derived from system identification. The data is plotted with force (N) on the y-axis and time (s) on the x-axis.

3.3.2 Simulation of direct force controllers

Using the model derived from the system identification, Section 3.2, and the PID-, LQG- and MPC-controller, further simulations are done. The step responses can be seen in Figure 3.8, where the blue curve represents the MPC-controller and the red curve the LQG-controller. The response of the PID-controller is plotted in Figure 3.9.

The fact that the model does not have a physical representation makes it hard to tune the LQG- and the MPC-controller. This together with the result of the simulation with the tuned PID in Figure 3.9, which is better than both the MPC and the LQG, evolved in a decision to just implement the PID-controller on the real process.

3.4 Summary of simulations

In Table 3.1, one can see the performance of the different controllers in simulation. The "Rise time" is the time it takes for the controller to regulate the output error of the system, after a step change, see details in Table 3.1 below. The "Overshoot" is how much the output signal of the system overshoots the reference after a step change and the "Settling time" is the time it takes for the controller to get back inside a region of an absolute error after this step change.

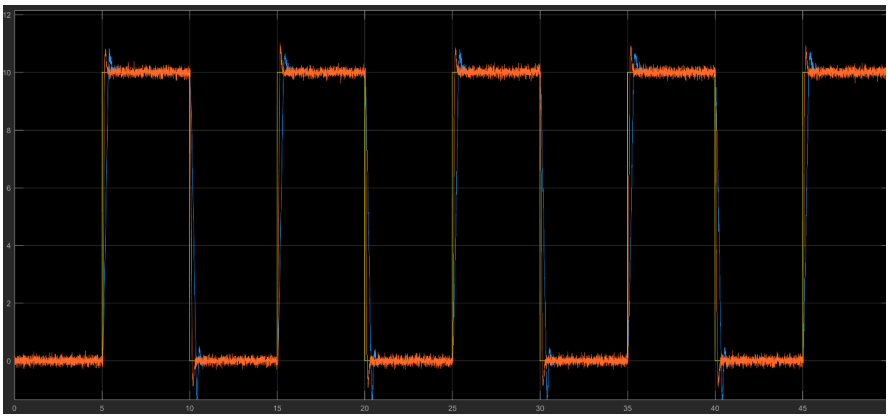


Figure 3.8 Step responses of the LQG and MPC controllers using a model derived from system identification. The data is plotted with force (N) on the y-axis and time (s) on the x-axis.

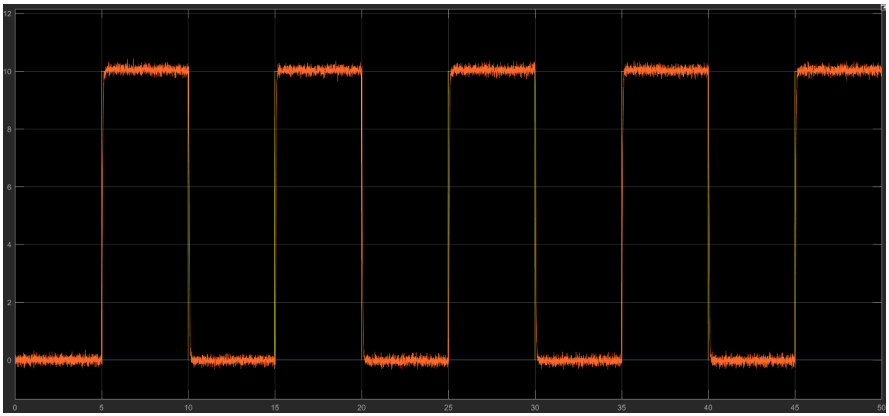


Figure 3.9 Step responses of the PID controller using a model derived from system identification. The data is plotted with force (N) on the y-axis and time (s) on the x-axis.

Controller	Rise time 10-90 %	Rise time 5-95%	Overshoot	Settling time 5%	Settling time 2%
PID-controller	0.07 s	0.09 s	<2 %	0.12 s	0.15 s
Adaptive con-troller	0.65 s	0.88 s	<2%	0.93 s	1.15 s
LQG-controller	0.07 s	0.085 s	9 %	0.24 s	0.3 s
MPC-controller	0.25 s	0.29 s	8 %	0.51 s	0.6 s

Table 3.1 The performance of the different controllers in simulation.

3.5 Implementation of sensors

3.5.1 Optoforce sensor

The calibration is done by placing the sensor on a kitchen scale and zeroing the scale. By applying different weights and reading both the scale and the sensor values, two data sets are created. By multiplying the data values given by the scale with (g/1000) where g is the acceleration of gravity (9.81 m/s^2), one gets a representation in N. The sensor data is then fitted to the scale data by a least square sense and thereby a polynomial is derived. How this polynomial fits to the data can be seen in Figure 3.10.

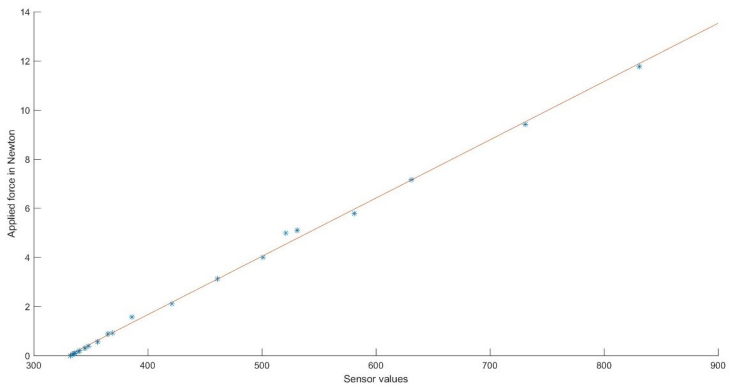


Figure 3.10 Calibration of Optoforce sensor.

3.5.2 Own developed force sensor

To compensate for temperature changes and other variances two strain gauges are used, one to compensate for temperature changes and one for measurements. These strain gauges are connected to create a Wheatstone bridge, Figure 3.11. A Wheatstone bridge can be used to measure small changes in resistance with a very high accuracy, this makes it perfect for this setup. The voltage difference between V_0- and V_0+ can be derived and is given by Equation 3.7. The voltage when applying a force to the strain gauge does only vary a couple of millivolts. To get an accurate reading one has to amplify this signal to a desired level. A gain of over 1000 was sufficient to get a good digital reading from an analog to digital converter. The amplification is done with an Instrumental amplifier built from three operational amplifiers. The design of this circuit is done according to Figure 3.12. This gives a theoretical amplification given by Equation 3.8. Some things to consider when implementing this design is to tune it to the correct working range. This sensor is constructed to work in a range from 0 N to 11 N and will thereby give the most accurate reading possible.

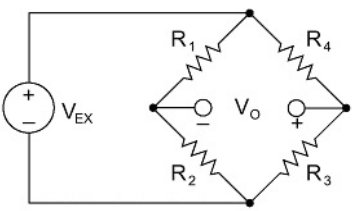


Figure 3.11 Wheatstone bridge [*How is Temperature Affecting Your Strain Measurments Accuracy?*]

$$V_0 = (\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2})V_{EX} \tag{3.7}$$

The first part of the amplifier is the gain stage, which is where most of the amplification takes place. The second stage is the differential stage where the difference of the two outputs from the Wheatstone bridge is derived together with some additional amplification. An Industrial amplifier is to prefer in this application due to the fact that it has a very high common mode rejection ratio, low noise amplification, eliminates the need for impedance matching, low DC offset, low drift, very high input impedance and very high loop gain.

$$V_{out} = (1+2\frac{R_1}{R_{gain}})\frac{R_3}{R_2}(V_2 - V_1) \tag{3.8}$$

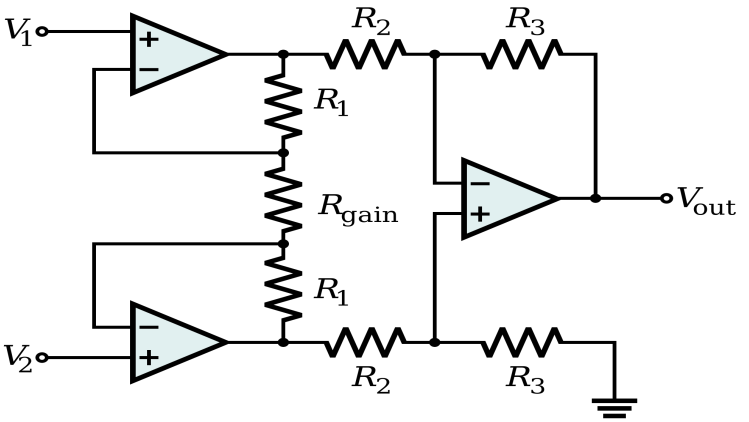


Figure 3.12 Instrument amplifier [*Instrumentation amplifier*].

