ISSN 0280-5316
ISRN LUTFD2/TFRT--5837--SE

Haptic Interface for a Contact Force Controlled Robot

Fredrik Eriksson
Marcus Welander

Department of Automatic Control
Lund University
May 2009

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

Document name
MASTER THESIS

Date of issue
May 2009

Document Number
ISRN LUTFD2/TFRT--5837--SE

Author(s)
Fredrik Eriksson and Marcus Welander

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

Sponsoring organization

Title and subtitle
Haptic Interface for a Contact Force Controlled Robot
(Haptiskt gränssnitt för en kontaktkraftreglerad robot)

Abstract
The use of haptics in teleoperation and robotics can help the human operator to greatly improve the performance. This master thesis presents a haptic interface between a haptic device, a Phantom Premium A, and an industrial robot, an IRB140B, on which a force sensor was mounted. This will enable a human operator controlling the Phantom to get information about the environment surrounding the IRB140B through the sense of touch. An impedance controller is also introduced in order to avoid too large contact forces in the robot-environment interaction and to obtain a good behavior of the Phantom.
The haptic interface was achieved by deriving a general mathematical mapping from the states of a haptic device to these of an industrial robot and by implementing that mapping for the Phantom and the IRB140B. To handle the Phantom e.g., the force feedback, a C++ program was developed. The program also features a virtual representation of the IRB140B. The impedance controller for the robot-environment interaction was implemented in Matlab's Simulink. That controller was translated into C code and downloaded to the axis computer of the IRB140B. Everything was connected via a TCP/IP network. Tuning the impedance controller resulted in a stable teleoperation system with haptic feedback to the human operator for the materials touched by the IRB140Be.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
98

Recipient's notes

Security classification

Contents

1. Introduction

While robotics is a fairly well known field of science nowadays, haptics is a relatively unknown one. It is a field of great potential which may be a reason why the number of articles written each year within the field of haptics increase rapidly. Combining haptics with robotics, allows humans to perform different tasks using a robot but still with the feeling that they are doing it by hand. That can be used in a wide range of areas where one of the more important ones is teleoperation in surgery. Today, robots are used in surgery, but tactile feedback is lacking. For a surgeon to be able to operate with pinpoint precision through a small incision with the sensation of doing it with his own hands instead of joysticks and robots, haptics enters the picture.

The goal of this master thesis is that a human operator controlling a haptic device should be able to get information about the environment surrounding an industrial robot through the sense of touch. A contact force controller should also be developed in order to avoid too large forces in the robot-environment interaction and to obtain a good behavior of the haptic device.

This should be achieved by a mathematical mapping from the states of a haptic device to these of an industrial robot and by connecting the haptic device and the robot via a TCP/IP network. The human operator should be able to control the movements of the tool held by the robot by using the haptic device. To make the robot follow the directions from the haptic device properly a contact force controller should be implemented. Furthermore, the force sensed by the robot should also be sent as feedback to the haptic device to close the haptic loop.

1.1 Previous Work

There exist many articles on the theory of different master and slave systems, how they work and how they can be controlled, e.g., [14], [16] and [9]. One thing that was not included in this thesis, due to a limited time span, was an investigation of how the time delays that appear in such systems may affect the performance and how that negative effect can be reduced, [4].

At the Robotics Lab at the Department of Automatic Control, Lund University, work concerning different forms of force control for industrial robots has been done before. This thesis will use some knowledge from that previous work and aims to integrate force control with haptics.

1.2 Motivation

At the Robotics Lab at the Department of Automatic Control, Lund University, where this master thesis was done, no previous work within the combination of haptics and robotics has been performed. Hopefully, this thesis will therefore serve as a foundation for future work in this area at the Department and maybe as an inspiration to others to pick up where this work left off.

In the future, the Department of Automatic Control wishes to continue working with haptics and to explore its potential when integrated with robots in applications in the field of medicine.

1.3 Outline

In Chapters 2 and 3 the foundations of haptics and robotics are presented. The devices used in this thesis are described. These two chapters serve as a brief introduction for a reader with no or little experience from haptics or robotics respectively but will however also provide the more experienced reader with some useful basic information needed later on in the report. Especially Section 3.4, where some important notations are defined, is recommended for everyone.

The mathematical mapping from a haptic device to an industrial robot is stated in Chapter 4. How they were connected is described in Chapter 5, which also describes the communication.

To make the haptic feedback possible a force sensor must be attached to the industrial robot. How the calibration of that sensor was done is shown in Chapter 6.

A Simulink model is needed in the communication and it is presented in Chapter 7. Furthermore, a computer program taking care of the haptic device was developed and how it works is presented in Chapter 8. How the program code was implemented to achieve that behavior is described in Chapter 9.

The performance of the full system, the results and conclusions of different tests of it are presented in Chapter 10. In Chapter 11 a summary of this work is presented together with a brief presentation of some future work, i.e., what can be done to improve the results of this thesis and the performance of the full system implementation.

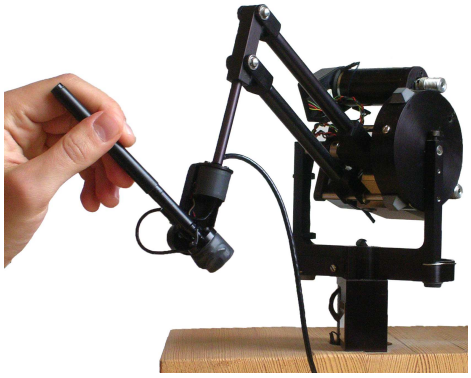

Figure 2.1 The Phantom Premium A from *SensAble Technologies, Inc.* is the haptic device used throughout the thesis.

2. Haptics

Haptics is the field of tactile feedback in human-machine interactions. A haptic device is an instrument that provides the user's sense of touch with feedback. Such a device is defined by its number of degrees of freedom in motion and feedback, as these not necessarily need to agree. Haptic devices may be used for entertainment purposes in computer games but also have a great potential in more serious contexts. For example in surgery, both in training and in practice, a haptic device can help a surgeon using teleoperation to perform better [24].

2.1 Haptic Devices

At the Virtual Reality Lab at Ingvar Kamprad Design Centrum (IKDC), Lund, several haptic devices are at hand. Among these, two were considered for this thesis. The *Novint Falcon* by *Novint Technologies, Inc.* [17] and the *Phantom Premium A* by *SensAble Technologies, Inc.* [22]. Due to the superior performance of the Phantom Premium A, it was selected as the haptic device to work with throughout this thesis.

The Phantom Premium A

The Phantom Premium A is used by holding the end of the device as if it were a pen, see Figure 2.1. Six degrees of freedom in motion is featured, but only three degrees of freedom in force feedback. Compared to the Novint Falcon it is more expensive but with much better performance. The user hardly feels any friction at all when moving the "pen" around, twisting or turning it and the workspace does not feel as limited as with the Novint Falcon. The three motors of the Phantom Premium A are stronger than the motors on the Novint Falcon. This can help produce more distinct boundaries of objects in the virtual world.

The Phantom Premium A is connected to a computer through a 110 V amplifier box to a PCI-card. The drivers at hand at IKDC are for Windows. The available software includes a program that features calibration and testing of the device.

Figure 2.2 The Novint Falcon by *Novint Technologies, Inc.* is a rather simple haptic device.

The Novint Falcon

This is a haptic device developed for gaming purposes. Thus, it is more robust than the Phantom Premium A but also more simple, see Figure 2.2. For starters it only has three degrees of freedom in both motion and force feedback. All three are translational so no rotations are featured. Furthermore its workspace is considerably smaller than that of the Phantom Premium A and the user becomes very aware of the arms holding the knob in place when using the Novint Falcon.

2.2 OpenHaptics

The Application Programming Interface (API) chosen for the thesis is *OpenHaptics*, [20], [21] and [22]. It is developed by *SensAble Technologies, Inc.* and therefore only compatible with the Phantom series. To be able to use it, a license must be obtained from *SensAble*. This API was chosen because of its close relation to the Phantom Premium A in Section 2.1. The downside of using *OpenHaptics* is of course that any software developed can not be used on other brands of haptic devices, such as the Novint Falcon in Section 2.1.

OpenHaptics is based on *OpenGL* (*Open Graphics Library*), an API for 2D and 3D graphics compatible with several programming languages and platforms, [19]. This makes *OpenHaptics* easy to work with for a programmer familiar with *OpenGL*.

OpenHaptics is divided into two APIs, the *HDAPI* and the *HLAPI*. The *HDAPI* stands for *Haptic Device API* and it lets the user take care of low level communication with the Phantom, e.g., position, velocity, force feedback, raw data I/O (encoder/DAC) etc.

The *HLAPI* is the *High Level API* and is built upon the *HDAPI*. The *HLAPI* handles the force rendering without the programmer needing to know anything about the force equations involved. It also incorporates the use of graphics, i.e., (*OpenGL*), together with the haptics and takes care of all synchronization between the underlying haptic and graphic threads needed. For more detailed information, see [21].

The API chosen to be used in this thesis was *HDAPI*. The *HDAPI* gives a straight forward way of coding the force feedback without the extra complexity of the features found in *HLAPI*.

3. Robotics

Robotics is the scientific field of robots and is a relatively new field but with increasing importance in the modern world. It is about the knowledge of constructing, designing and controlling robots. There are many kinds of robots but in this thesis a robot is referred to as an industrial robot. Industrial robots are often used in welding, pick-and-place and assembly operations in factories or in other situations where something tiresome, monotonic or outright dangerous for a human to do should be performed.

A robot can kinematically be described as links connected by joints. To control the robot it is necessary to have some knowledge about the position and orientation of each joint in comparison to a universal coordinate system. One way of choosing the description of position and orientation of the joints is presented in Section 3.5. This chapter also features other basic mathematic tools needed when dealing with robots, [23] and [6].

Figure 3.1 The IRB140B by *ABB*. The photograph is taken from [1].

3.1 IRB140B

The robot used in this thesis is the *IRB140B*, see Figure 3.1, from *ABB*, [1]. Information about the IRB140B can be found in [2]. It is a serial robot with six degrees of freedom, one for each of its six revolute joints. A revolute joint revolves around its so called joint axis. The joints are numbered from 1 to 6, starting at the base. To each joint corresponds a coordinate frame. Some joint frames have special names beside their number, see Figure 3.2. The frame of the first joint is called the *base frame* and the frame at the end of the robot is called the *flange frame*. In Section 3.5, a method for choosing the joint frames in a smart way is presented. The frames does not necessarily end up in the joints themselves. This is used to give a mathematical relation between the different coordinate frames.

The three last joints are often called a spherical wrist due to their orientation and another common term is the Wrist Center Point, WCP, which is situated where the three last joint axes intersect, i.e., in joint 5. If a tool is

mounted on the robot, the Tool Center Point (TCP) frame appears. It should be defined as the tip of the tool.

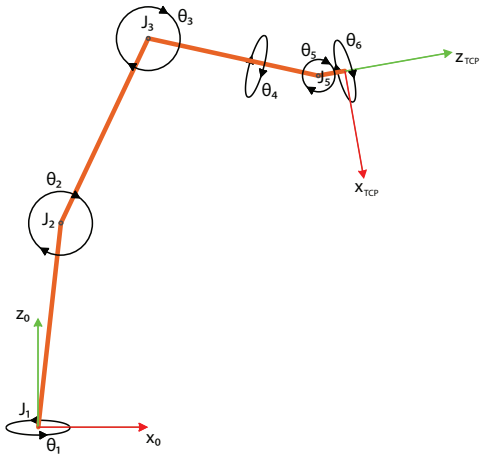

Figure 3.2 This is how the joint angles' positive directions are defined. The index 0 indicates the base frame and the TCP frame coincides with the flange frame as no tool is attached to the robot.

The IRB140B can be moved manually by using the Teach Pendant, a hand-held terminal for robot control, see Figure 3.3. This is called *jogging* the robot. To make the robot move at all the operator must hold the Teach Pendant's dead man's switch and release the brakes. The dead man's switch has a Off-On-Off functionality so that the robot stops both if the user squeezes or lets go of the switch.

Figure 3.3 The Teach Pendant.

In Figure 3.2 the definition of the six joint angles can be seen. For every angle θ_i, it holds that θ_i is constant when any of the other angles, θ_j, $j \neq i$, varies. However, this is not the case for all robots. The usual way of defining the joint angles in *ABB*'s robot series is that θ_3 is measured relative to a horizontal plane, or the xy-plane expressed in the robot base frame. This slight difference is easy to miss at first and can cause unnecessary confusion when working with the IRB series.

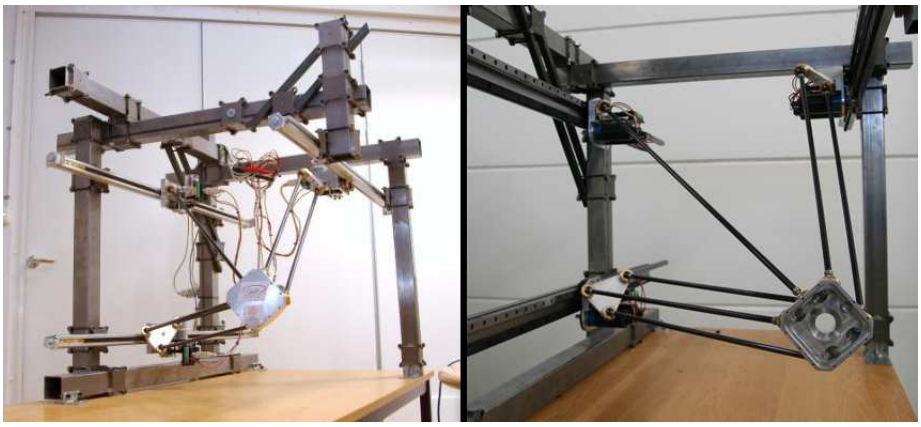

Figure 3.4 The Parallel Kinematic Robot T1 at the Robotics Lab at The Department of Automatic Control, Lund University.

3.2 Parallel Kinematic Robots

Another type of robots are the Parallel Kinematic Robots (PKR). In Figure 3.4 an example from the Robotics Lab at the Department of Automatic Control, Lund University, can be seen. It has four degrees of freedom, linear motion in all directions plus a rotation of the flange.

An upside of a robot of this type is that the moving parts of the robot are built very light in comparison to many other robots, e.g., the IRB140B in Section 3.1. From Newton's second law the following conclusion can be drawn. If the robot is built lighter, the accelerations due to a constant force exerted by its motors will be higher.

Parallel Kinematic Robots can be used with great success in situations where things should be picked from a rapidly moving conveyor belt.

If a Parallel Kinematic Robot should be used in a haptic master and slave system a device suitable is the Novint Falcon, see Section 2.1. They have similar constructions apart from the rotation possibility of the PKR flange that is missing on the Novint Falcon. A haptic device such as a Phantom, see Section 2.1, can also be used to control a PKR. Doing so, two of the rotational degrees of freedom for the Phantom can be neglected in the mathematical mapping to the slave robot to acquire full control of the PKR's movements.

If all the kinematics needed to describe a PKR fully is derived, it can replace for example the IRB140B as the slave robot in a master and slave system, as will be shown in Chapter 4.

3.3 Force Sensor

In order to establish a force controller some knowledge is needed about the force the environment affects the robot with. There are several approaches to achieve this and in this thesis a wrist sensor is used. A wrist mounted force/torque sensor is a mechanical structure equipped with, e.g., a strain gauge that measures force and torques acting on the end effector. It is attached between the flange and the end effector. However, it takes no consideration to whether the force is due to inertia, gravity or contact forces. It measures all

the forces acting on the force sensor no matter the underlying reason for the force.

The force sensor used is the *100M40* sensor from *JR3, Inc.*, [15]. It can be seen in Figure 3.5 and is mounted on the robot with the help of a tool changer, the *TK-50* from *IPR (Intelligent Peripherals for Robots)*, [13]. The force sensor itself is the blue part seen in Figure 3.5. In this thesis, two of the metal bars pointing outwards from the force sensor construction were removed so that the TCP could be defined as the tip of the sole metal bar that was left.

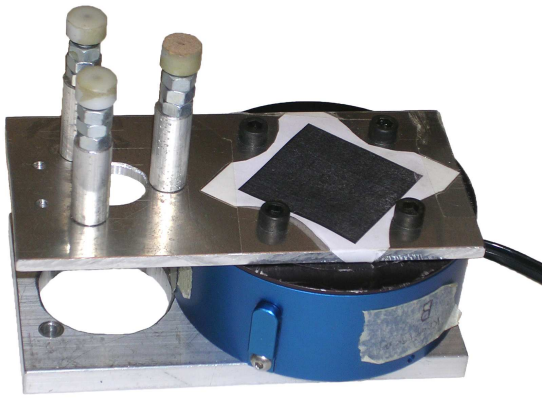

Figure 3.5 The force sensor 100M40 from *JR3, Inc.* that is used in this thesis. The sensor is the blue part in the construction.

3.4 Homogeneous Transformations

In robotics one often need to be able to shift between various coordinate frames. For example: when a tool is used it is much more convenient to work in a coordinate frame attached to the Tool Center Point (TCP) than in the world frame (in which the robot exists) or in the base frame (where the first joint is situated). A shift to another frame can be performed using a rotation matrix and a translation vector. A rotation matrix is a matrix containing the base vectors of the new frame's axes, described in the previous frame, as rows. In the same way a translation vector is a vector from the origin of the new frame to the origin of the previous frame described in the new frame.

A vector with its coordinates described in frame 0 will throughout this thesis be denoted p_0. To translate the vector p_0 into p_1, i.e., the same vector but instead described in frame 1, the following equation can be used:

$$p_1 = R_0^1 p_0 + d_1^0 \tag{3.1}$$

R_0^1 is a 3x3 dimensional matrix where each row corresponds to the base vectors of frame 1 described in frame 0. The matrix is thus a rotation matrix which in this case rotates p_0 to a new orientation. The vector d_1^0 is a vector from the origin of frame 1 to its counterpart in frame 0, see Figure 3.6. Throughout this thesis all base vectors will be ordered in a right-hand system and have a length of one.

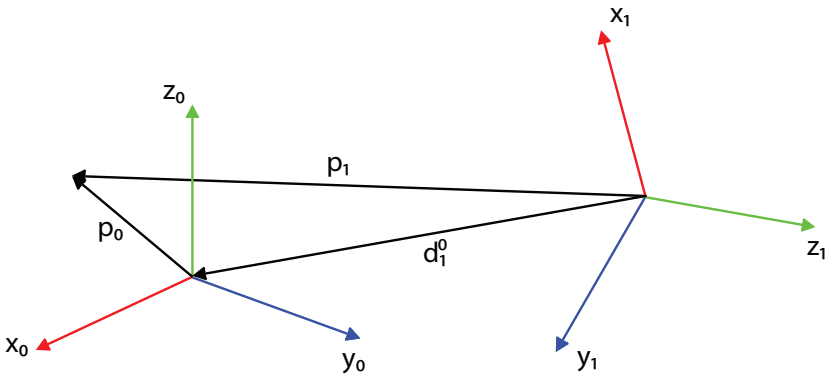

Figure 3.6 The vector p_0 has been rotated by a matrix R_0^1 and translated by the vector d_1^0 into p_1.

A simpler way of representing Equation 3.1 is to use a homogeneous transformation. It is a matrix that holds all the information needed for a rotation and a translation of a vector. A homogeneous transformation matrix will always be non singular. A homogenous transformation matrix is shown in Equation 3.2.

$$H_0^1 = \begin{bmatrix} R_0^1 & d_1^0 \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

If the vectors p_1 and p_0 are augmented with a fourth element equal to one, this together with Equation 3.1 and Equation 3.2 leads to Equation 3.3. Throughout this thesis the vector p will denote either the extended or the unextended vector depending on the context.

$$p_1 = H_0^1 p_0 \tag{3.3}$$

3.5 Denavit-Hartenberg Representation

The forward kinematics for a robot is determined by multiplying the homogeneous transformation matrices for the coordinate frames attached to the robot joints, see Section 3.6. The question how to choose these coordinate frames arises. One solution is the Denavit-Hartenberg representation. There are two alternative representations called the *standard representation* and the *modified representation*. Throughout this thesis the standard representation will be used [23].

The idea of both representations is that by choosing the coordinate frame for each joint by certain rules one can reduce the number of parameters needed to fully describe a joint from six to four. And furthermore, out of these four values only one is a variable hence called the joint variable. For a revolute joint the joint variable is the angle, θ, of the joint whereas for a prismatic joint the variable is the offset, d. The other two parameters are the twist, α, and the length, a, see Figure 3.7. The parameters for the IRB140B are presented in Table 3.1. The values for a and d were computed based on the IRB140B Product Specification [2].

The seven frames needed for the IRB140B were placed as shown in Figure 3.8. Due to the Denavit-Hartenberg rules the origin of a joint frame doesn't

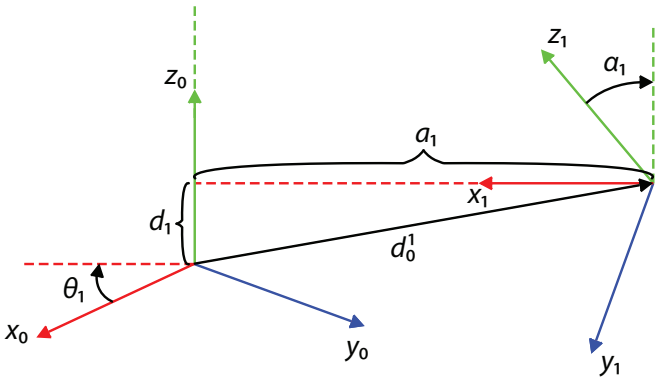

Figure 3.7 The parameters in the standard Denavit-Hartenberg representation, [23], are defined as shown.

$Link(i)$	a_i (mm)	α_i (degrees)	d_i (mm)	θ_i
1	70	$-90°$	352	θ_1
2	360	0	0	θ_2
3	0	$90°$	0	θ_3
4	0	$90°$	380	θ_4
5	0	$90°$	0	θ_5
6	0	0	d_6	θ_6

Table 3.1 Link parameters for the IRB140. $d_6 = 65$ if no tool is attached.

necessarily have to end up in the joint itself. It can in several cases be situated in another point or even in another joint. Therefore origins belonging to different joints can coincide which leads to problems when displaying them in the same figure which is the reason for using two different figures.

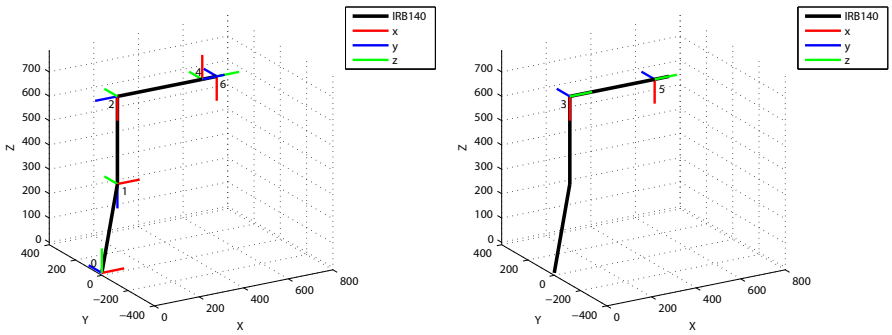

Figure 3.8 The index of the coordinate frame is one less than the corresponding joint index. The origins of frames 3 and 5 coincide with the origins of frames 2 and 4 respectively.

3.6 Forward Kinematics

Forward kinematics is about describing position, velocities and acceleration without taking forces and torques into account. The main goal is to relate the frame of the TCP to the base frame using the joint variables. To each joint a coordinate frame is attached. Multiplying the homogenous transformations for the frames step-by-step it is then possible to derive a homogenous transformation from the base frame to the TCP frame. To achieve a simple relation it is useful to follow a convention when assigning frames to the joints. In this thesis the standard Denavit-Hartenberg representation is used which results in that the relation between two frames is described by Equation 3.4. In the matrices the notations c_θ and s_θ are used. These are short for $\cos\theta$ and $\sin\theta$.

$$H_i^{i-1} = A_1 A_2 A_3 A_4 \tag{3.4}$$

$$A_1 = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix H_i^{i-1} describes the transformation from frame i to frame $i-1$. Using these matrices the position and orientation of the TCP, described in the base frame, can be calculated, see Equation 3.5. In a similar way all parts of the robot can be expressed in the base frame.

$$p_0 = H_1^0 H_2^1 H_3^2 H_4^3 H_5^4 H_6^5 p_6 \tag{3.5}$$

3.7 Inverse Kinematics

In Section 3.6, Equation 3.5 was derived. It gives the possibility to calculate the position and orientation of the TCP frame for a given set of joint angles. But what if the opposite situation occurs, i.e., given a position and orientation of the TCP frame you want to find the joint angles? That is what inverse kinematics is all about. In practice this means to solve the equations that relates the homogeneous transformation matrix H to the joint angles. H describes the TCP frame.

$$H = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equations 3.4 and 3.5 give a relation between r_{ij} and the joint angles. Solving these twelve nonlinear equations will generate the joint angles. However, these equations are nontrivial. Some constraints exist, e.g., the determinant of

a rotation matrix must be equal to one since the volume of the cube defined by the base axes does not change when a rotation is performed. Furthermore, the norm of each row or column in a rotation matrix part has to be one.

However, multiple solutions may still exist. One possibility of multiple solutions is illustrated in Figure 3.9. There are a number of different ways to solve these equations. One could for instance use a numerical algorithm or try to find the closed-form solution.

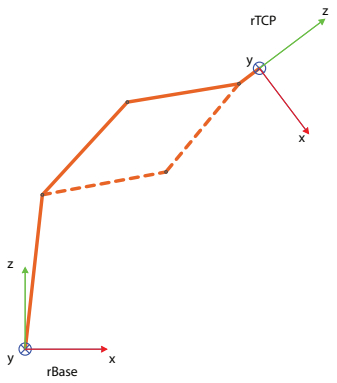

Figure 3.9 Two different ways of reaching the TCP with a certain orientation called elbow up and elbow down.

3.8 Jacobian

The Jacobian matrix, J, is mathematically defined as a matrix containing all partial derivatives of a vector valued function, see Equation 3.6 and 3.7.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2 \dots x_n) \\ \vdots \\ f_n(x_1, x_2 \dots x_n) \end{bmatrix} \tag{3.6}$$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{3.7}$$

$$\begin{bmatrix} \partial y_1 \\ \vdots \\ \partial y_n \end{bmatrix} = J(x_1, \cdots, x_n) \begin{bmatrix} \partial x_1 \\ \vdots \\ \partial x_n \end{bmatrix} \tag{3.8}$$

The relation of how a small change ∂x relates to small change ∂y is described in Equation 3.8. By dividing each side by the differential time element Equation 3.9 appears. The Jacobian matrix can therefore be seen as the mapping between the velocities of x and the velocities of y.

$$\begin{bmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_n \end{bmatrix} = J(x_1, \cdots, x_n) \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} \tag{3.9}$$

In robotics the Jacobian matrix is often denoted Jacobian and the Jacobian relates the link velocities with the frame velocities. The relation can be seen in Equation 3.10.

$$\nu_i^j = J_i^j(\theta)\dot{\theta} \tag{3.10}$$

$$\nu_i^j = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.11}$$

The first three elements in ν_i^j is the velocity for frame j expressed in frame i and the last three is the angular velocity for each corresponding axis for frame j. The Jacobian J_i^j is expressing the relation of the angular velocities of the joints, $\dot{\theta}$, and the velocity of frame j expressed in frame i. The Jacobian can be calculated in all given robot configurations and therefore the Jacobian will only contain numerical values.

For some set of joint angles the Jacobian will lose rank and this is called a singularity or a singular configuration. A loss in rank for the Jacobian is equivalent to the inverse of the Jacobian being undefined. If there is a singular configuration there are several interpretations on why this occur. Singularities appear where some robot motions are unachievable. At a singularity there is infinitely many or no solutions to the inverse kinematics problem, which makes the robot very sensitive near singularities. This will cause a problem if the robot path is calculated using the Jacobian. Further information about Jacobians can be found in [23] and [6].

3.9 Impedance Control

There exist many different methods for controlling a robot in different kinds of situations. For industrial robots it is common to look at position control and force control. To apply position control to move the robot to the right spot and then use force control when contact forces emerge. That is basically what is done in hybrid control. A switch of regulators must then be done at the proper moment. How to make that switch is not trivial. The advantage of impedance control is that it will manage both position and force control at the same time, [10], [11] and [12]. Therefore, a switch of regulators is not needed. In this thesis impedance control is used.

The idea of impedance control is to make the robot's TCP behave as a spring-mass-damper system, see Figure 3.10. The spring-mass-damper system is described by Equation 3.12, where M represents the mass, D the damping coefficient and K the spring stiffness.

$$F = M\ddot{x}_{ref} + D(\dot{x} - \dot{x}_{des}) + K(x - x_{des}) \tag{3.12}$$

The force F is what the environment is affecting the robot's TCP with. The desired position of the system is x_{des} and $\dot{x}_{des}$ is the desired velocity. The robot's TCP position and velocity are expressed as x and $\dot{x}$ respectively.