The analog to digital conversion is done by an Arduino Uno board. The Arduino board has a 10-bit 5 V analog to digital converter that is sampling at a rate of 10000 times a second which converts the analog signal to a digital by representing the voltage in integer values between 0 and 1023, i.e., 0 V is represented by 0 and 5 V is represented by 1023. The Arduino also handles the filtering and conversion from voltage level to newton. The filtering is done by an ordinary moving average filter, Section 2.6.4, with a sampling window of 50 samples. The conversion from voltage level to newton is done by calibrating the sensor and mapping the voltage level to a force level. This is done by gathering data with a voltage level and a corresponding force level. The force is measured by applying a weight to the sensor when it is placed on a kitchen scale. The value on the scale (weight of the object in grams) is then converted to newton by multiplying with g/1000 where g is the acceleration of gravity (9.81 m/s^2). When sufficiently many values are taken a polynomial representation of the first degree is derived by iterating a least square solution. The results of this can be seen in Figure 3.13.

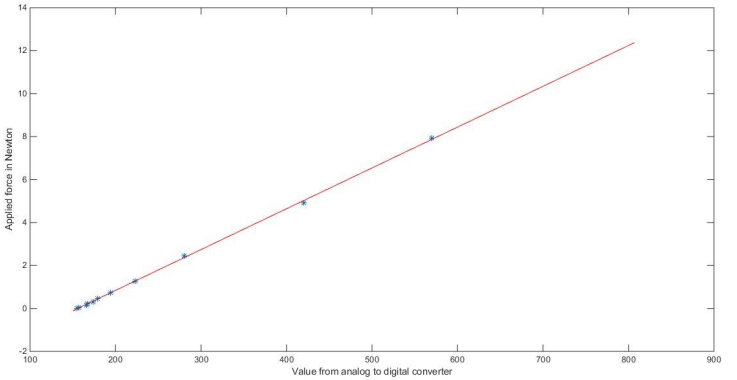


Figure 3.13 Calibration of own developed sensor.

Chapter 4

Results

The results are presented by first giving an overview of the complete system and then by a deeper presentation of each part of the structure. The complete system with the GUI program, ABB Rapid program, Opcom, sensor program 3.2 and force sensor program 3.0 is displayed in Figure 4.1.

4.1 Robot control GUI

The developed user interface consists of the graphical user interface (GUI) showed in Figure 4.2 and is a Windows .Net application programmed in *C#*. This interface enables the user easy control over the test functionality described below.

1. A list of robot controllers that is available in the network.
2. Shows the selected robot.
3. Lists all the connected FPC sensors.
4. Shows the order in which the sensors will be tested.
5. Displays a picture taken by a sensor. One is also able to toggle between pictures taken under the test.
6. Shows the picture index of the current picture and the force applied when it was taken.
7. Clears the list that shows the order of which the sensors will be tested.
8. Clears all the images.
9. Updates the list of all the connected FPC sensors.

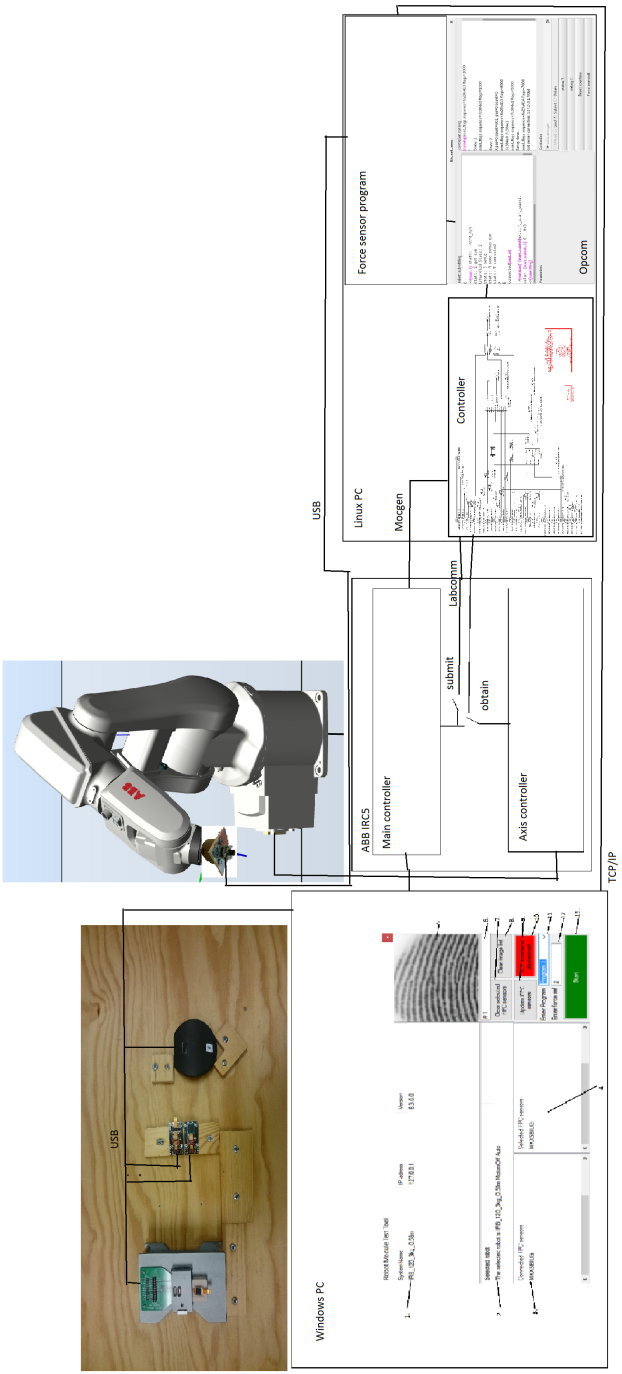


Figure 4.1 Figure of the complete system.

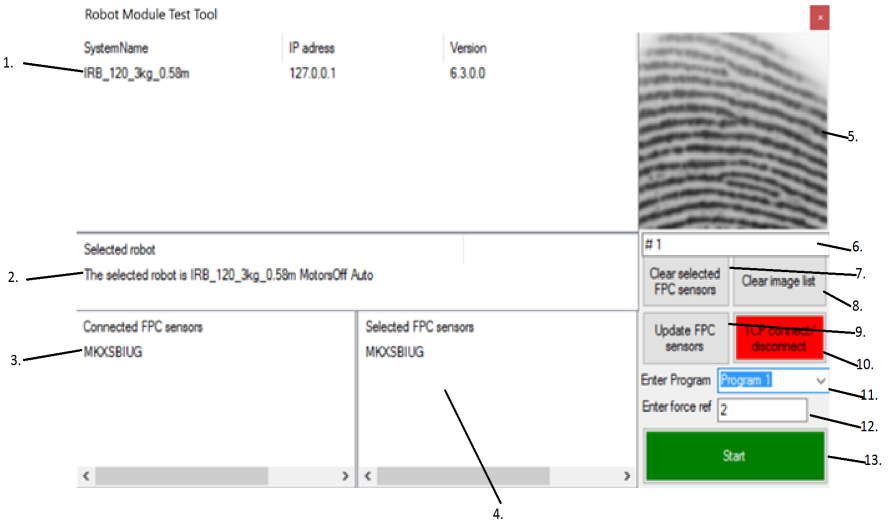


Figure 4.2 The robot control interface.

10. Connects or disconnects the connection to the TCP/IP server. Red indicates that there is no connection, yellow that the program is trying to connect and green that there is a connection.
11. A drop down list of all the ABB Rapid programs that can be run.
12. A text box were one enters the desired force to be used under the test.
13. A start button to start a new test using the set parameters. Green indicates that one is able to start a new test and yellow indicates that a test is currently running.

Additional functionality includes sending acknowledgements to the robot controller when a picture is taken and possibility to implement different tests of the fingerprint sensor, using Fingerprint Cards Module Test Tool.

4.2 ABB Rapid Program

The ABB Rapid programs are developed for the user to create new programs in the easiest possible way. This is done by using ABB's standard Rapid program structure and functions but with extended features to communicate with the force controller. Extended features include:

1. Switch on the force controller. The controller is turned on and the desired force is automatically set to the force chosen in the GUI.

2. Switch off the force controller. Turns off the force controller and sets the rapid program to the correct position to avoid large jumps with high speeds.

3. Set an offset from the current position. Makes it possible to move along the surface when in contact. Parameters to be set are direction to move or rotate, how far to move or rotate and the speed to use when doing this.

4. Acknowledge that the offset has been reached. Makes the rapid program pointer wait until the movement is done.

To show the functionality of the program five demo programs are created. The programs are described below.

1. Moves to sensor X, performs a test until an acknowledgement is given by the GUI, does this x times. Moves to sensor Y, performs a test until an acknowledgement is given by the GUI, rotates to a different angle and does this x times.

2. Tests all the connected sensors, moves to the next sensor when a picture is taken by the sensor.

3. Tests sensor X, when the correct force is applied, the probe is moved in a square over the surface using the offset function. This can be seen in a live stream using the FPC-TSD Touch fingerprint demo.

4. Tests sensor X, when the correct force is applied the probe rotates around the z-axis using the offset function. This can be seen in a live stream using the FPC-TSD Touch fingerprint demo.

5. Test sensor X, when the correct force is applied the probe rotates around the x-axis using the offset function. This can be seen in a live stream using the FPC-TSD Touch fingerprint demo.

4.3 ExtCtrl computer

This computer runs Linux and is host for three different programs. These programs are presented separately in the section below.

4.3.1 Opcom

The first program is Opcom described in Section 1.4.1. It enables the communication between the sensor program and the Matlab/Simulink generated program and it is also in charge of the communication to the axis computer. This program is used with all the programs running on the computer as well as the robot controller. In Opcom one is able to load a controller, submit and obtain signals. When loading a

controller Opcom loads a force controller compiled with real time workshop. Opcom is then able to "submit" signals from the robot controller's main computer and thereby read the signals presented in Figure 4.13. By pressing obtain one can modify the signal with the force controller and send the signal to the axis computer. A figure describing this is displayed in Figure 4.3.

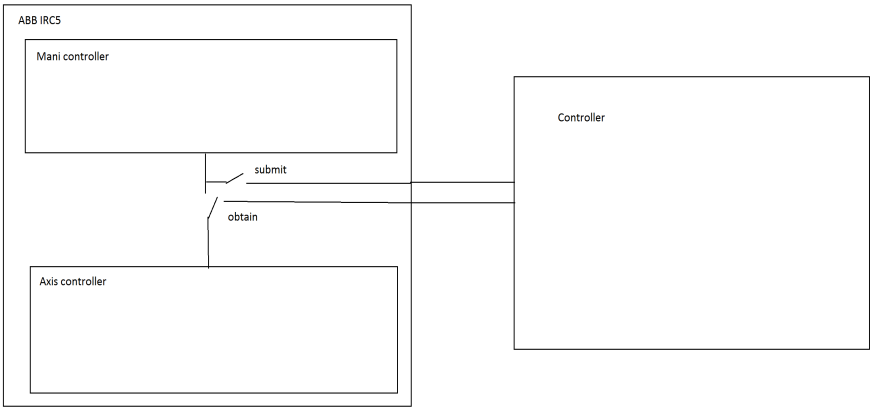


Figure 4.3 Visualization of submit and obtain [Blomdell et al., 2010].

4.3.2 The sensor programs

Many different sensor programs are developed during the thesis to read different sensors and to enable different functionality. The program that reads the JR3 sensor and the own developed sensor is only used to test the force controller and evaluate the performance of the sensors i.e., it does not involve GUI or communication to IRC5's main controller. One more thing to note is that all the sensor programs described below except for sensor program 3.2 is implemented using two programs; one that reads the sensor and one that interface with the controller. The two programs use Opcom to communicate to the IRC5's main computer.

1.0 This program is based on an existing C-program developed at the department of Automatic Control at LTH, which enables the user to read the JR3 sensor. Additional functionality is added to this program so the user can send the force measurements to the force controller program using Opcom, set the desired force and start the force controller. This program runs on an external computer and is reached using SSH (secure shell).

2.0 Written in C with the same functionality as for 1.0, but reads sensor data from the Arduino that is connected to the own developed force sensor instead of the JR3

sensor.

2.1 Uses the same sensor as 2.0 but the user is not allowed to set the force reference. Instead the program generates a random step length with two preset reference levels. This program is developed to be used to test the force controller and for system identification purposes.

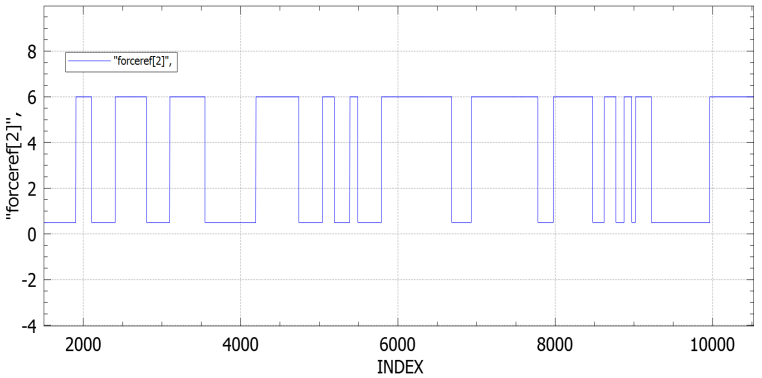


Figure 4.4 A random force reference generated by sensor program 2.1. The data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

3.0 Is written in $C++$ but uses the *C* libraries to enable communication through Opcom. This program is used to read the Optoforce sensor and has the same functionality as 1.0 and 2.0.

3.1 Same sensor as 3.0 but the same functionality as 2.1.

3.2 Same sensor as 3.0 and 3.1 but is developed to be run with the GUI and the main computer communication enabled. This is done by setting up the Opcom communication, enabling MocGen communication and starting a TCP/IP server. The server sends the current force applied on the sensor upon request.

4.3.3 Force controller programs

Written in Matlab/Simulink and utilizing the ExtCtrl interface. A couple of different control programs were programmed, to compensate for different sensors and utilize different functionality and different controllers. All these control programs are presented below but only the controller with the most functionality is presented in depth. One thing to note is that the dead time in the step responses is not caused by time delay. The controller is started close to the surface but not in direct contact, the robot is then "searching" for the surface and this is shown as a delay in the step response. The reason why the force measurements is not at zero when the

force reference is, is that there is an offset in the sensor measurements. This offset is removed when the controller is started. All the tests use the same contact surfaces which consists of a rubber probe and an aluminum block placed directly on a table.

1.0 Programmed to be used with JR3 sensor for testing. The controller is based on the Impedance controller described in Section 3.1.2. It works with sensor program 1.0. The result of a step response can be seen in Figure 4.5 and the convergence of the RLS-parameters in Figure 4.6 where "returnvector[11]"=K_e, "returnvector[12]"=D_e and "returnvector[13]"=$K_e x_e$.

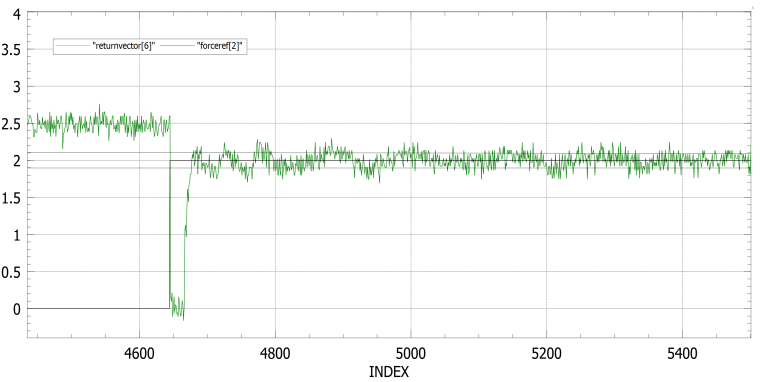


Figure 4.5 Step response of Impedance controller using JR3 sensor. The data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

Controller	Rise time 10-90 %	Rise time 5-95%	Overshoot	Settling time 5%	Settling time 2%
1.0	0.18 s	0.22 s	<5 %	0.24 s	— s

Table 4.1 The performance of the Impedance controller using the JR3 sensor.

1.1 Same controller and functionality as for 1.0, but works with sensor programs 2.x. A step response can be seen in Figure 4.7. The RLS parameters and their convergence can be seen in Figure 4.8, where "returnvector[11]"=K_e, "returnvector[12]"=D_e and "returnvector[13]"=$K_e x_e$.

1.2 Same controller and functionality as for 1.0 and 1.1 but works with sensor program 3.0. A step response can be seen in Figure 4.9 and the RLS parameters and their convergence can be seen in Figure 4.10, where "returnvector[11]"=K_e, "returnvector[12]"=D_e and "returnvector[13]"=$K_e x_e$.

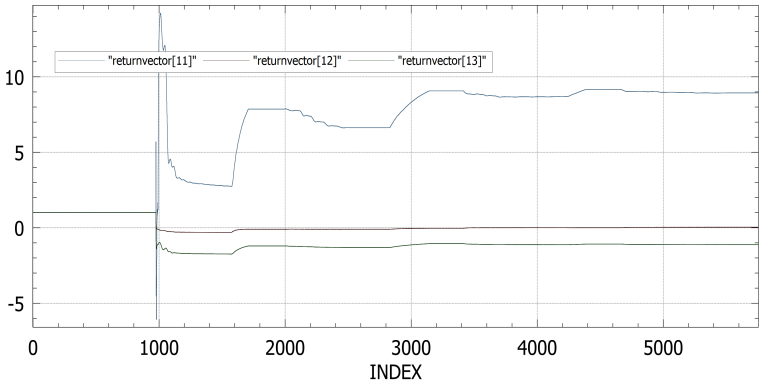


Figure 4.6 The convergence of the RLS-parameters using the JR3 sensor.