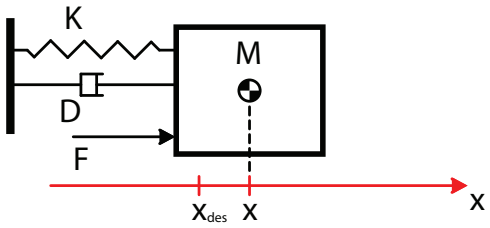

Figure 3.10 A simple spring-mass-damper system affected by an external force, F. The external force is causing the system to move its equilibrium point from x_{des} to x.

If $\ddot{x} = \ddot{x}_{ref}$, where $\ddot{x}$ is the robot's TCP acceleration, the robot will behave like the spring-mass-damper system in Figure 3.10. Using Equation 3.12 leads to the control law seen in Equation 3.13.

$$\ddot{x}_{ref} = \frac{F - D(\dot{x} - \dot{x}_{des}) - K(x - x_{des})}{M} \tag{3.13}$$

By defining the M, D and K parameters and using the current measured and desired values the acceleration reference can be calculated using Equation 3.13. By integrating that acceleration one or two times the velocity and position can be derived, see Equations 3.14 and 3.15. If the robot is set to follow the derived position and velocity signals it will behave as Equation 3.12 and Figure 3.10 implies.

$$\dot{x}_{ref} = \int \ddot{x}_{des}\, dx \tag{3.14}$$

$$x_{ref} = \iint \ddot{x}_{des}\, dx \tag{3.15}$$

But in order to implement the proposed control law it has to be discretized. A discretization with a sampling rate of h and a discrete integration of a signal $u[k]$ defined as $\sum u[k]h$ is used. By assuming that the robot's TCP velocity is zero and that the TCP position is $x[0]$ at k=0, Equations 3.16 and 3.17 are derived.

$$\dot{x}_{ref}[k] = \sum_{i=0}^{k} \ddot{x}_{des}[i]h \tag{3.16}$$

$$x_{ref}[k] = x[0] + \sum_{i=0}^{k} \dot{x}_{ref}[i]h \tag{3.17}$$

Thus, the implementation of a discrete impedance controller, defined in Equations 3.16 and 3.17, can be used to make a robot behave as a spring-mass-damper system.

Choosing the parameters M, D and K is not a trivial task as it is hard to imagine exactly how a spring with a stiffness, K, or a damper with a damping coefficient, D, actually behaves. Therefore the characteristic polynomial $Ms^2 + Ds + K$ is compared to $s^2 + 2\zeta\omega s + \omega^2$. That results in Equations 3.18 and 3.19. ω represents the eigenfrequency of the spring-mass-damper system and ζ is the damping. The advantage of expressing the damping in terms of ζ instead of D

is that $\zeta \in [0, 1]$ where zero means no damping and 1 means critical damping, i.e., no overshoot in a step response.

$$D = 2M\zeta\omega \tag{3.18}$$

$$K = M\omega^2 \tag{3.19}$$

4. Mapping a Haptic Device to a Robot

In order to control a robot's movements with an external control device some mapping have to be defined. In this thesis the external control device is a haptic device. The relation between the haptic device and the robot can be referred to as a master and slave relation. The master's position and orientation controls the position and orientation of the slave.

This chapter will explain in detail how the mapping between the Phantom and the IRB140B is done in this master thesis. The mapping will be explained through mathematical and geometrical interpretations of how the master and slaves relates to each other. However, the proposed mapping could be used in any general master and slave relation and not only in the Phantom and IRB140B case. More specifically, any haptic device could use this mapping, e.g., the Novint Falcon, see Section 2.1.

4.1 Frames and Transformations

When connecting a haptic device to a robot, virtual or real, various coordinate frames appear. There are six different frames used, see Figure 4.1. The four main frames are the base frame and the TCP frame for both the robot and the haptic device. The haptic TCP (hTCP) is defined as the tip of the haptic "pen" on the Phantom. The haptic base frame (hBase) has its origin in the center of the haptic workspace. The robot base frame is denoted rBase whereas the robot TCP frame is denoted rTCP.

Because of the close relation between the frames rTCP and hTCP the latter has been placed in the same point as the first, i.e., in the robot TCP, see Figure 4.2. The haptic base frame (hBase) is placed in the position where the robot TCP is located at the startup. This creates an opportunity to easily work in different areas of the robot's workspace and not just in the proximity of the TCP in the configuration where $\theta_i = 0$ or in any other predetermined start configuration.

Transformations between the frames in Figure 4.1 are needed to make the robot behave in a desired manner according to the haptic device state, see Figure 4.3. A transformation from rTCP to rBase, H_{rTCP}^{rBase}, is used in the inverse kinematics problem, see Section 3.7, to find the robot angles needed to describe the desired state determined by the haptic device. Using homogenous transformations as described in Section 3.4, Equation 4.1 gives the transformation needed in the inverse kinematics.

$$H_{rTCP}^{rBase} = H_{hBase}^{rBase} H_{hTCP}^{hBase} H_{rTCP}^{hTCP} \tag{4.1}$$

The origins of hTCP and rTCP always coincide and their orientation to each other always remains the same. This makes it easy to define the therefore

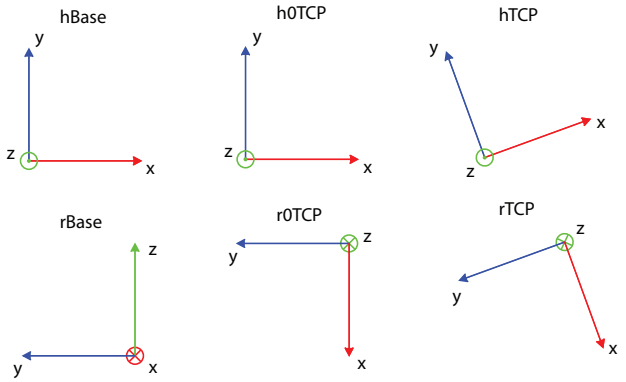

Figure 4.1 The different frames used in the mapping. h0TCP and r0TCP are the startup versions of hTCP and rTCP. The transformations between hTCP and rTCP are always constant. The origins of hBase, h0TCP and r0TCP coincide. The four frames on the left are viewed from the rear of the robot for a start configuration $\theta_i = 0$. The two on the right are simply reoriented after startup.

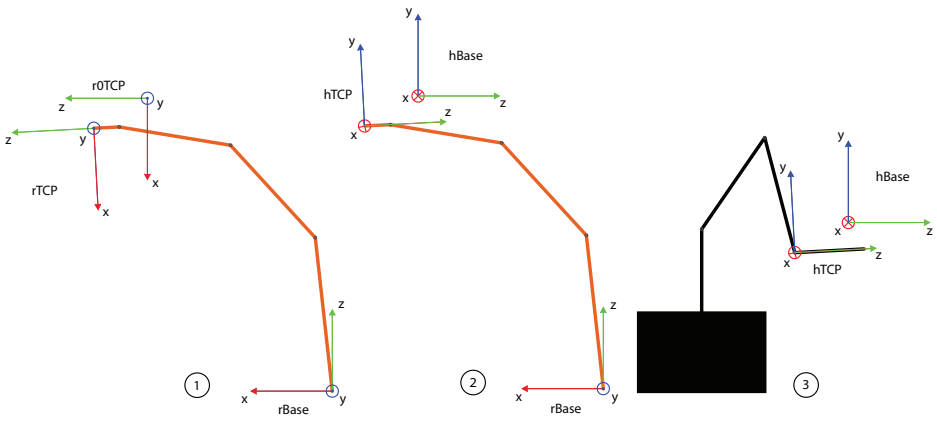

Figure 4.2 The haptic and robot frames. 1 and 2 are robots with a configuration determined by 3, the Phantom. The position of hBase and r0TCP described in rBase is determined by the starting robot configuration. Keep in mind that any haptic device and any robot could have been used in this mapping, not just the Phantom and a robot as the one in this figure.

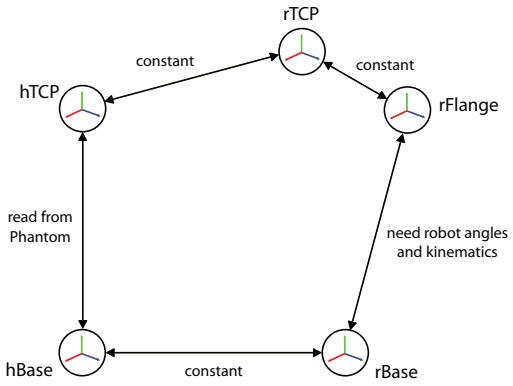

Figure 4.3 The paths available between the haptic and the robot frames.

constant transformation between them, see Equation 4.2.

$$H_{rTCP}^{hTCP} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Out of the three transformations on the right-hand-side in Equation 4.1 two remain to be determined. It is assumed that H_{hTCP}^{hBase} can be read from the haptic device. H_{hBase}^{rBase} on the other hand can be evaluated at startup due to the previously mentioned fact that the position and orientation of the hBase frame depends on the robot's starting angles and thereafter remain constant in the eyes of the rBase frame. Equation 4.3 shows the general way of computing H_{hBase}^{rBase}.

$$H_{hBase}^{rBase} = H_{rTCP}^{rBase} H_{hTCP}^{rTCP} H_{hBase}^{hTCP} \tag{4.3}$$

At startup however, Equation 4.3 is simplified as presented in Equation 4.4.

$$H_{hBase}^{rBase} = H_{r0TCP}^{rBase} H_{h0TCP}^{r0TCP} H_{hBase}^{h0TCP} = H_{r0TCP}^{rBase} H_{hTCP}^{rTCP} I = H_{r0TCP}^{rBase} H_{hTCP}^{rTCP} \tag{4.4}$$

The extra 0 in the indices implies a startup configuration of a frame. H_{r0TCP}^{rBase} can be calculated using forward kinematics for the robot's starting angles, see Section 3.6, and H_{hTCP}^{rTCP} is defined in Equation 4.2. That $H_{hBase}^{h0TCP} = I$ can be realized by looking at Figure 4.1.

4.2 Bumpless Switching

When using the mapping proposed in Section 4.1, the haptic base frame (hBase) is assumed to be located at the robot's TCP frame (rTCP) at startup. However the mapping indicates that the rTCP frame is mapped to the hTCP frame. Therefore it also assumes that the hTCP frame is the same as the hBase frame at startup. However this is rarely the case, as the position and orientation of the Phantom tip is likely to differ from the hBase frame even if the user tries to hold it as close to the hBase frame as possible. It is undesirable to make the robot attempt a jump in position or orientation in order to achieve the proposed mapping. To avoid this problem the current hTCP frame will always be mapped to the rTCP frame in the mapping initialization. This means that the current hTCP frame is mapped to the r0TCP frame at startup. A way of achieving this is to use the matrix $A = (H_{hTCP}^{hBase})^{-1}$ in the initialization and when the mapping begins, multiply the current H_{hTCP}^{hBase} with A from the left which leads to Equation 4.5. The matrix H is then interpreted as the H_{hTCP}^{hBase} matrix and is used in the same way as described in Section 4.1.

$$H = A \cdot H_{hTCP}^{hBase} \tag{4.5}$$

Another way of expressing Equation 4.5 is in terms of h0TCP. In fact, the hBase frame is always situated in the same place, what changes is where the h0TCP frame is put, see Equation 4.6. The matrix H_{hTCP}^{h0TCP} is interpreted as the H_{hTCP}^{hBase} matrix in Section 4.1.

$$H_{hTCP}^{h0TCP} = H_{hBase}^{h0TCP} \cdot H_{hTCP}^{hBase} \tag{4.6}$$

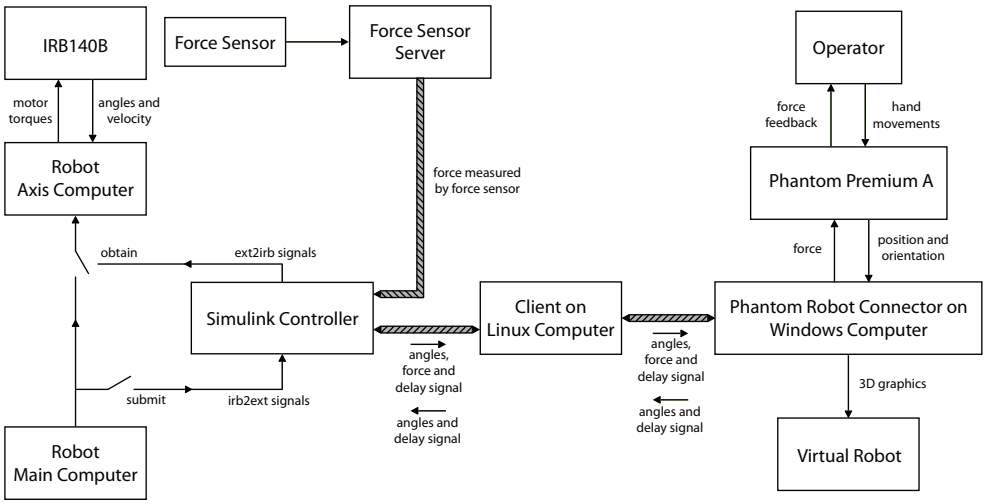

Figure 5.1 An overview of the setup. The thicker connections indicates a TCP/IP connection via a local network.

5. How to Connect the Phantom to the IRB140B

The Phantom is connected to the IRB140B as shown in Figure 5.1. For more details on how the robot main and axis computers cooperate with the Simulink controller, consult [7]. Briefly, the Simulink controller runs the result of a Simulink model built with Simulink's Real-Time Workshop and downloaded to the robot system, see Section 5.2.

The *Phantom Robot Connector* is a computer program written in C++ that is running on a Windows computer, see Chapters 8 and 9. The Phantom is connected to that computer through a PCI-card. The client on the Linux computer serves as an interpreter between the *Phantom Robot Connector* and the Simulink controller. The thick wires in Figure 5.1 indicates that data is sent over a TCP/IP network, see Section 5.1.

There is a graphical user interface (GUI) that can be run on the computer that the robot's two serial ports are connected to, [7], see Figure 5.2. In the GUI, the user can choose to select, in the order as presented here, *submit* and then *obtain*. The Simulink controller will affect the robot first when obtain is active. It is possible to define a parameter in the Simulink controller as changeable during runtime. If so, the parameter can then be altered from the GUI.

The force sensor is mounted on the robot but sends data to a separate server. The *Virtual Robot* block is simply a computer screen displaying the graphics determined by the graphics part of the *Phantom Robot Connector*. The *Operator* is the human controlling the haptic device.

As can be seen in Figure 5.1, there are a lot of different systems and programs that have to operate simultaneously. Therefore a guide was developed that goes through the correct order in which to start every process, [8]. It also presents some advices and cautions to make things easier for the operator.