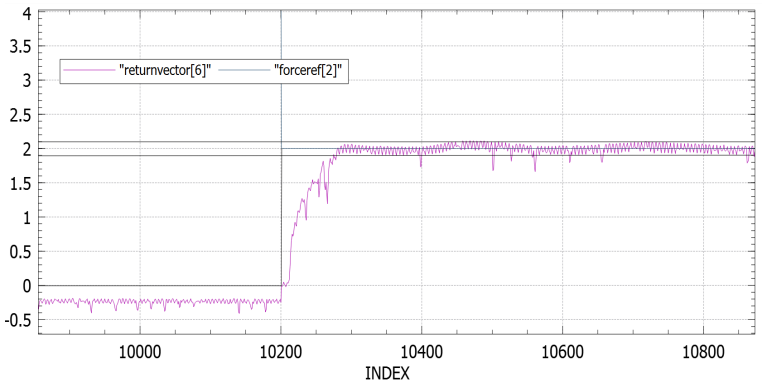


Figure 4.7 Step response of Impedance controller using the own developed force sensor. The data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

Controller	Rise time 10-90 %	Rise time 5-95%	Overshoot	Settling time 5%	Settling time 2%
1.1	1.14 s	1.34 s	<2 %	1.58 s	1.7 s

Table 4.2 The performance of the Impedance controller using own developed force sensor.

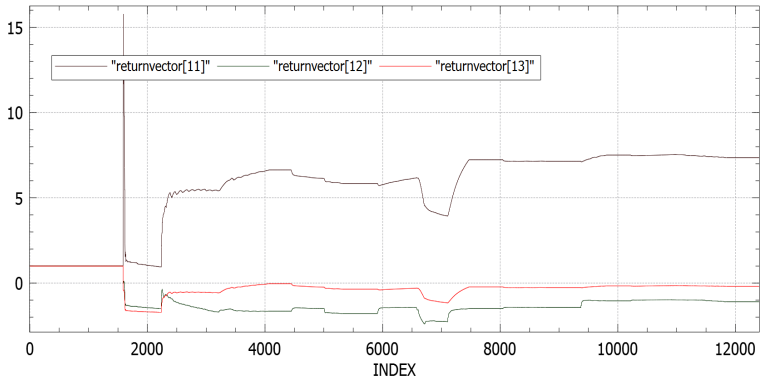


Figure 4.8 The convergence of the RLS-parameters using the own developed force sensor.

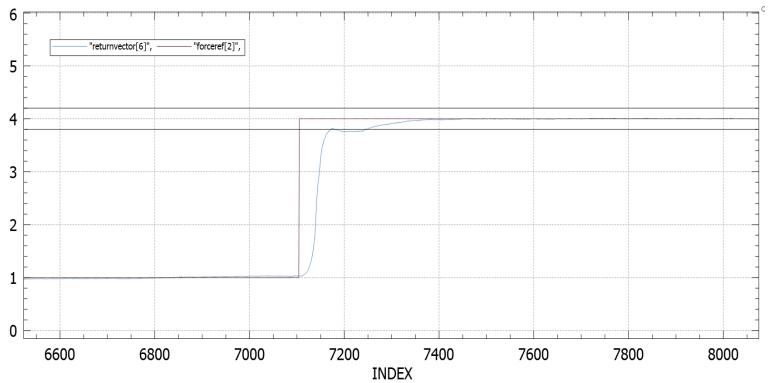


Figure 4.9 Step response of Impedance controller using the Optoforce sensor. The data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

Controller	Rise time 10-90 %	Rise time 5-95%	Overshoot	Settling time 5%	Settling time 2%
1.2	0.52 s	0.86 s	<2 %	2.78 s	4.28 s

Table 4.3 The performance of the Impedance controller using the Optoforce sensor.

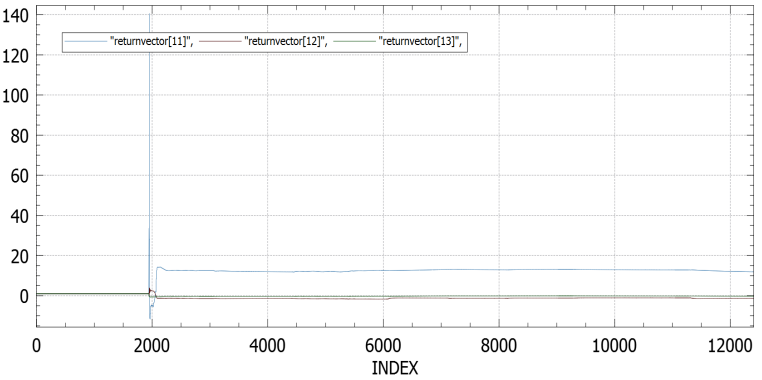


Figure 4.10 The convergence of the RLS-parameters using Optoforce sensor.

2.0 Same functionality and sensor as 1.2, but with a PID-controller instead. The step response can be seen in Figure 4.11 and in Figure 4.12. The only thing that differs in the two step responses is the position of the tool in x- and y-direction in base frame coordinates. A reason why the results varies depending of position of the tool is discussed in Section 5.3.

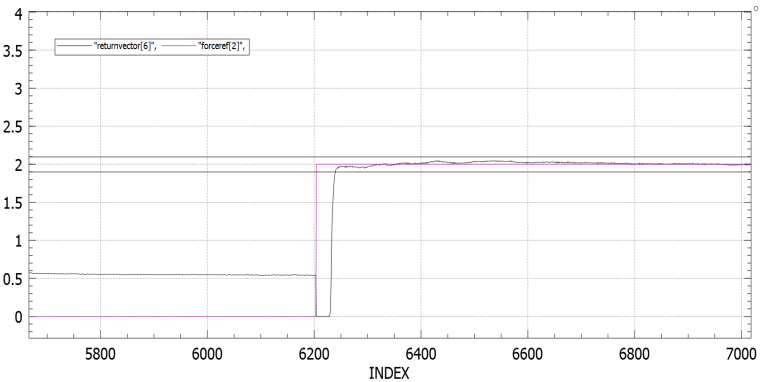


Figure 4.11 Step response one using PID-controller. The data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

Controller	Rise time 10-90 %	Rise time 5-95%	Overshoot	Settling time 5%	Settling time 2%
2.0	0.14 s	0.18 s	<2 %	0.22 s	0.38 s

Table 4.4 The performance of the PID-controller using the Optoforce sensor.

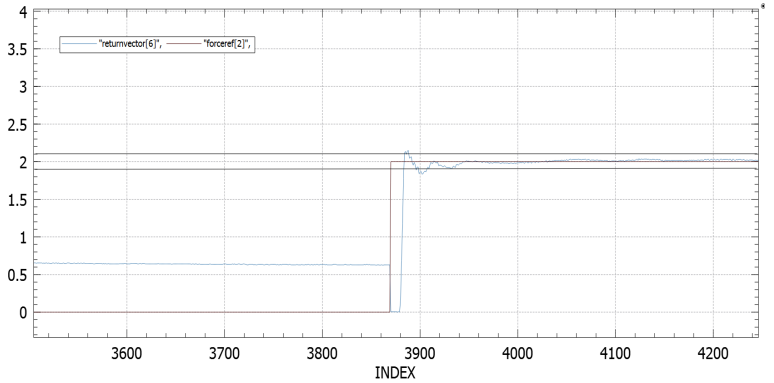


Figure 4.12 Step response one using PID-controller. The data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

Controller	Rise time 10-90 %	Rise time 5-95%	Overshoot	Settling time 5%	Settling time 2%
2.0	0.04 s	0.06 s	6 %	0.58 s	1.4 s

Table 4.5 The performance of the PID-controller using the Optoforce sensor.

3.0 Same controller and sensor as 2.0 but with the possibility to communicate with the Rapid program and the GUI.

The force controller program 3.0 is displayed in Figures 4.13 to 4.16, together with a brief presentation of the different blocks.

Simulink program level 1, Figure 4.13:

1. The irb2ext.x input blocks represent all the available outputs from the main robot controller and the ext2irb output blocks represent the input to the axis computer.

2. The two red blocks initiate the controller parameters such as sample time, robot type (used to get the correct kinematics) and other compiling information and displays the parameters.

3. The MocGen block sends and receives data from the Rapid program and is thereby the block that interprets all the Rapid functions.