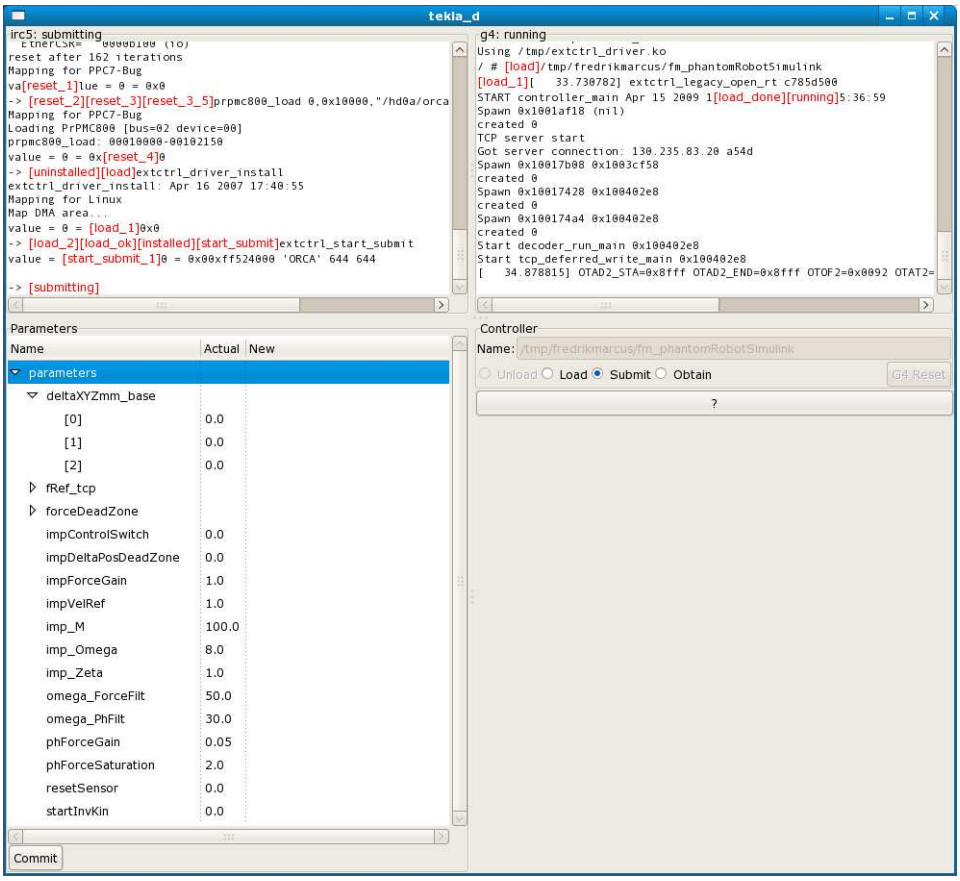

Figure 5.2 The graphical user interface in which a Simulink model can be loaded and its parameters altered.

5.1 TCP/IP Communication and Endianness

To establish communication between the *Phantom Robot Connector* and the Simulink controller a TCP/IP connection was chosen. Depending on which platform the *Phantom Robot Connector* is run on, different implementations of the TCP/IP connection are needed, see Section 9.4.

There is a problem that may arise when sending data over network byte by byte. What happens if a computer wants to send a value, for example a `double`, to another computer and they don't use the same order when storing bytes? There are two different conventions when it comes to storing data in the memory of a computer called big-endian and little-endian, [5]. The difference is that big-endian stores the highest ordered byte in the memory location with the lowest address, i.e., the most significant byte is stored first. Little-endian does the opposite so that the least significant byte comes first in the memory. Which endianness is used depends on the processor. This is not a problem in everyday usage but when sending data byte by byte a difference in endianness can corrupt the value sent. If this is the case, the bytes can be sent in reverse order to solve the matter.

5.2 Using Simulink in a Network With Labcomm

A model was created in Simulink, see Chapter 7. To be able to use that model for robot control, it has to be built by Simulink's Real Time Workshop. What this does, is that it generates and compiles C code that is downloaded to the robot's main computer [7]. To enable the *Phantom Robot Connector* to send data to the in ports in the model and to collect data from the out ports Labcomm is used. In a separate Labcomm file `filename.lc`, the input and output data types are defined as shown below.

```
sample float ph2RobAngles[6];
sample float rob2PhAngles[6];
sample float rob2PhForce[3];
sample float ph2RobDelay;
sample float rob2PhDelay;
sample float jr3_comedi[6];
```

A Labcomm command then creates `filename.c` and `filename.h`. These files contain functions for sending and reading the data defined in `filename.lc` to and from the built Simulink model on the robot computer from within a C or C++ program via a TCP/IP connection. These functions are written for Linux. There is currently no counterpart written for Windows. More on useful commands and how to integrate the mentioned files with the Simulink model can be found in [8].

6. Calibration of Tool and Sensor

By default the IRB140B is controlling the flange frame. If a tool is attached it has to be defined, for example by using the Teach Pendant, [3]. Selecting the newly defined tool in the Teach Pendant will then allow the operator to jog the robot according to the TCP frame instead of the flange frame. The new tool must also be defined in the Simulink model, see Section 7.3, so that the information sent to the *Phantom Robot Connector* will be correct. Here, a force sensor is part of the tool. The force sensor frame must therefore also be defined.

6.1 Force Sensor

The JR3 force sensor described in Section 3.3 needs to be calibrated when mounted on the flange of the IRB140B, i.e., a transformation matrix between the force sensor frame and the flange frame should be determined. This transformation matrix is used when calculating the force acting on the tool based on the data from the force sensor. A method of determining the transformation matrix will be presented below.

First, the rotation matrix from the flange frame to the sensor frame must be derived. The robot was jogged manually to a certain position and the JR3 force sensor was reset. A weight was hung from the tip of the tool. The force sensor then measures the gravitational force of the weight. This means that the force can be written in the robot base frame as

$$F_{base} = \begin{bmatrix} 0 \\ 0 \\ -f \end{bmatrix} \tag{6.1}$$

The force is however measured in the force sensor frame as

$$F_{sensorFrame} = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \tag{6.2}$$

f in Equation 6.1 is determined by Equation 6.3

$$f = \sqrt{f_x^2 + f_y^2 + f_z^2} \tag{6.3}$$

The relation between Equation 6.1 and 6.2 is given in Equation 6.4. $R_{flange}^{sensorFrame}$ and R_{base}^{flange} are rotational matrices where the latter is derived using forward kinematics.

$$F_{sensorFrame} = R_{flange}^{sensorFrame} \cdot R_{base}^{flange} \cdot F_{base} \tag{6.4}$$

The force sensor is attached to the flange in such a way that the z-axis of the force sensor frame coincides with the z-axis of the robot flange frame, but

oriented in the opposite direction. This leads to Equation 6.5.

$$R_{flange}^{sensorFrame} = \begin{bmatrix} a & b & 0 \\ -b & a & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{6.5}$$

Since the force sensor measures the force it is exerting on the environment the following slight adjustment was made

$$F_{sensorFrame} = -F_{measured}$$

Equation 6.4 results in three equations. The first equation gives an expression for determining a and b in Equation 6.5. The second equation will be the same as the first, whereas the last equation will be trivial. To get a complete set of equations the experiment was repeated for another configuration of the robot. This procedure can be, but was not, done numerous times in order to acquire a least squares approximation.

In a rotation matrix, the rows represent the base vectors of the new frame expressed in the old frame, see Section 3.4. Hence, for each row (and column) in a rotation matrix it holds that their norm equals to one. However, this was not the case for the rows and columns when the derived equation system of order two above was solved in Matlab. a and b were therefore divided by the norm of for example, the first row. The resulting matrix is found in Equation 6.6. Possible reasons for the rows not to have a norm of one naturally are the resolution in the force measurements and the resetting of the sensor functioning poorly, see Section 7.4.

$$R_{flange}^{sensorFrame} = \begin{bmatrix} -0.7894 & 0.6139 & 0 \\ -0.6139 & -0.7894 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{6.6}$$

Second, the cartesian distances from the flange frame to the sensor frame was measured using a vernier caliper to $d = [0, 0, 56.7]$ mm. The sensor frame has its origin in the center of the sensor. Because of the difficulty in finding and measuring from the center, the distance to the end of the sensor was determined as 76.7 mm. In a data sheet for the JR3, the distance from the end to the center is stated to be 20.0 mm.

The translation vector, d, in the transformation matrix from the force sensor frame to the flange frame, see Section 3.4, is only used if the torques measured by the force sensor should be used. Otherwise, to shift frames for a force, only the rotation matrix part of the homogeneous transformation matrix is used.

6.2 Tool

The tool was defined using the Teach Pendant. Following step-by-step instructions in [3] will result in a definition of the transformation from the flange frame to the TCP frame. The method used is a so called four-point-calibration. Basically, the robot is jogged to four different configurations where each configuration makes the tool center point end up in the same point in the workspace.

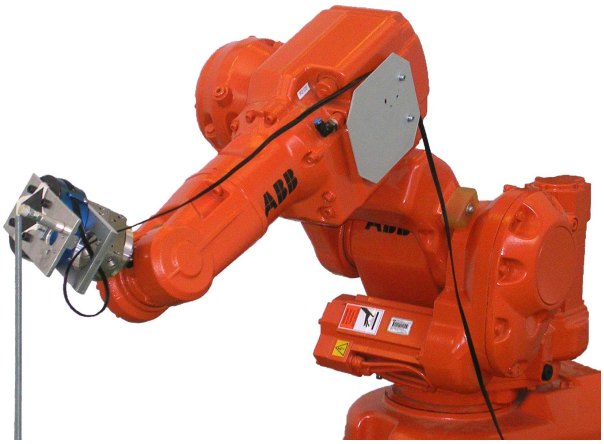

Figure 6.1 One of the four configurations the robot was set to in order to perform a four-point-calibration of the tool center point.

Using data from the four configurations, the translation part of the desired transformation matrix can be calculated. A four-point calibration will orient the TCP frame in the same way as the flange frame is oriented. To change the orientation, other methods described in [3] have to be used.

To define the calibration point in the workspace a metal stick was used. The calibration point was placed in mid-air half a centimeter above the stick to enable access from several different angles. In Figure 6.1 one of the four robot configurations is shown. It may seem a bit rough to use a calibration point as the one described above. During this thesis, no known structures will however be dealt with, so it will not be crucial to have a very small error in the TCP position. The four-point-calibration resulted in the homogeneous transformation matrix in Equation 6.7. The unit in the translation vector is millimeters.

$$H_{flange}^{TCP} = \begin{bmatrix} 1 & 0 & 0 & 50.82 \\ 0 & 1 & 0 & 64.51 \\ 0 & 0 & 1 & 173.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.7}$$

7. Simulink Model

A model created in Simulink can be used for communication with a robot, see Section 5.2. The model created in this thesis, see Section 7.3, has two roles. It serves to feed the robot and the *Phantom Robot Connector* with data from one another but it also performs calculations on its own regarding the force control, see Section 7.5.

Used in the model is a Simulink library called *ExtCtrl Library* that contains Simulink blocks for various calculations needed in robotics applications. The library blocks are based on the same C code that is used in Chapter 9. A new Simulink library was also created, holding blocks that were needed more than once in the full scale model, see Section 7.2.

Many different signals are available for communication with the robot. There are for example a number of different position, velocity and torque signals. These signals hold information about the joint angles and not a specific coordinate frame. A position signal for example, is therefore containing the joint angles, measured in motor side angles. Before they are at any use, they must be transformed into arm side angles by a Simulink block from the *ExtCtrl Library*. For this reason, whenever a position or velocity signal is mentioned in this chapter, it is assumed that if it originally contains motor side angles, they have been transformed into arm side angles. There are also signals concerning parameters in the robot's own controller and a signal containing the force measured by the force sensor attached to the robot.

The signals' names tell whether the signal is coming from or going to the robot. Most signals have either *irb2ext* or *ext2irb* as a prefix, *irb* being the robot and *ext* referring to external control, i.e., the Simulink model.

7.1 Position Signals

Since the *Phantom Robot Connector* uses the robot's joint angles to perform it's calculations, see Chapter 9, a clever choice of which position signal should be used is called for. Figures 7.1 and 7.2 show how the two signals *irb2ext.posFlt* and *irb2ext.posRaw_Abs* act when the robot is performing a step on the first joint angle. The step back and forth was induced by jogging the robot manually on the teach pendant. From the figures the following conclusion can be drawn: *irb2ext.posFlt* is filtered too heavily for the application in this thesis. However, to use *irb2ext.posRaw_Abs* instead will also lead to problems. *irb2ext.posRaw_Abs* contains, as the name imply, the position signal directly from the joint angle encoders. Using it as a reference to the *Phantom Robot Connector* will cause any disturbances entering in the measurements to affect the reference sent to the robot from the *Phantom Robot Connector* directly.

A better signal to use as a reference to the *Phantom Robot Connector* is *irb2ext.posRef*. It is a purely mathematically induced signal, i.e., it is not based on measurements as the position signals previously mentioned. It comes from the robot's main computer and will change when the robot is jogged manually. However, it will not change if the other position reference signal, *ext2irb.posRef*, is varying, see Figure 7.3. So even if changes in *ext2irb.posRef* will cause the robot to move, these changes will not affect the *irb2ext.posRef* signal, see Figure 5.1. This fact is causing a problem that will be treated in

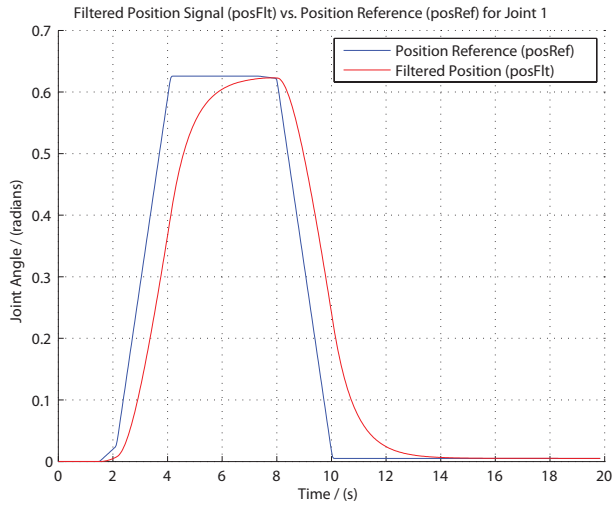

Figure 7.1 A filtered position signal.

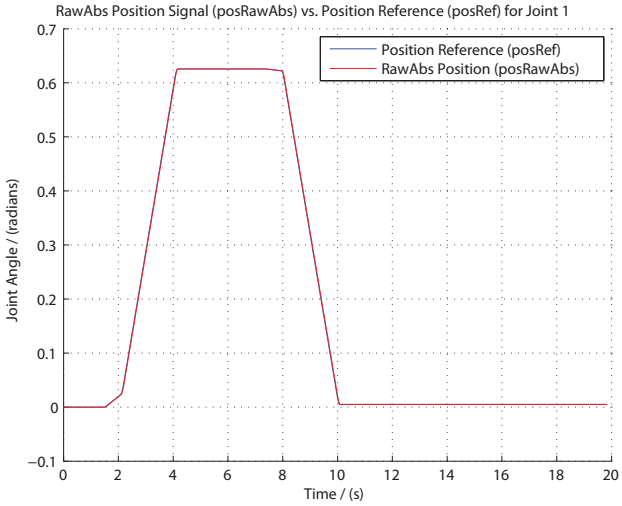

Figure 7.2 The raw position signal follows the position reference well, evidence of the robot's good controllers.