4. The function-call subsystem contains the blocks in Figure 4.14 and it is this block that contains all the kinematics.

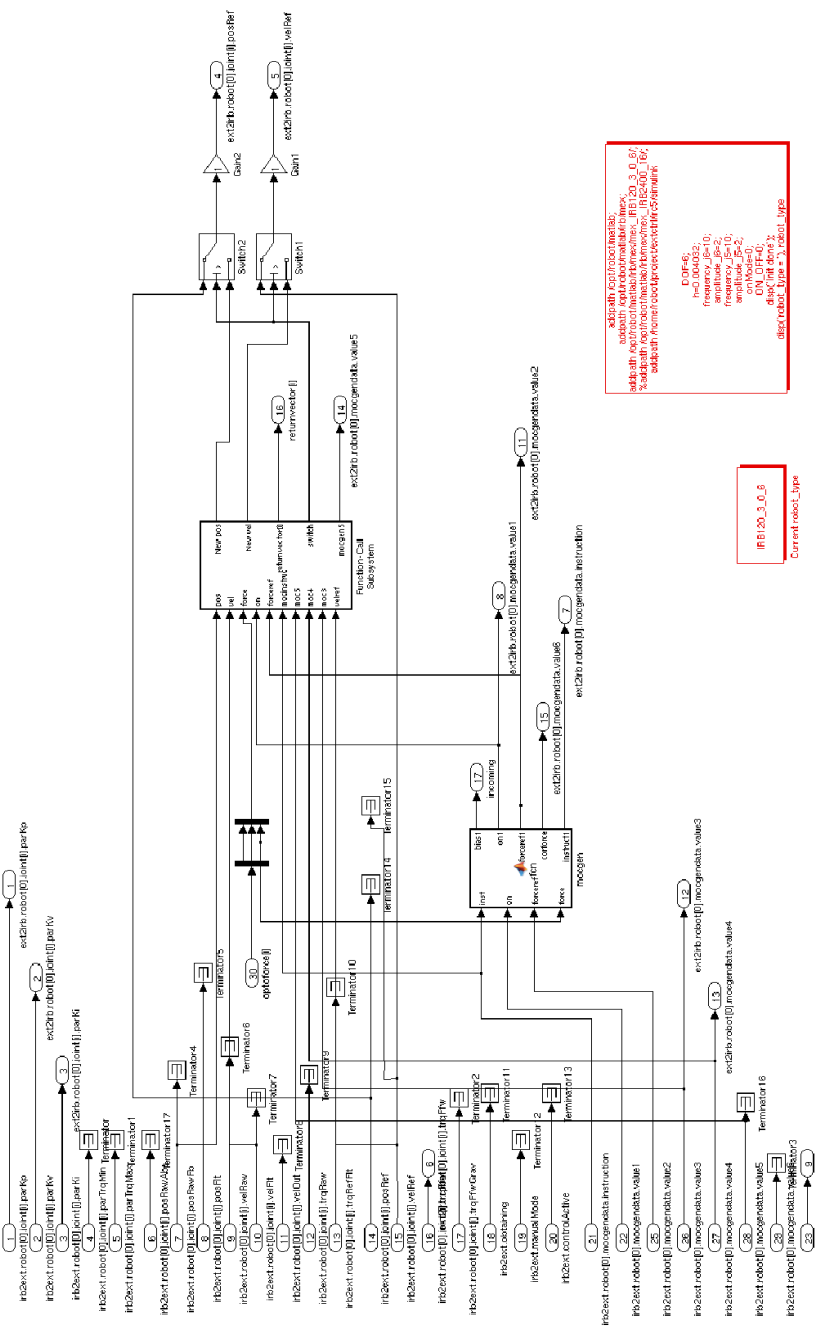


Figure 4.13 Simulink program level 1.

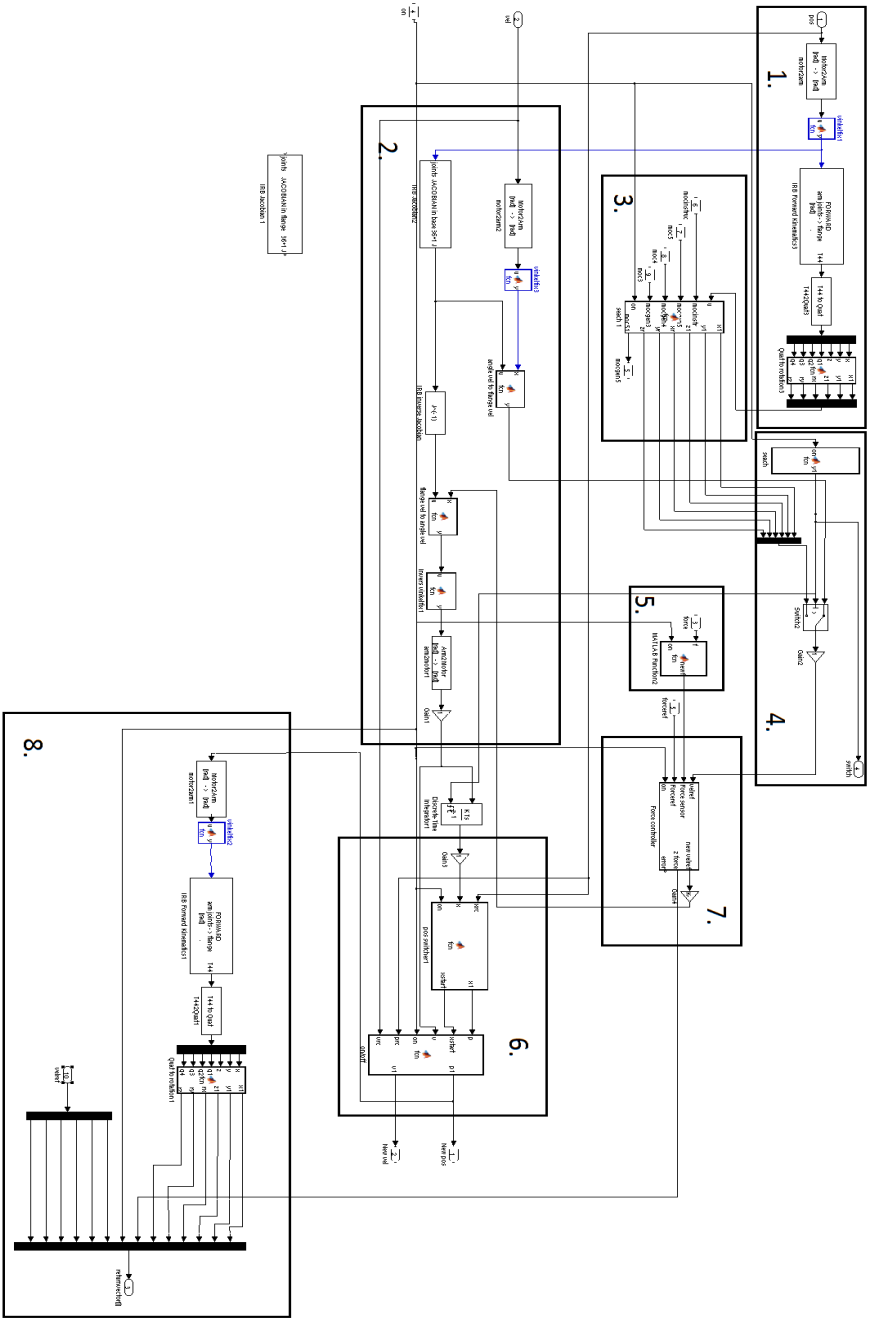


Figure 4.14 Simulink program level 2.

Simulink program level 2, Figure 4.14:

1. Kinematics from motor angles to tool position in base frame.
2. Velocity kinematics from motor angles, to robot force control signal to motor angles.
3. Reads and sends MocGen signals, sets and acknowledges offset position.
4. Decides when the controller should turn on and off.
5. Removes offset and calculates the force given the sensor values.
6. Remembers start position and acts as fail safe blocks.
7. Contains the block showed in Figure 4.15.
8. Output blocks displaying position and rotation of tool, current force.

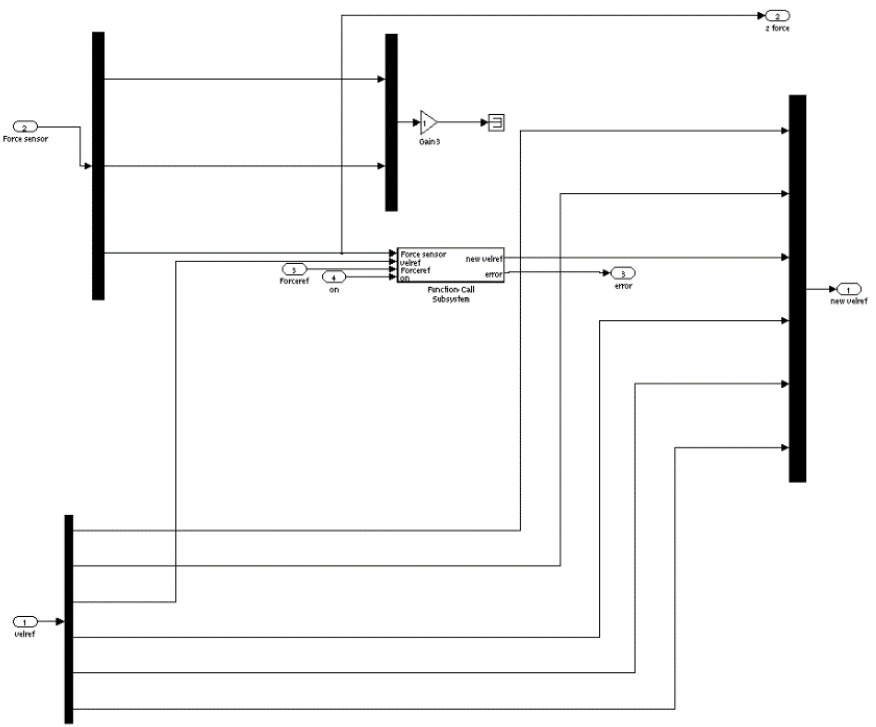


Figure 4.15 Simulink program level 3.

On level 3 one can choose which force signal/signals that is going to be used for the control. In Section 5.4 a further discussion of why only the force in the force sensors z-axis is utilized, is presented.

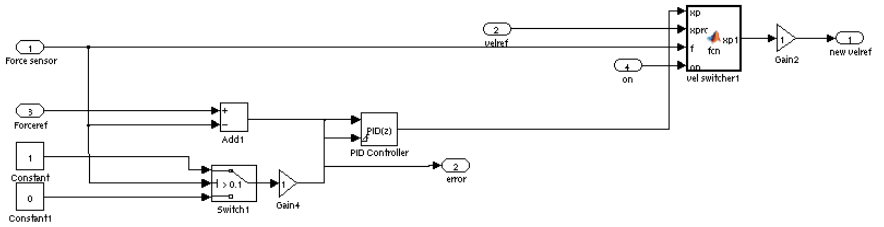


Figure 4.16 Simulink program level 4.

The final force controller is a discrete parallel PID-controller and is implemented using Simulinks PID-controller block. The performance of the controller is showed in Figures 4.11 and 4.12.

4.3.4 Arduino program

The program that is running on the Arduino can be seen in Appendix A.3. The program implements functionality as a moving average filter, conversion from voltage to force and serial communication to a connected computer running sensor program 2.x.

4.3.5 System identification

The identification of the system using the Optoforce sensor generated a discrete-time ARX-model (the degree of the C-polynomial in the ARMAX model is zero), sampled at 0.02 seconds. This model is described by Equation 4.6.

$$A(z)y(t) = B(z)u(t) + e(t), \begin{cases} A(z) = 1 - 1.295z^{-1} + 0.003601z^{-2} + 0.2912z^{-3} \\ B(z) = 0.09522z^{-2} + 0.04814^{-3} \end{cases} \tag{4.1}$$

The validation of this model is displayed in Figures 4.17 and 4.18, where Figure 4.17 is the overall fit to the validation data set and Figure 4.18 is the correlation function of residuals and cross correlation function between the inputs and the residuals from the output, using the validation data set.

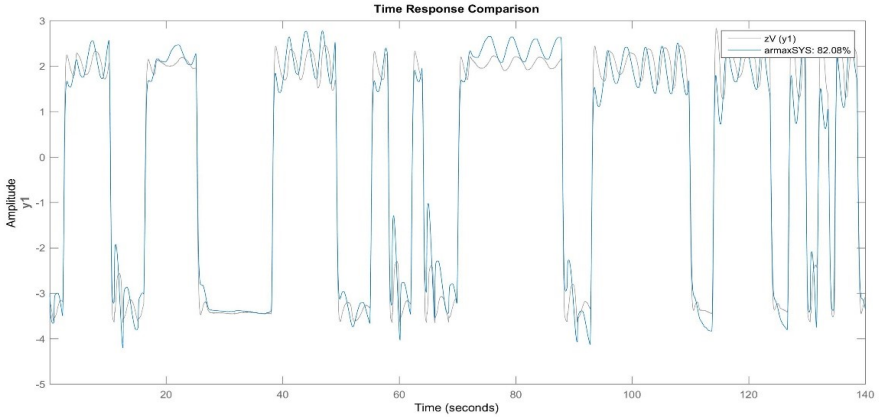


Figure 4.17 Model and validation data plot.

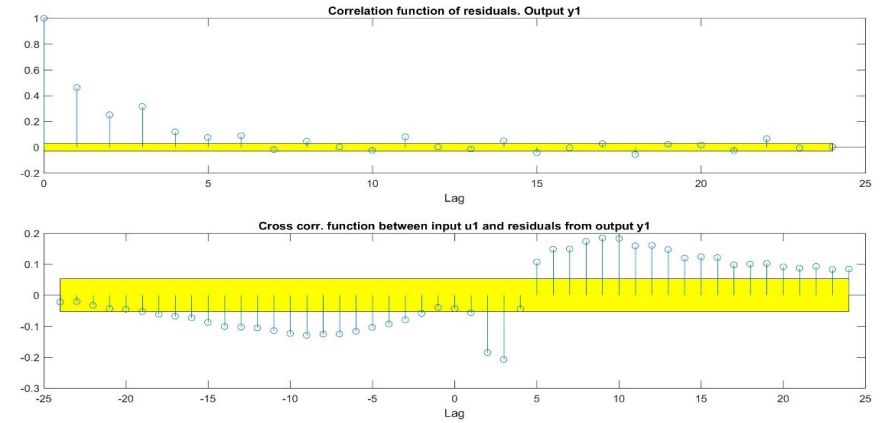


Figure 4.18 Residual plot.

4.3.6 Use of program

To show the use of the system, two pictures are taken when different forces is applied. The result of this can be seen in Figure 4.19. One can observe that the picture taken with larger force is sharper and more defined. This can e.g., be used to compare different coatings. FPC cannot do this comparison with an accurate result at the moment. This due to the fact that if one picture taken with one sensor with a speci-

fied coating is better than another picture taken with a sensor with another coating, may only depend on that a higher force is applied when this picture is taken.

Figure 4.19 Pictures taken with fingerprint sensor when different forces are applied.

Chapter 5

Conclusion and discussion

Over all I am quite satisfied with the results of the thesis, due to the fact that I managed to do everything I was set to do. The results derived can of course be improved and extended if more time is given. In the two following sections a conclusion and a discussion of the results, what improvements can be made and what future work can focus on, are presented.

5.1 Conclusion

The purpose of this thesis was to develop and examine a force feedback controller that could be used for the purposes mentioned in Section 1.2. This controller should be implemented in a system that is able to; preform different tests of the fingerprint sensors and controlled through a simple interface, preferably in a Windows environment. All these specifications were implemented and showed a satisfying result. The final conclusion one can draw from this is that a robot using force feedback is suitable for use when testing fingerprint sensors, even though the forces applied is very small and as long as the force sensor is good enough.

5.2 Models and system identification

The model derived from the system identification is not the best model, which is underlined by both the residual analysis in Figure 4.18 and by the fact that all the tested controllers had to be retuned. The result discussed in Section 5.2.2 further underlines why the model cannot capture all the dynamics of the system. The model made from low pass filters and a spring damper model did, despite its simplicity, capture the dynamics of the real process quite well, even though the real process is much more sensitive than the simulated. This led to that all the real controllers had to be slower than in the simulation. The Optoforce sensor does also behave

differently depending on where on the sensing surface the force is applied i.e., the sensor is deforming more along the edges then in the middle when the same force is applied.

5.3 Force controllers

The choice of controller is based on the performance of the different controllers. In the following section a discussion of the implemented controllers are presented.

5.3.1 Impedance controller

There are a couple of reasons why this controller was not the final choice of controller. First of all, the controller was the slowest both in the simulations and on the real process. When increasing the speed of the controller i.e., moving the poles further away from origin, the step response becomes faster as expected and in the simulation where no low pass filter is used, one is able to get a satisfying result. But as for the simulation with low pass filters and even more so on the real process, the faster response results in higher overshoot and more oscillations until the point where the system becomes unstable. As for the RLS-algorithm it does what it should. The convergence in Figure 4.8, where the RLS-parameters do not converge in a completely uniform matter, can be explained with the fact that a mass, spring, damper system cannot model the system good enough so the parameters will change depending on e.g., force applied, position etc.

5.3.2 Direct force controllers

The difference in step response when using the same controller but at different positions can be explained by the fact that the robot dynamics change depending on its position, e.g., the robot does not have the same dynamics when the end effector is close to the base as it has when the end effector is far away from the base. An evidence of this can be seen in Figure 5.1 and Figure 5.2 where the force is plotted alongside the tool position in z-direction in base frame. If one looks at which sample the peak force is in the lower plot in Figure 5.1 and compare this to the z-position of the robot, one sees that even though the position of the tool is lower in later samples the force is lower. This implies, due to the fact that the force sensor just measures deformation, that the robot is moving in the z-direction even though it has stopped according to the position measurements. If one then looks at Figure 5.2 one can see that this "overshoot" is much smaller and thereby gives a faster settling time.