Section 7.8.

7.2 Simulink Library

The advantage of gathering functions needed in more than one place in a library is that if they need some adjustments, these only have to be done in the library. A Simulink model using blocks from a library will update them at startup. Apart from the *ExtCtrl Library*, that was available from the beginning, another library was created in order to simplify working with the full model, see Figure 7.4. The contents of each block can be found in Appendix C.1.

The ordinary Simulink subsystem *Forward Kinematics – arm joints -> tcp* encapsulates blocks from the *ExtCtrl Library*. The subsystem will return the

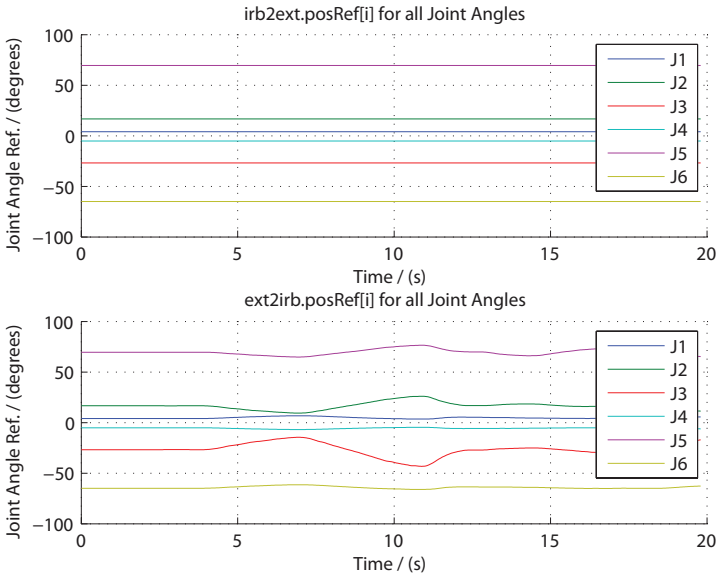

Figure 7.3 The signal *irb2ext.posRef* does not respond to changes in *ext2irb.posRef*.

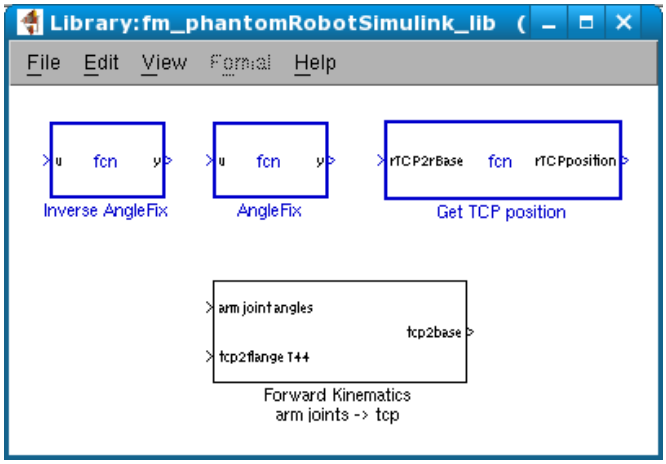

Figure 7.4 The created Simulink library.

transformation from the robot TCP to the robot base instead of from the flange to the base.

The *Angle Fix* and *Inverse Angle Fix* are used to transform the third joint angle to relative the horizontal plane form and back to normal IRB140B form respectively, see Section 3.1. The reason this has to be done is that the C code in the *ExtCtrl Library* treats the IRB140B in the same way as the other ABB robots, i.e., the third joint angle is considered to be defined relative the horizontal plane.

The third embedded Matlab function in Figure 7.4 will return the translation vector, d, in a given homogeneous transformation matrix on row-major form.

7.3 The Model

During the thesis work many different tests were performed using a Simulink controller, for example the solution to the problem with bumpless switching problem that is presented in Section 7.8, leading to different appearances of the Simulink model. The final version of the Simulink model can though be seen in Figure 7.5. The sample time used is $h = 0.004$ seconds. Many times throughout the remainder of this chapter, different inports, outports and subsystems will be mentioned. These can, if no other specific reference is given, always be found in this figure.

The force sensor signal treatment is a big part of the Simulink model why it has been given its own section in Section 7.4. The block *Impedance* is presented in Section 7.5 whereas the subsystems called *Position Reference* and *Velocity Reference* are described in Sections 7.6 and 7.7 respectively.

Some parameters in the Simulink model has been made changeable from the GUI, see Figure 5.2. A complete list over all the parameters and their function is included in [8].

7.4 The Force Sensor Signal

The force sensor signal is received through the in port *jr3_comedi*. It is measured in the force sensor frame, from here on referred to as the sensor frame, see Section 6.1. Before it is at any use, methods for resetting it, transferring it to the TCP frame and preparing it for the Phantom must be implemented. This is done in the subsystem called *Treat Force Signal* in Figure 7.5. The contents of the subsystem is shown in Figure 7.6.

Resetting the force sensor is crucial at startup as the force sensor has quite a big offset due to the weight of itself and the construction it is attached to, see Figure 3.5. It will however also be necessary to reset the force sensor if the robot's TCP is reoriented. That is due to the sensor frame's reorientation compared to the gravitation force vector. There is another version in the *ExtCtrl Library* of the subsystem in Figure 7.6 called *offset correction/reset* that will compensate for the gravitational forces acting on the tool. It is not crucial to do this since the force sensor is not reoriented while controlling the robot with the Phantom, the more simple version of the subsystem was used.

First, two ways to reset the force sensor signal are presented. There is an inport called *ph2RobDelay* and an outport called *rob2PhDelay*, see Section 5.2. These are used by the *Phantom Robot Connector* to measure the time it takes for a round trip back and forth to the robot, or in reality the Simulink Controller, see Section 9.7. Briefly, the idea is that the delay signal is shifting between two values, either zero and one, or two and three. The default is zero and one but when requested by the user, that can change to two and three instead and from there back again to zero and one. That fact can be used to reset the force sensor. Whenever a shift between the pair of values sent is induced the force sensor will be reset.

The purple subsystem, called *offset correction/reset*, in Figure 7.6 was available, [18]. It will reset the force sensor signal when given a positive flank in its in port *reset_at_pFlank*. That positive flank can be induced either by altering the parameter *resetSensor* from the GUI or through altering the delay signal from the *Phantom Robot Connector* as described above. In the latter

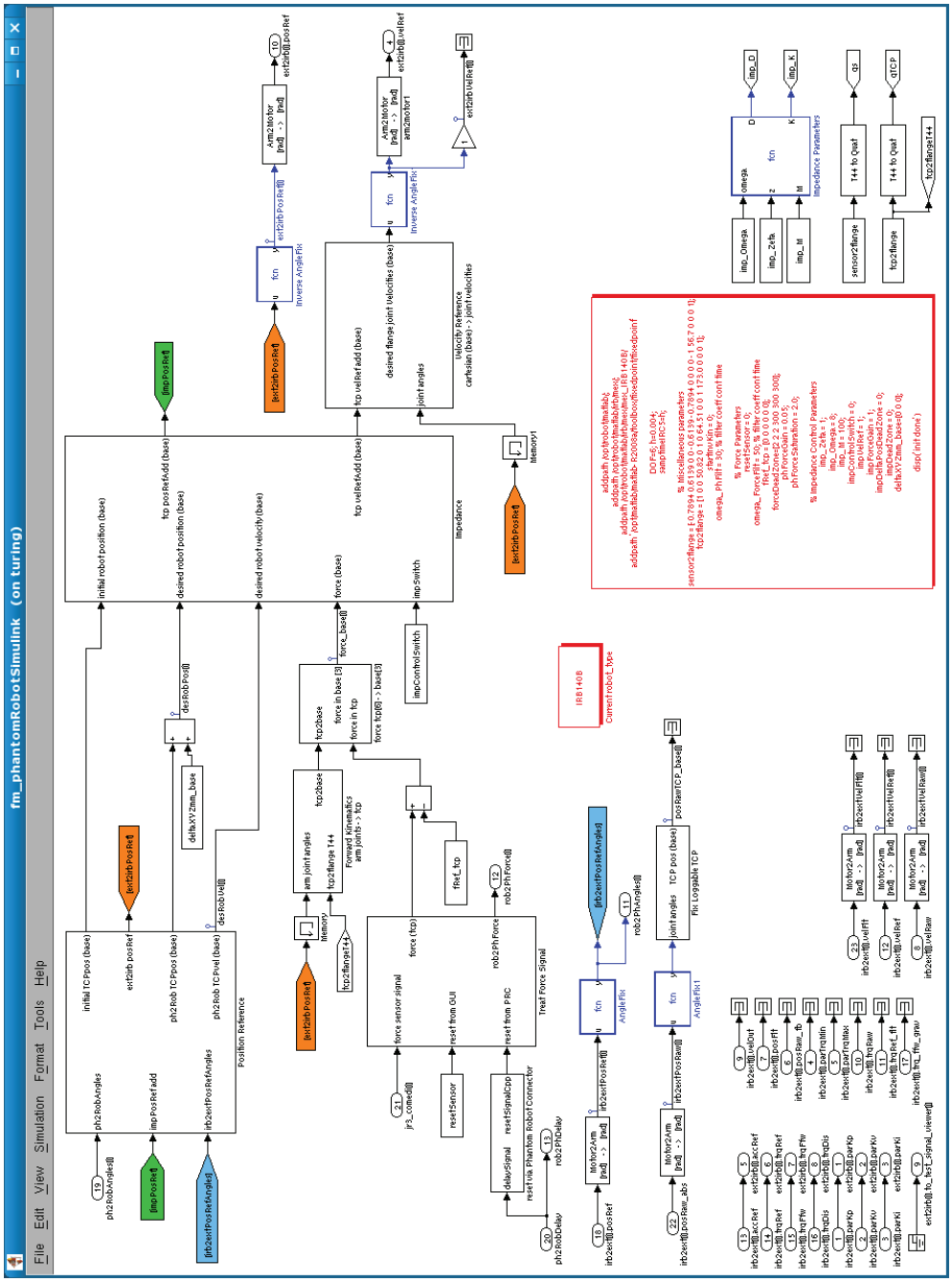

Figure 7.5 The Simulink model used in this thesis.

case the subsystem *reset via Phantom Robot Connector* comes in, see Figure 7.7. The idea is that a zero will be sent out at all occasions except for when the two last values, after having subtracted 1.5, have different signs. That indicates that the user has shifted pairs in the delay signal, i.e., the user wants to reset the force sensor signal.

The resetting of the force sensor does not give a perfect result. A reason for this might be the force sensor quantization and resolution rather than the logics behind the resetting.

The orange subsystem, called *Force sensor -> TCP*, in Figure 7.6 was also

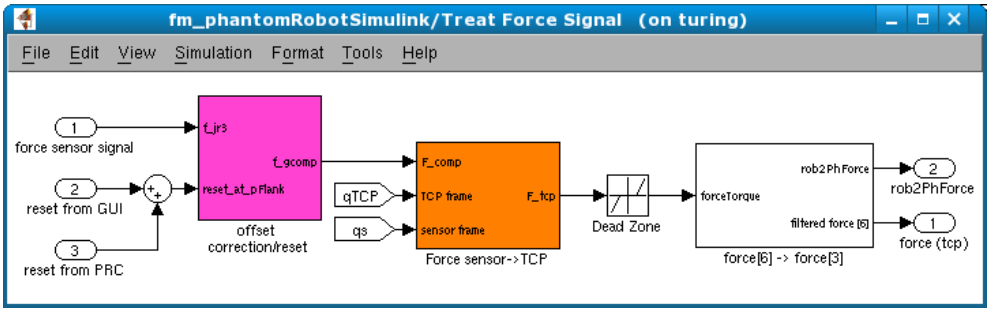

Figure 7.6 This subsystem handles the resetting, the transformation and the filtering of the force sensor signal. The outputs are the force that should be sent to the Phantom and a filtered force signal.

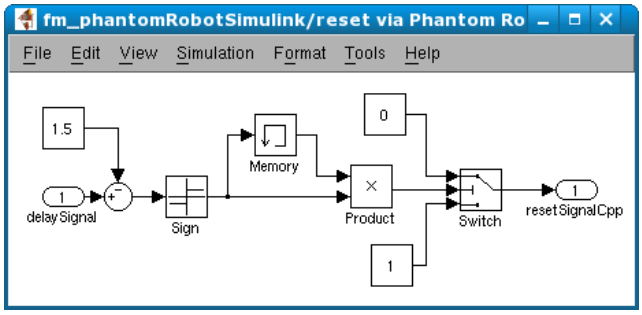

Figure 7.7 This subsystem will render a value that can be used to reset the force sensor. The criteria for passing input port one in the switch block is that $u_2 > 0$.

available from start, [18]. It will transform a force sensor signal from the sensor frame to the TCP frame by using some blocks from the *ExtCtrl Library*.

After a reset at startup, or after a reorientation, and a transformation to the TCP frame, the force sensor signal is treated some more in the subsystem called *force[6] -> force[3]*, see Figure 7.8. The name refers to the fact that only the forces and not the torques will be sent to the *Phantom Robot Connector*. However, the second out port of the subsystem contains a filtered force signal of size 6. As can be seen in the figure, the torques are not filtered. The reason is simply that they will not be used and are only brought along to enable the use of a force transmission block from the *ExtCtrl Library* block later on. The *phForceGain* and the saturation limit, *phSaturationLimit*, can be set from the GUI. The default values are 0.05 and 2.0 respectively.

The discrete low pass filter in Figure 7.8 has a transfer function as shown in Equation 7.1.

$$G = \frac{1 - e^{-\omega h}}{z - e^{-\omega h}} \tag{7.1}$$

The continuous low pass filter constant, ω, can be altered from the GUI through *omega_ForceFilt*. An example of the impact of the filter can be found in Figure 7.9 where $omega\_ForceFilt = 30$. There will be a slight delay in the force measurements due to this filter and in an attempt to minimalize that delay but still keep the disturbances in the signal away, the continuous filter constant was set to $omega\_ForceFilt = 50$.