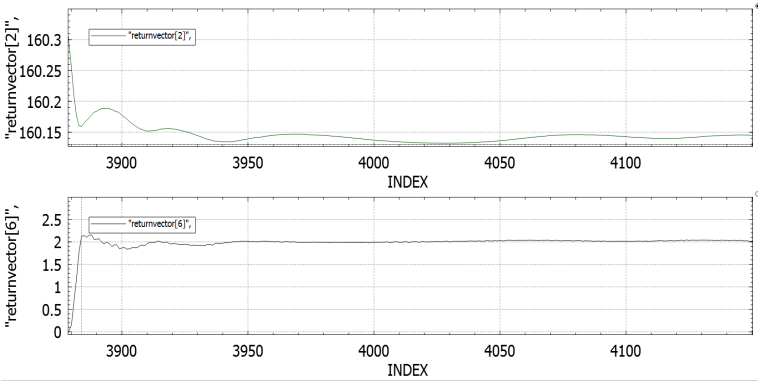


Figure 5.1 Force to position comparison. I the upper plot the data is plotted with the position in z-direction in base frame on the y-axis and sample index on the x-axis (one sample = 0.02 s)., I the lower plot the data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

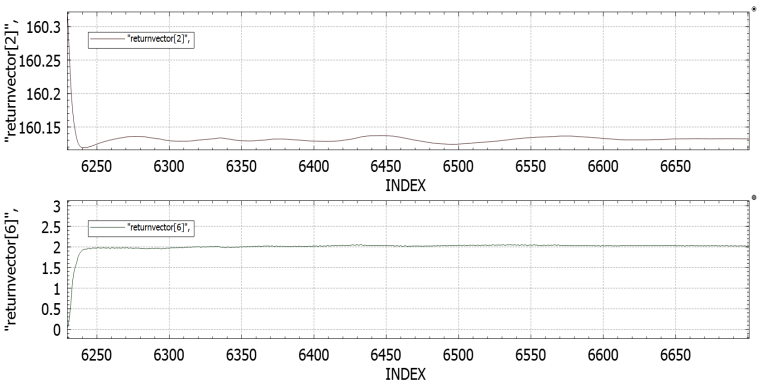


Figure 5.2 Force to position comparison. In the upper plot the data is plotted with the position in z-direction in base frame on the y-axis and sample index on the x-axis (one sample = 0.02 s)., In the lower plot the data is plotted with force (N) on the y-axis and sample index on the x-axis (one sample = 0.02 s).

5.4 Force sensors

5.4.1 Optoforce sensor

The Optoforce sensor has a quite big hysteresis and crosstalk i.e., it takes a long time for the sensor to get back to its initial reading after a force is applied and a

force in e.g., z-directions results in a change of the force in x- and y-directions. This became a problem during the calibration due to the fact that the sensor did not show the same value when applying the same weight before and after a heavier weight was applied. This was solved by taking more values with a lower weight applied and by waiting before applying the next weight. The Optoforce sensor is selected to be used in the final setup, due to the fact that it is possible to place it on the robot and it has less noise than the JR3 sensor. The Optoforce sensor utilization of rubber in the design acts like a low pass filter which together with the small weight of the probe makes the noise level about the same as when the sensor is placed on the table. This is the big advantage of this sensor. The Optoforce sensor is sufficiently good for demo purposes but will not be a suitable solution for industry application. The reason to why this is, is listed below:

1. It is not calibrated and there are no instructions on how to get an accurate calibration.

2. The sensor displays different amount of crosstalk depending on where the force is applied on the sensor surface i.e., if the force is applied in the center surface, very little crosstalk to the force in x- and y-directions is showed, but when the force is applied further away from the sensors center the crosstalk increases i.e., the force in x- and y-direction increases. This is an example of bad design of the sensor. This is one of the reasons why only the force in the z-direction is used and not the force given by Equation 2.7. Further reasons why I made this choice is that Equation 2.7 is not applicable when moving over the surface. This because a force is not just applied in the nominal direction of the surface but also in a direction opposite to the movement, in form of a friction force. The test when applying the probe with an angle is the only test that is negatively impacted, but this test is done with a small angle which makes the force measurements in only z-direction sufficient in demo purposes.

3. When moving over the surface when in contact, the sensor "bends".

4. It has too big hysteresis to take repeatable measurements (2% of 100 N).

5.4.2 Own developed force sensor

The own developed force sensor performed well, especially when taking into consideration the amount of amplification. The sensor has a linear relation between voltage and force applied, as one can see in the calibration data. One thing I noticed with the sensor is that the noise increased over time. The first time it was run the sensor had around the same noise level as the Optoforce sensor and nearly no transients, this is to compare with the noise in Figure 4.7. The reason for this is probably that the sensor and the cables used have worn over time, due to the lack of enclosure.

5.4.3 JR3 sensor

The JR3 sensor was used together with force controller 1.0, after that, it was discarded due to that the noise level was too high for the application, even though the sensor was placed on the bench and not on the robot.

5.5 Future work

Suggestions to make the test setup better:
First of all is the choice of force sensor. In future work I would recommend to place sensors under the fingerprint sensors. This will lower the noise and the force offset does not need to be reset when rotating the probe. To minimize the amount of sensors I would recommend to use four 1 DOF force sensors and place them under a plate. The test stations can then be placed on top of this plate.

In the regard of the GUI, many more things can be implemented that has not been implemented due to lack of time. One of these things is to auto detect which sensors are connected to which sensor module and from this information display only the program that is possible to run and to automatically know which probe to use. The force controller can with a bit more tuning perform better, especially if another force sensor is used. Regarding the rest of the controller program, more functionality could be added and more fail safe and error handling functions should be added.

Bibliography

ABB. *RobotStudio® It's just like having the real robot on your PC!* Accessed: 2016-06-07. URL: http://new.abb.com/products/robotics/robotstudio.

ABB (2010). *Learn with ABB*. Accessed: 2016-05-31. URL: https://library.e.abb.com/public/9a0dacfdec8aa03dc12578ca003bfd2a/Learn%20with%20ABB.%20Robotic%20package%20for%20education.pdf.

Arduino. *Arduino 1.6.9*. Accessed: 2016-05-31. URL: https://www.arduino.cc/en/Main/Software.

Blomdell, A., I. Dressler, K. Nilsson, and A. Robertsson (2010). "Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers". In: *2010 IEEE International Conference on Robotics and Automation*. Anchorage, Alaska.

Crowther, P. (2015). *YuMi-IRB 14000*. Accessed: 2016-05-31. URL: https://library.e.abb.com/public/4bcf2603f76f49088b80f7f1c49045eb/IRB14000ExternalVersionFinal.pdf.

Freidovich, L. B. (2014). *Control methods for robotic applications: lecture notes, Umeå University, Sweden*.

Glad, T. & Ljung, L. (2003). *Reglerteori: flervariabla och olinjära metoder*. Studentlitteratur. ISBN: 9789144030036. URL: https://books.google.ie/books?id=3sTsMAAACAAJ.

Hogan, N. (1985). *Impedance Control: An Approach to Manipulation: Parts 1-3. ASME J. Dynamic Systems, Measurement, and Control*, pp. 1–24.

Hägglund, T. (2013). *REGLERTEKNIK AK: Föreläsningar*. Instetutionen för reglerteknik, Lunds tekniska högskola.

Johansson, R. (2015). *Predictive and Adaptive Control*. Dept. Automatic Control, Lund University, Lund, Sweden.

Johansson, R., K. Nilsson, and A. Robertsson (2015). "Force control". In: *Handbook of Manufacturing Engineering and Technology*. Springer-Verlag, London.

JR3 (2016). *Products*. Accessed: 2016-06-08. URL: http://www.jr3.com/products.html.

Keijser, N. D. (2012). *ABB's smallest robot - IRB 120*. Accessed: 2016-05-31. URL: https://library.e.abb.com/public/4ba04345ce481405482579800011cbfa/Product%20presentation%20IRB%20120%20Revision%20E.pdf.

Mathworks (2016a). *Features*. Accessed: 2016-07-20. URL: http://se.mathworks.com/products/matlab/features.html.

Mathworks (2016b). *Linear-Quadratic-Gaussian (LQG) design*. Accessed: 2016-06-17. URL: http://se.mathworks.com/help/control/ref/lqg.html.

Mathworks (2016c). *PID Controller, Discrete PID Controller*. Accessed: 2016-06-11. URL: http://se.mathworks.com/help/simulink/slref/pidcontroller.html#bugphy7.

Microsoft. *Application Development*. Accessed: 2016-05-31. URL: https://www.visualstudio.com/features/.