To prepare the force signal for the impedance controller it must be expressed in the robot's base frame. This is fixed by a *Force Transmission* block

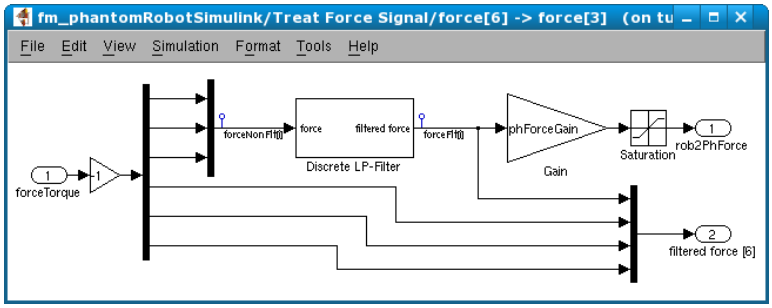

Figure 7.8 The force signal is scaled down and saturated to prevent a strange behavior of the Phantom. The reason for the minus one gain is the force sensor measuring the force it exerts on the environment. Notice that the torques are not filtered as they will not be used further on.

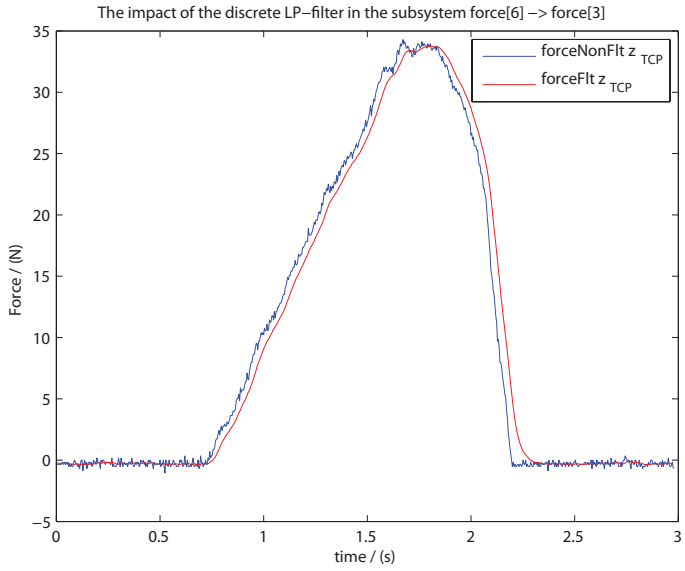

Figure 7.9 The impact of the discrete low pass filter in the subsystem in Figure 7.8 when the filter constant $\omega = 30$.

from the *ExtCtrl Library* hidden in the subsystem called *force tcp[6] -> base[3]* in Figure 7.5.

7.5 Implementation of Impedance Control

The force control method used in this thesis is impedance control, see Section 3.9. An implementation in Simulink can be seen in Figure 7.10. The feedback loops in this figure indicates that the impedance controller does not receive any information about the actual state of the robot, but only about the desired state of the robot. It is assumed that the robot's own controllers does a fine job making sure the robot will follow the trajectories asked for. An evidence of this can be seen in Figure 7.2. Therefore, to prevent any unnecessary disturbances to enter the impedance controller, these two feedback loops

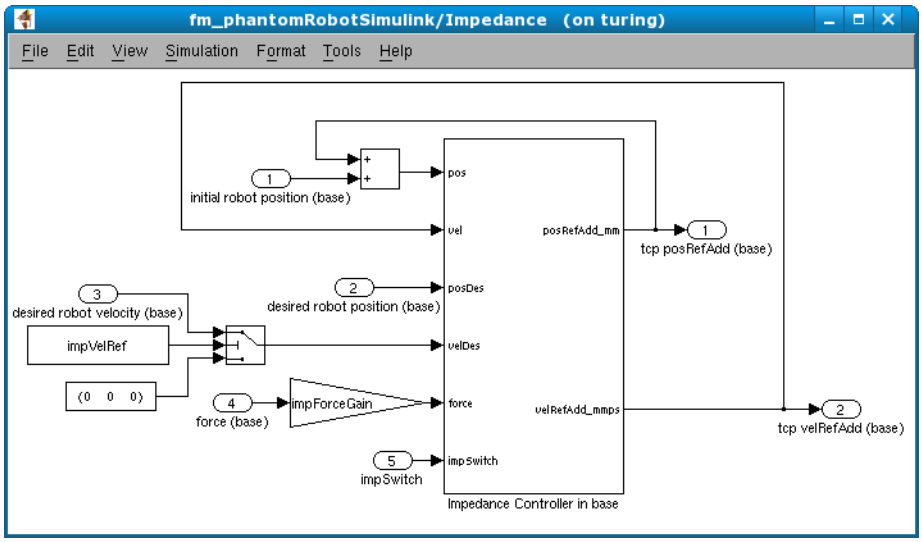

Figure 7.10 This subsystem holds both the impedance controller and some routing necessary to enable the implementation of the impedance control law.

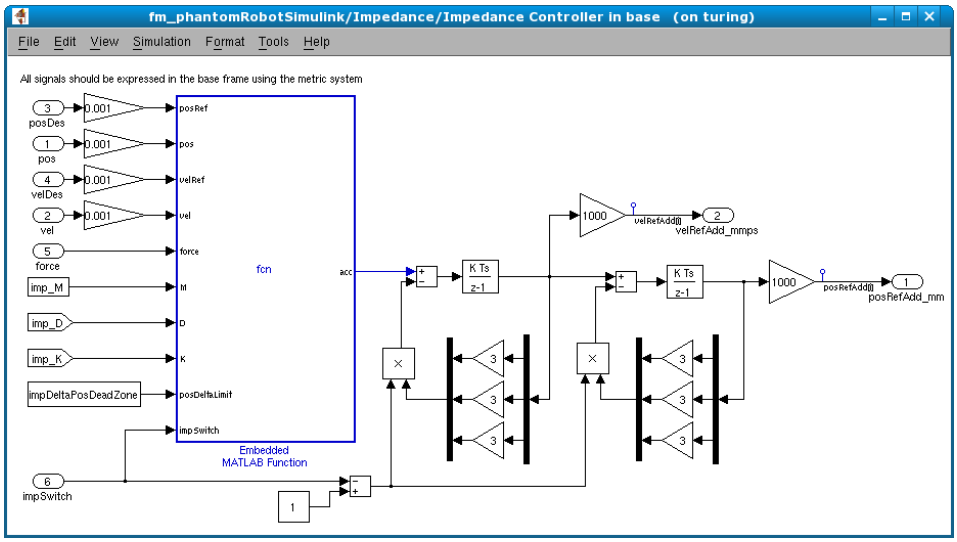

Figure 7.11 A Simulink implementation of the impedance controller described in Section 3.9.

were created. The impedance controller itself can be found in Figure 7.11. The latter subsystem implements the control law given by Equations 3.16 and 3.17. The embedded Matlab function in Figure 7.11 implements Equation 3.13 and the code can be found in Appendix C.

The parameter *impedanceControlSwitch* determines the in port *impSwitch* in both Figure 7.10 and Figure 7.11 and can be set from the GUI. A value of zero results in no impedance control being used whereas a value of one activates the impedance control. The default value is zero and the impedance controller should not be started until the *obtain* mode has been entered in the GUI. In order to prevent the integrals that result in additions in the position and velocity references in Figure 7.11 from building up when the impedance controller is turned off, these two integrals are drained whenever

impedanceControlSwitch equals to zero. That will cause the robot to return to the position it had before the impedance controller was activated.

From the GUI it is also possible to neglect the external force in the impedance controller and to turn off the velocity reference, see Figure 7.10, through the parameters *impForceGain* and *impVelRef*. For the latter, as before, it holds that a value of one will activate and a value of zero will deactivate. Deactivating the velocity reference is equal to setting it to zero which means the impedance controller will try to make the TCP position follow a change in position reference with a speed of zero. This will naturally utterly decrease the performance and there is, due to that fact, no other reason to turn off the velocity reference following than to investigate how great the performance loss will be.

The impedance controller uses the parameters M, D and K, see Section 3.9. It is not trivial to interpret how a system with three given values of these parameters will behave. Therefore, they are determined by Equations 3.18 and 3.19. *imp_Omega* and *imp_Zeta* represent the eigenfrequency and the damping and can be set from the GUI together with the mass, *imp_M*. They are then recalculated into *imp_D* and *imp_K* in the *Impedance Parameters* embedded Matlab function in Figure 7.5. The code for *Impedance Parameters* can be found in Appendix C. In the model the parameters are defined as constants, i.e., the system will have the same behavior in all directions.

7.6 The Position Reference Generation

The Simulink model has the responsibility to join the position references from the Teach Pendant, the *Phantom Robot Connector* and the impedance controller. Part of this is done in the subsystem called *Position Reference*, see Figure 7.12. The main idea is that the position given by the Teach Pendant in the *irb2ext.posRef* signal is sent to the *Phantom Robot Connector* as a static reference. In reality, the robots position is only needed when starting the *Phantom Robot Connector* so the fact that *irb2ext.posRef* does not change when the robot is moved by other means than those of the Teach Pendant, see Figure 7.3, does not concern the *Phantom Robot Connector*. The signal received by the Simulink model from the *Phantom Robot Connector* will then be used as a position reference in the impedance controller and its derivative used as a velocity reference. The derivation and also low pass filtering of the signal from the *Phantom Robot Connector* is done in the subsystem called *Discrete Filter* in Figure 7.12. The low pass filter is of the same type as the one in Section 7.4. The continuous filter constant is called *omega_PhFilt* and can be set from the GUI. The default value is $omega\_PhFilt = 30$. The derived joint angle velocities are then reinterpreted as a TCP frame velocity in the subsystem called *Calculate TCP velocity* in Figure 7.13.

The impedance controller will listen to appeals from both the position reference, velocity reference and any external forces when calculating its additions in position and velocity references. That position reference addition is then added to the TCP position given by the *irb2ext.posRef* signal and will be transformed into joint angles before sent to the robot as a new position reference through the *ext2irb.posRef* out port. A schematic view over the position reference loop is shown in Figure 7.14. In it the feedback loops in Figure 7.10 can be found as well.

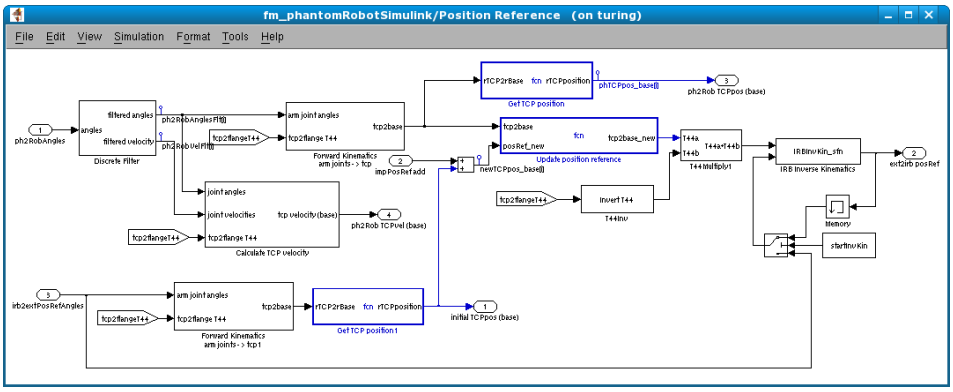

Figure 7.12 This subsystem will create the position reference that will be sent to the robot.

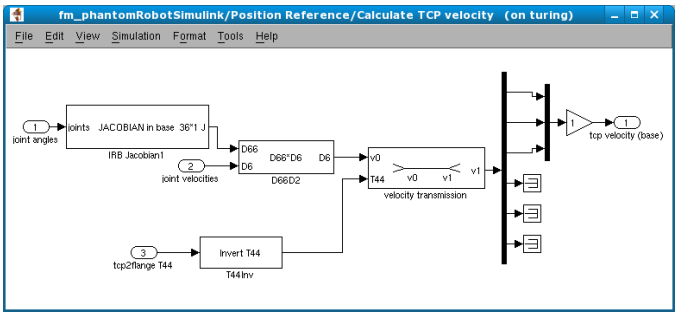

Figure 7.13 The subsystem *Calculate TCP velocity*. The inner subsystems come from the *ExtCtrl Library*. The angular velocities of the TCP frame are neglected.

In Figure 7.12 there is an inverse kinematics block from the *ExtCtrl Library*. The input parameters for that block are the desired homogeneous transformation matrix and the previous set of joint angles. The latter is the reason for the existence of the memory block.

It is important that the inverse kinematics block will select a solution near the one selected by the *Phantom Robot Connector* to obtain good performance. A problem may therefore occur when starting the Simulink controller. The output of the memory block during the first sample will be zeros which will lead to that the inverse kinematics block is, in a way, given free hands and may therefore select a quite random solution compared to the *Phantom Robot Connector*. This is prevented by sending the angles determined by the jogging of the robot to the inverse kinematics block until the user has decided it is safe to use the previous set of angles and has changed the parameter *startInvKin* from 0 to 1 in the GUI, [8].

7.7 The Velocity Reference Generation

The impedance controller seen in Figure 7.11 will determine an addition in velocity reference, see Equation 3.16. Figure 7.14 also shows the velocity reference generation loop schematically.

The addition in velocity reference is described as the cartesian flange velocity expressed in the base frame. The subsystem *Velocity Reference*, see Fig-

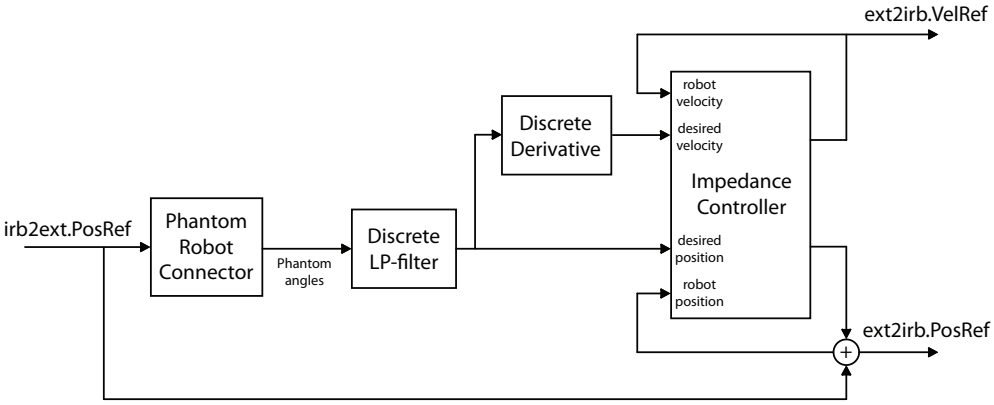

Figure 7.14 The loop determining the position and velocity references that will be sent to the robot. The jogging of the robot, the *Phantom Robot Connector* and the impedance controller will all have their say. It is assumed that the robot will follow the references sent to it very well, why the references can be used as feedback in order to prevent disturbances in the real measurements from entering. The *Phantom angles* is a set of robot joint angles based on the state of the Phantom.

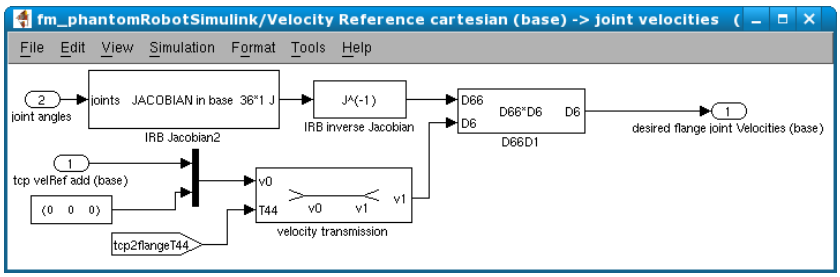

Figure 7.15 This subsystem will create the velocity reference that will be sent to the robot.

ure 7.15, will transform that velocity into joint velocities to prepare for sending the addition to the robot. Being an addition in reference, it is added to the signal *irb2ext.velRef* and the result is passed on to the outport *ext2irb.velRef*.

The addition is however not crucial as the *irb2ext.velRef* will be zero at almost all times as it is recommended not to jog the robot in *obtain* mode, see [8]. The *irb2ext.velRef* signal will not change if the robot changes its state, just as the with the *irb2ext.posRef* signal, see Section 7.1. It is therefore often unnecessary to add the *irb2ext.velRef* signal to the velocity reference sent to the robot through *ext2irb.velRef*. The only time there is a need to add the two reference signals is if an external program is run on the robot.

7.8 Solving the Bumpless Switching Problem

The previously mentioned fact that the signal *irb2ext.posRef* does not listen for changes in *ext2irb.posRef* will cause a problem when the user wants to stop controlling the robot through the Phantom.

The reason for the problem is that, as the Simulink model in Figure 7.5 is constructed, the signal *irb2ext.posRef* is used as a reference to the *Phantom Robot Connector*. In that program a new position reference will be calculated

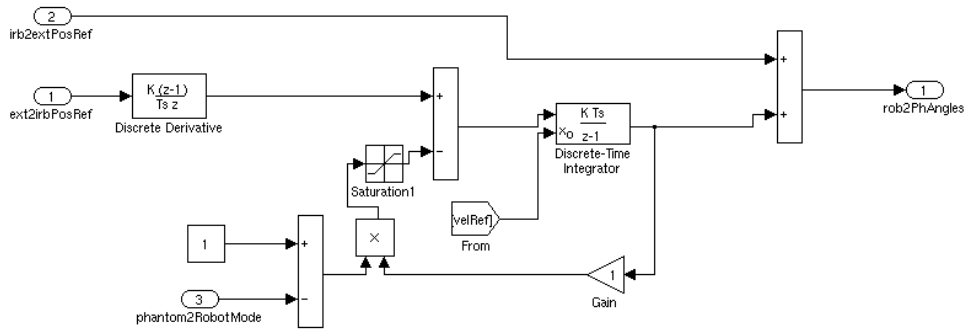

Figure 7.16 The subsystem *Mode Switch* will make the robot return to the its original position when control of the robot using the Phantom ceases. The derivative of the joint angle references will then go to zero. The from tag *velRef* contains the signal *irb2ext.velRef*.

depending on the state of the Phantom, see Chapter 9. This new position reference can be sent to the robot through the *ext2irb.posRef* port (here without any impedance control) and therefore not affect the *irb2ext.posRef* signal, see Section 7.1. The *Phantom Robot Connector* can deal with that but a problem will occur when the user decides to stop controlling the robot with the Phantom. As a result, the *ext2irb.posRef* signal will start to contain the values in *irb2ext.posRef*, a signal holding the position where the robot was when control using the Phantom was started. If the user has moved the robot away from that starting position, the robot will attempt a jump back to its starting position due to the fact that it will appear to the robot as if though a step in the position reference has just occurred. This will cause an error and the robot will hit the brakes.

To handle the above described situation, some kind of *Go Home* routine has to be implemented. One way of doing this is shown in Figure 7.16. The figure shows how a subsystem, *Mode Switch*, has been implemented to deal with the problem.

Outside the subsystem in Figure 7.16 there is some logics treating the value from the *ph2RobDelay* port. That makes the signal going in to the subsystem *Mode Switch* through the port *phantom2RobotMode* have the value one if the user is controlling the robot with the Phantom and zero if not.

The control structure in Figure 7.16 is presented as a block diagram in Figure 7.17. The value sent to the *rob2PhAngles* port will be passed on to the *ph2RobAngles* port by the *Phantom Robot Connector*, for more details see Chapter 9. It is therefore assumed that there will be a delay of one sample in the outer loop in Figure 7.17.

To analyze the inner loop in Figure 7.17 the transfer function from v to δ is needed. Equation 7.2 gives the desired transfer function

$$\delta = \frac{K_i T_s}{z - 1 + K_i T_s} \cdot v \tag{7.2}$$

From Equation 7.2 the pole of the closed loop system is determined as

$$p_i = 1 - K_i T_s \tag{7.3}$$

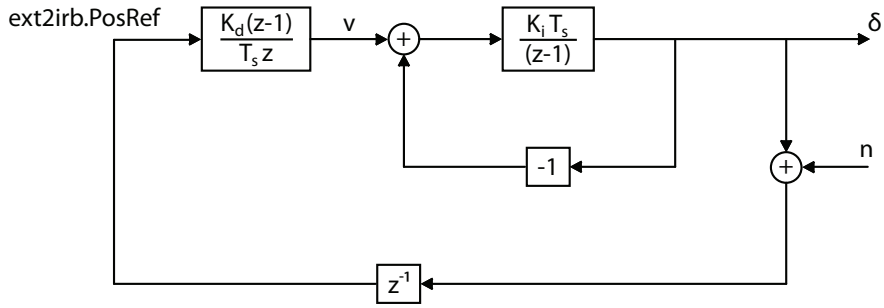

Figure 7.17 A block diagram over the control structure implemented in Figure 7.16 when the draining of the integral is not saturated.

The sampling time is set to $T_s = 0.004$ s which will lead to a stable pole for $K_i \in]0, 500[$.

An important question regarding the outer loop in Figure 7.17 is how a disturbance entering through n will affect δ. The transfer function from n to δ was therefore derived, see Equation 7.4

$$\delta = \frac{K_d K_i (z-1)}{z^3 + (K_i T_s - 1)z^2 - K_d K_i z + K_d K_i} \cdot n \tag{7.4}$$

Through clever choices of K_d and K_i, all poles can be placed within the unit circle which leads to stability. However, tests showed that doing so, the *Go Home* routine though completing its task, did so very slowly. Another issue is that the *Go Home* routine affected the position reference to the robot even when it should not. The problem with bumpless switching was therefore solved in another way, see Section 9.7.

8. Phantom Robot Connector

A program named *Phantom Robot Connector* written in C++ was developed. Its main purpose is to enable communication between the Phantom and the robot to provide a way of controlling the robot's motions based on these of the Phantom and to give force feedback to the operator holding the Phantom.

In order to do this, the *Phantom Robot Connector* collects information from both the robot and the Phantom, performs calculations and then sends new information back to them. The communication between Simulink and the *Phantom Robot Connector* on a Windows platform, was enabled by another program that was written in C, see Section 9.5. The C program was written for Linux, and was a convenient solution to the problem with the *Labcomm* and Windows compatibility, see Section 5.2.

The data received from the robot by the *Phantom Robot Connector* through the TCP/IP connection is the force signal and the robot angles. The force signal is sent to the Phantom as force feedback and the robot angles are used to determine the starting configuration of the robot. From the Phantom, data concerning the position and orientation of the Phantom tip is gathered. This data is used to provide the robot with a new set of joint angle references sent through the TCP/IP connection.

The *Phantom Robot Connector* will also draw a representation of the robot and the orientation of the robot's TCP frame in a graphical window on the computer screen. This virtual robot can be used to test the performance when controlling a robot with the Phantom. Virtual objects have been created to simulate contact forces affecting the virtual robot.

8.1 The Program Modes

There are four different modes in the program, see Table 8.1. They are *phantom2graphic*, *robot2graphic*, *goHome* and *phantom2robot*. Depending on which mode is active the program will act differently.

The default mode is the *robot2graphic* mode, where the state of the Phantom is ignored and the graphical robot is a correct representation of the real robot. From the *robot2graphic* mode, the user can switch to either the *phantom2graphic* mode or the *phantom2robot* mode. In the *phantom2graphic* mode the user can control the graphical robot through the Phantom. The r0TCP frame for the graphical robot will be the same as the TCP frame for the real robot, but the graphical robot's TCP frame is based on the state of the Phantom. In this mode it is also possible for the user to simulate interactions between the virtual robot and a virtual wall or a virtual sphere, i.e., the user can "touch" the objects by controlling the virtual robot with the Phantom.

In the *phantom2robot* mode on the other hand, the Phantom controls both the virtual and the real robot. Due to the force control entering in the Simulink model there will be a small difference between the real robot and the graphical representation of it.

Lastly, the mode *goHome* can not be selected by the user. It is merely a transition mode when switching from the *phantom2robot* mode to the *robot2graphic* mode. This is the solution to a problem occurring in Simulink, see section 7.8. Briefly its purpose is to prevent the robot to attempt a jump in

Mode	*Program Functionality*
phantom2graphic	The Phantom is controlling the virtual robot. The real robot is not affected by the Phantom.
robot2graphic	This mode is active when the *Phantom Robot Connector* starts. The virtual robot is a representation of the real robot, i.e., any changes in the Phantom position or orientation are not displayed.
goHome	This mode can not be chosen by the user and is only active in the transition from the *phantom2robot* mode to the *robot2graphic* mode in order to make the real robot return to the position it had when the human operator switched to the *phantom2robot* mode.
phantom2robot	The mode where the real robot will follow the Phantom's movements.

Table 8.1 The program modes in the *Phantom Robot Connector*.

position and/or orientation when switching back to the *robot2graphic* mode from the *phantom2robot* mode.

Besides the different mode settings the user can also decide whether the mapping between the Phantom and the robot should just depend on the position or on both the position and orientation of the Phantom tip. This must be chosen when the program is in the *robot2graphic* mode. If the mapping is done using both position and orientation everything works as in Chapter 4. If it is done using the position alone instead, the part of the homogeneous transformation matrix that describes the Phantom's orientation is set to the unit matrix.

8.2 Digital Force Filter

The force signal received by the *Phantom Robot Connector* from the Simulink controller is sampled at the frequency 250 Hz in contrast to the haptic rendering which is run at approximately 1000 Hz. To let the Phantom render the force in 250 Hz is not good for the performance of the haptic rendering, which will behave more stable with a higher updating frequency, see [21]. In order to take advantage of the fast haptic rendering a discrete filter was developed. The purpose of the filter is to smooth the force signal and by doing this attain smaller force steps to be rendered by the Phantom. When sending this force through the proposed filter it will appear as though the force signal is being rendered in 1000 Hz.

Equation 8.1 describes the filter used, where $u[k]$ and $y[k]$ is the force signal received and rendered by the Phantom respectively, at the time step k.

$$y[k] = b_1 u[k] + a_1 y[k-1] + a_2 y[k-2] \tag{8.1}$$

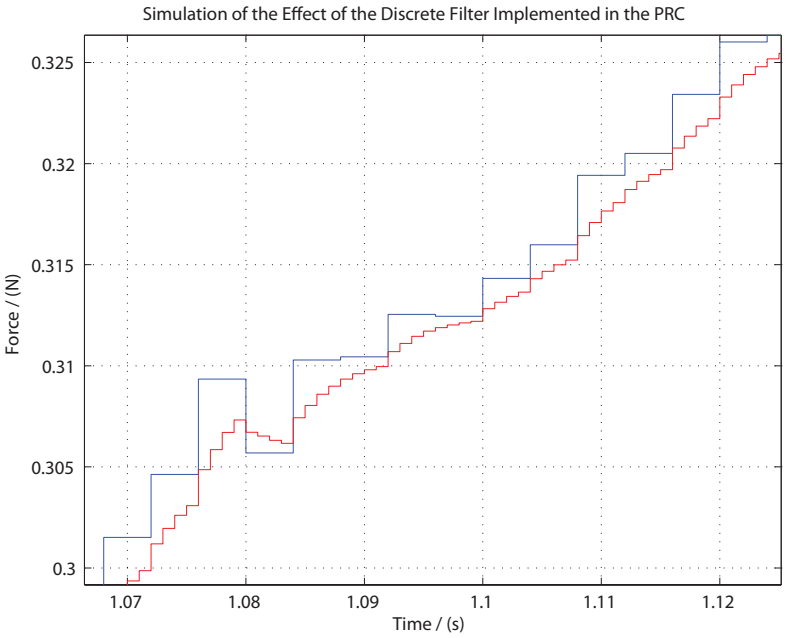

Figure 8.1 Shows a force signal and its filtered version. The data used in this figure has been gathered from the actual force sensor in Section 3.3. However, the filter is simulated in Matlab under the assumptions that the force signal has a frequency of 250 Hz and that the filtered signal has a frequency of 1000 Hz.

By choosing the parameters b_1, a_1 and a_2 such that equation $1 = \frac{b_1}{1-a_1-a_2}$ holds and that the poles of the filter is inside the unit circle, $y[k]$ will go towards $u[k] = u$ if u is a constant. The discrete filter equation and the corresponding poles can be seen in Equation 8.2 and 8.3 respectively.

$$y[k] = b_1 u[k] + a_1 y[k-1] + a_2 y[k-2] \tag{8.2}$$

$$z = \frac{a_1}{2} \pm \sqrt{\frac{a_1^2}{4} + a_2} \tag{8.3}$$

The filter used in *Phantom Robot Connector* has the parameters $a_1 = 0.5$, $a_2 = 0.3$ and $b_1 = 0.2$. The poles are $z = \{0.8521, -0.3521\}$ and $\frac{b_1}{1-a_1-a_2} = 1$ which proves that the filter is stable and that $y[k] \to u[k], k \to \infty$. A force signal filtered by the proposed filter can be seen in Figure 8.1.

8.3 Using the Phantom Robot Connector

When the *Phantom Robot Connector* starts, a console will appear and the user will be given three options to choose between, see Figure 8.2. The first option is to start running a virtual robot equipped with a tool that is a virtual replica of the force sensor construction in Figure 3.5. The second option is to run the program with the same virtual robot but without the tool. Both these options make the program run without any communication with the real robot. In these cases, the *Phantom Robot Connector* can therefore be seen as a pure virtual robot that will demonstrate how the real robot would have

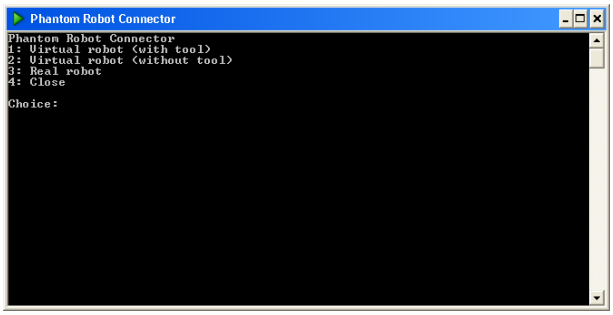

Figure 8.2 The *Phantom Robot Connector* awaits the user's choice at startup.

behaved, had it been connected. The third option in the console is to connect to the IRB140B over a network. As soon as a connection is established the main program will start. The force sensor described earlier is assumed to be attached to the IRB140B in option number three.