National Instruments. *How is temperature affecting your strain measurments accuracy?* Accessed: 2016-05-31. URL: http://www.ni.com/white-paper/3432/en/.

Nikoleris, G. (2015). *MMKF15 Applied Robotics*. Lecture notes from lecture three in course MMKF15 Applied Robotics, LTH.

Optoforce (2016). *Technology*. Accessed: 2016-05-31. URL: http://optoforce.com/technology/.

Spong, M. W., S. Hutchinson, and M. Vidyasagar (2006). *Robot modeling and control*. John Wiley & Sons, Hoboken (N.J.) ISBN: 0-471-64990-2.

Stolt, A. (2015). *On Robotic Assembly using Contact Force Control and Estimation*. PhD thesis TFRT-1109. Department of Automatic Control, Lund University, Sweden.

Universal Robots (2015). *UR3 Tecnical Specification*. Accessed: 2016-05-31. URL: http://www.universal-robots.com/media/240736/ur3_en.pdf.

Vougioukas, S. (2001). "Bias Estimation and Gravity Compensation For Force-Torque Sensors". Accessed: 2016-05-31. URL: http://www.wseas.us/e-library/conferences/crete2002/papers/444-809.pdf.

Wikipedia. *Instrumentation amplifier*. Accessed: 2016-05-31. URL: https://en.wikipedia.org/wiki/Instrumentation_amplifier.

Wikipedia. *Strain gauge*. Accessed: 2016-05-31. URL: https://en.wikipedia.org/wiki/Strain_gauge.

Åkesson, J. (2006). "MPCtools 1.0 — Reference Manual". Accessed: 2016-06-10. URL: http://www.control.lth.se/media/Research/Tools/MPCtools/MPCtools-1.0-manual.pdf.

Appendix A

Appendix

A.1 System identification script

The script below can be used to get the ARMA-modell that has the highest fit to the validation data (zV) based on a model made from the identification data (zI) using system identification toolbox in Matlab. The value 0.02 corresponds to the sample time of the data.

```
outputdata=[force_data1(2000:1:9000);...
force_data2(2000:1:9000)];
verifieringsoutputdata=[force_data3(2000:1:9000)];
inputdata=[vel_data1(2000:1:9000);...
vel_data2(2000:1:9000)];
verifieringsinputdata=[vel_data3(2000:1:9000)];

zI=iddata(outputdata,inputdata,0.02);
zI = detrend(zI,0);
zV=iddata(verifieringsoutputdata,...
verifieringsinputdata,0.02);
zV = detrend(zV,0);

%% Decide ARMAX order
Fitvector = [];
order_vec = [];
FITvalvec=[];
FITvalvec1=[];
bestOrder=0;
bestFIT=0;

for na = 2:5
```

```
    for nb = 1:3
        for nc = 0:2
            for nk = 0:2
                armaxSYS = armax(zI,[na nb nc nk]);
                Fitvector = [Fitvector;...
                armaxSYS.Report.Fit.FitPercent];
                order_vec=[order_vec;[na nb nc nk]];
                [YH, FIT, X0]=compare(zV,armaxSYS);
                FITvalvec=[FITvalvec;FIT];
                if(FIT>bestFIT)
                    bestOrder=[na nb nc nk];
                    bestFIT=FIT;
                end
            end

        end

    end

end
bestFIT
bestOrder
x=1:1:length(FITvalvec);
plot(x,FITvalvec);
```

A.2 6 DOF force sensor, gravity compensation script

The script below can be used to find gravity compensation parameters when using a 6 DOF force/torqe sensor, by giving an estimation of the bias, mass and center of gravity.

```
%% Bias estimation

%test algorithm
f=1:6;
e=1:6;
e=e*-1;
for i=1:12
P(:,1,i)=f;
P(:,2,i)=e;
end
%}
```

```
s=[0,0,0,0,0,0];
for j=1:12
    for i=1:6
    s(i)=s(i)+(P(i,1,j)+P(i,2,j))/2;
    end
end
bias=s./12;
%% Mass estimation
g=[0 0 -9.81]';%gravity vector in world coordinates
%Needs Rotation matrix and force measurements
R(:,:,1)=[0 0 1;1 0 0;0 1 0];
R(:,:,13)=[0 0 -1;1 0 0;0 -1 0];
R(:,:,2)=[0 -1 0;1 0 0;0 0 1];
R(:,:,14)=[0 1 0;1 0 0;0 0 -1];
R(:,:,3)=[1 0 0;0 0 -1;0 1 0];
R(:,:,15)=[1 0 0;0 0 1;0 -1 0];
R(:,:,4)=[1 0 0;0 1 0;0 0 1];
R(:,:,16)=[1 0 0;0 -1 0;0 0 -1];
R(:,:,5)=[0 0 1;0 1 0;-1 0 0];
R(:,:,17)=[0 0 -1;0 1 0;1 0 0];
R(:,:,6)=[-1 0 0;0 -1 0;0 0 1];
R(:,:,18)=[-1 0 0;0 1 0;0 0 -1]
R(:,:,7)=[0 1 0;0 0 -1;-1 0 0];
R(:,:,19)=[0 1 0;0 0 1;1 0 0];
R(:,:,8)=[0 1 0;-1 0 0;0 0 1];
R(:,:,20)=[0 1 0;1 0 0;0 0 -1];
R(:,:,9)=[0 0 1;0 1 0;-1 0 0];
R(:,:,21)=[0 0 1;0 -1 0;1 0 0];
R(:,:,10)=[0 0 1;1 0 0;0 1 0];
R(:,:,22)=[0 0 1;-1 0 0;0 -1 0];
R(:,:,11)=[0 -1 0;0 0 1;-1 0 0];
R(:,:,23)=[0 1 0;0 0 1;1 0 0];
R(:,:,12)=[1 0 0;0 0 1;0 -1 0];
R(:,:,24)=[-1 0 0;0 0 1;0 1 0];

Gs=R(:,:,1)'*g;
for i=2:24
Gs=[Gs;R(:,:,i)'*g];
end
%Gs=Gs';
f=P(1:3,1,1)-bias(1:3)';
for i=2:12
    f=[f;P(1:3,1,i)-bias(1:3)'];
```

```
end
for i=1:12
    f=[f;P(1:3,2,i)-bias(1:3)'];

end

%f=f';
m=(Gs'*f)/(Gs'*Gs);

%% Estimation COG
gz=Gs(3);
gy=Gs(32);
A=[0 gz -gy;-gz 0 gz;gy -gz 0];
for i=2:24
gz=Gs(3*i);
gy=Gs(3*i-1);
A=[A;[0 gz -gy;-gz 0 gx;gy -gx 0]];
end

tau=P(4:6,1,1)-bias(4:6)';
for i=2:12
    tau=[tau;P(4:6,1,i)-bias(4:6)'];

end
for i=1:12
    tau=[tau;P(4:6,2,i)-bias(4:6)'];

end

r=1/m*inv(A'*A)*A'*tau;

%% Gravity compensation
%the gravitycompensation is given by:
Fg=[m*R'*g;cross(m*r,R'*g)]
```

A.3 Arduino code

Arduino code that reads a strain gauge sensor connected via an instrumental amplifier. The readings is filtered using a Moving average filter and then sent via serial communication to a receiving C-program.

```
const int analogInPin = A0;
int s=0;
double a=0;
double y=0;
int i=0;
int window=150;
double bufSum=0;
double buf[300];
int sensorValue = 0;

void setup()
{
pinMode(13,OUTPUT);
Serial.begin(115200);
}
 int flag =0;
 char c;
void loop()
{
  sensorValue = analogRead(analogInPin);
  s=sensorValue;
  a=(s-155.7811)/0.51620;
  a=a/1000*9.81;
  bufSum = bufSum - buf[i];
  buf[i] = a;
  bufSum = bufSum + a;
  i = i + 1;
  if (i > window){
    i = 1;
  }
  y = bufSum/window;

  c=Serial.read();
  Serial.flush();
  if(c='t'){
       Serial.println(y,5);
       Serial.flush();
  }
}
```

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

Document name
MASTER´S THESIS

Date of issue
August 2016

Document Number
ISRN LUTFD2/TFRT--6014--SE

Author(s)
Viktor Johansson

Supervisor
Markus Andersson, Fingerprint Cards AB
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Fingerprint Sensor Testing Using Force Feedback Control

Abstract

Testing and validation of fingerprint sensors with human labor is time consuming, costly and lacks the speed, accuracy and repeatability required for sufficient results. The thesis investigates the possibility of an automated solution using an industrial robot with force feedback control. Several force feedback controllers are implemented and evaluated. The best controller is implemented together with a graphical user interface and programming of an industrial robot manipulator. The setup can successfully test multiple sensors and display pictures taken with a fingerprint sensor, together with the applied force.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-74

Recipient's notes

Security classification

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