There are a number of different keys the user can press to control the run-time behavior of the *Phantom Robot Connector*. A full list is found in Appendix B. One of the things the user can activate is the painting mode, by pressing [D]. If this is activated the program will draw small blue surfaces perpendicular to the sensed force at the correct location in the 3D-space. This will work for both virtual objects in the *phantom2graphic* mode and for real world objects in the *phantom2robot* mode. If the user presses [C] all the paint is erased. There are also a lot of small visual settings that the user can activate or deactivate.

The view perspective of the virtual robot can be changed by moving around on a sphere surrounding the robot. This is done by pressing the left and right arrow keys to move around a horizontal circle and by pressing the up and down arrow keys to move around a vertical circle. Pressing the page up and page down keys will change the radius of the sphere causing the view perspective to zoom in or out. By pushing key [P], [R], [H] the program will change mode to *phantom2graphic*, *robot2graphic* or *phantom2robot* respectively. To change the mapping setting between free orientation and fix orientation of the TCP frame the [O] key is pushed. All settings that are activated will be highlighted in the color green.

In the *phantom2graphic* mode some other settings will be revealed on the screen. These are the *No Obstacle*, *Wall* and the *Sphere* and they are activated by pressing the [N], [W] or [S] keys respectively. As the names imply they activate the virtual wall, the virtual sphere or in the first case neither of them. The setting that is activated will as before be highlighted in green.

Some settings can only be made in a predetermined order. This is in order to avoid a strange behavior from the program. For example, the virtual obstacles are only available in the *phantom2graphic* mode. They will disappear when changing mode. The allowed combinations can be seen in figure 8.3.

On the right side of the screen the real robot angles are printed. But if the mode is set to *phantom2graphic* it will only show the simulated robot angles that the graphical robot are using. Below the Robot angles the force signal is expressed. Also for the force it holds that it is not the real force signal that is represented in the *phantom2graphic* mode. In the other modes it is the received force signal that is printed on the screen.

To toggle full screen mode the [F] button is used, whereas pressing [Q] or

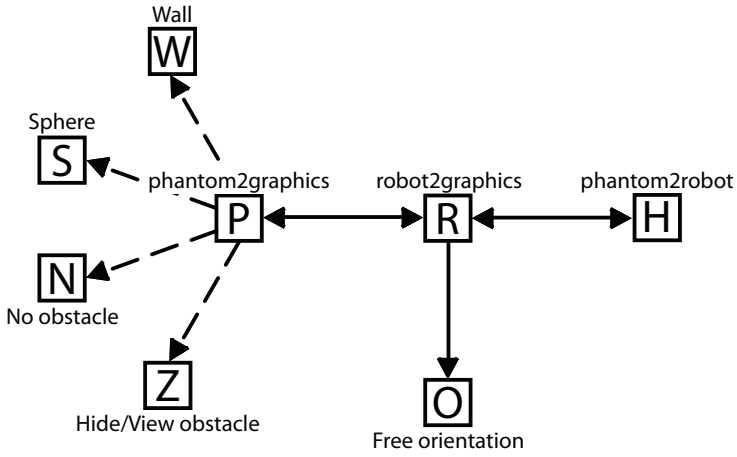

Figure 8.3 Depending on which mode is active, different actions can be taken. For the full list, see Appendix B.

[ESC] will exit the program. The force sensor attached to the robot can be reset by pressing [0].

9. Code Implementation

In this chapter follows a more detailed description about how the *Phantom Robot Connector* is coded. Some C++ functions will be mentioned, however without their parameters or return values. To make it clearer, all function names will be written in a `fixed width font`. For more details it is recommended to have a look directly at the C++ or C code. It can be accessed by visiting the publications page of the Automatic Control website (at present http://www.control.lth.se), and searching for the thesis title. There will be links to a zip-archive containing the code.

The program uses some different code packages. These are *GLUT* (OpenGL Utility Toolkit), *OpenHaptics* and *ExtCtrl Library*. They each represent one of three corner stones in the *Phantom Robot Connector*. To create the graphics, *GLUT* is used. It is a graphics API in C that uses *OpenGL* (*Open Graphics Library*), [19]. Besides providing an easy way of creating graphical applications it is also supported by many different platforms which makes it very useful. To render a haptic simulation and to communicate with the Phantom, the *OpenHaptics* API is used, or more specifically the *HDAPI*, see Section 2.2. *OpenHaptics* is also available on more than one platform. Finally, the *ExtCtrl Library* was used to make a proper mapping between the Phantom and the robot. It contains functions written in C that handles the forward and inverse kinematics for, amongst other robots, the IRB140B. There is also a fourth corner stone, the communication with the real robot that will be presented more in Section 9.4.

In the following, the term *Phantom angles* will refer to a set of robot joint angles calculated based on the state of the Phantom. *Robot angles* on the other hand will refer to joint angles received from the robot.

9.1 An Overview of the Code Structure

The program is based upon three threads: The haptic, the graphic and the underlying thread concerning the TCP/IP connection. An overview of the structure is seen in Figure 9.1. The program starts by setting the initial conditions and global variables and starting the threads.

There are some different classes used. The RobotHaptics class, see Section 9.2, is where all the calculations refereing to the mapping between the robot and the Phantom configuration are made. The RobotGraphics class, see Section 9.3, have a set of functions concerning the drawing of the virtual robot. The two classes are totally independent of each other. In order to take advantage of them both a new class named Robot extends them, see Figure 9.2. No further functions are added to the Robot class.

The connection to the robot is made through a TCP/IP connection an therefore a Communicator class was written. More correctly, three different classes were implemented where one of them is an abstract base class. For further details see Section 9.4. The idea is that the Communicator classes handle all code concerning the TCP/IP connection by themselves. When creating an instance of one of these classes, an underlying thread will take care of all receiving and sending of data over the network. This approach makes using them easy, because all synchronization is made within the classes which will avoid

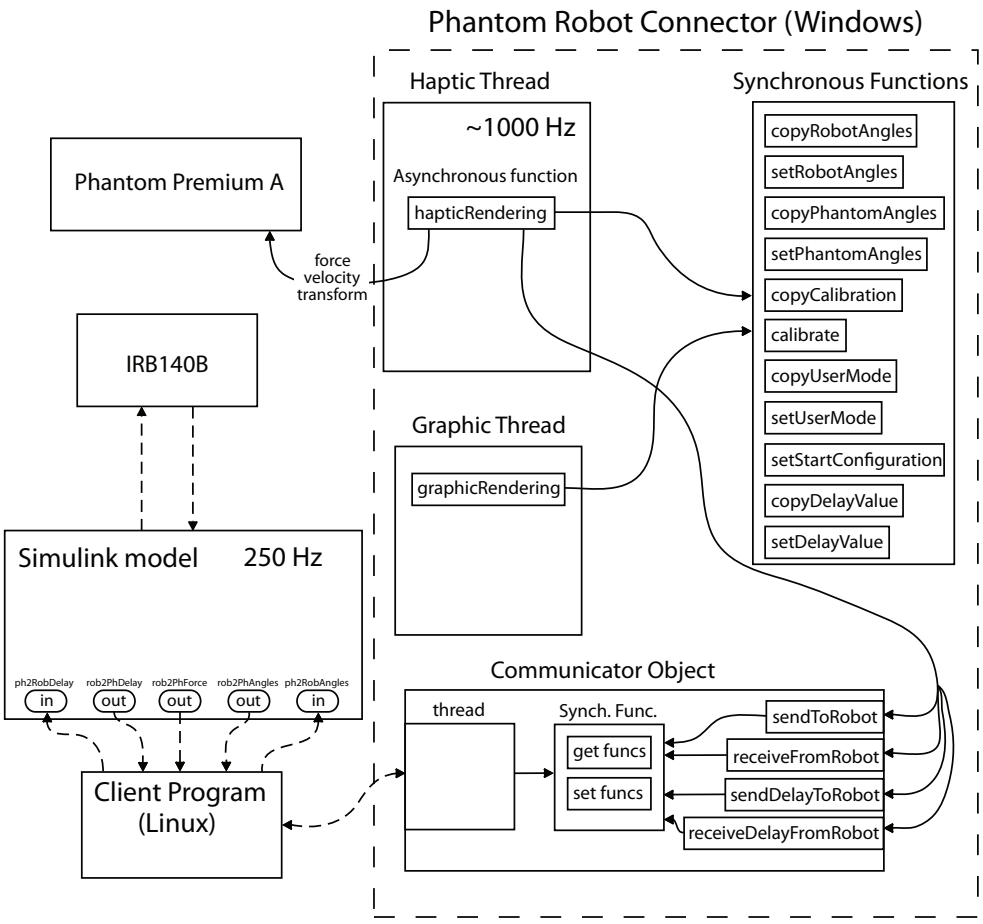

Figure 9.1 An overview of the program structure and communication when the *Phantom Robot Connector* is run on a Windows system. Far more functions and variables exist than what is shown here. Pay attention to the synchronization of the shared data.

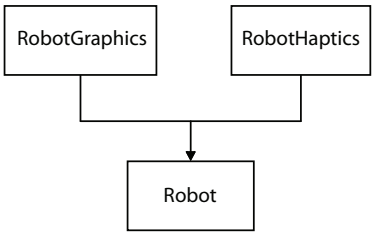

Figure 9.2 The inheritance tree for the Robot class.

halting the program.

The haptic thread uses an instance of the Communicator and can therefore send data to the Simulink model whenever it has finished its calculations. The lower paced Simulink controller then uses the latest data it has received when performing its own calculations.

The classes Obstacle, Wall and Sphere are used to simulate objects in the virtual environment, see Section 9.6. The Obstacle class is an abstract base class, or an interface, for the Wall and Sphere class and can be used to define the haptic and graphic behavior of a given object.

9.2 The RobotHaptics Class

The RobotHaptics class is responsible for all calculations concerning the mapping between the real robot and the Phantom. When initializing an instance of the RobotHaptics class the current angles of the real robot are used to calculate all the static transformations H_{hBase}^{rBase}, H_{rBase}^{hBase}, H_{r0TCP}^{rBase} and H_{rTCP}^{hTCP}, see Figures 4.1 and 4.3. The user must also define the transformation $H_{rTCP}^{rFlange}$. If no tool should be used, the matrix is simply set to an identity matrix. All calculations are assumed to be in millimeters.

The explicit mapping is made in the function `updateJointAngles` which has the matrix H_{hTCP}^{hBase} and the latest joint angles for the robot as arguments. H_{hTCP}^{hBase} is representing the position and orientation of the Phantom tip. By using the method of mapping described in Section 4.1, H_{rBase}^{rTCP} can be calculated. The inverse kinematics needed in that method are determined by the function `optInverseFlange` from the *ExtCtrl Library*. It determines all solutions to the inverse kinematics problem and returns the closest solution in means of the difference between the possible solution and the previous set of joint angles for the robot by using the 1-norm. The joint angles returned are called *Phantom Angles* as previously defined.

9.3 The RobotGraphics Class

All the code concerning the drawing of the robot can be found in the RobotGraphics class. It uses *OpenGL* for the graphic rendering. When initializing an instance of the RobotGraphics class all values representing the standard Denavit-Hartenberg representation, see Table 3.1, are set. The function `draw` will draw the robot and uses the joint angles as argument. Using Equation 3.4 and the values in Table 3.1 all joint positions can be calculated and thereby made drawable. All of the robot's links are visualized as cones and all joints as spheres in different sizes. Therefore it is not an entirely correct visualization of the real IRB140B.

The function `draw` will also draw the JR3 force sensor if it is mounted on the virtual robot along with the robot base frame and current TCP frame, see for example Figure 9.8. In all calculations in the RobotGraphics class, the unit is assumed to be meters.

9.4 The Communicator Classes

Depending on which operating system is used the communication with the robot has to be done differently. If the *Phantom Robot Connector* is run on a Linux system, the *Phantom Robot Connector* can communicate with the robot directly. In that case it can use the functions generated in Section 5.2 to send and receive data to and from the robot.

If however the Phantom is connected to a computer running Windows, as was the case during this thesis, complications arise. The main problem is that the labcomm functions, see Section 5.2, are written for Linux only. No Windows counterpart exists. Therefore, in order to be able to run the *Phantom Robot Connector* on a Windows platform, a new program has to be implemented on another computer that is running Linux, see Figure 9.1.

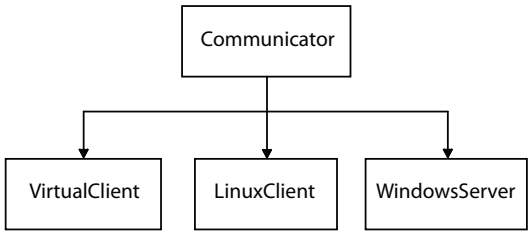

Figure 9.3 The inheritance tree for the communication classes.

The program is called *winComm* and was written in C, see Section 9.5. Its purpose is to establish communication over TCP/IP with both the *Phantom Robot Connector* on the Windows computer and the built Simulink model on the robot computer and to pass on the values they need to send to each other.

By introducing the class structure seen in Figure 9.3 the shifting between different platforms will be made easier. The *Phantom Robot Connector* has a pointer to a Communicator as a global variable. Then it can create an instance of the appropriate subclass and use the pointer for access.

The Communicator class is an abstract class with four public abstract functions: `sendToRobot`, `receiveFromRobot`, `sendDelayToRobot` and `receiveDelayFromRobot`. As a result, each of the subclasses to the Communicator class has to implement their own version of these four functions. For a Linux system, the *Phantom Robot Connector* acts as a client that connects to the robot. Therefore the subclass LinuxClient was written. It was however never implemented since the *Phantom Robot Connector* was running on Windows throughout the whole thesis. In case of Windows, the *Phantom Robot Connector* instead acts as a server to which the additional Linux run C-program, *winComm*, is connected. Hence the subclass to be used is given the name WindowsServer. Both subclasses are however dealing with the communication part of the *Phantom Robot Connector* why their common base class is called Communicator.

The WindowsServer has a thread of its own, as should the LinuxClient have if implemented. That thread copies the values stored when the *Phantom Robot Connector* calls `sendToRobot` or `sendDelayToRobot` and sends these values to the robot accordingly. Likewise, the functions `receiveFromRobot` and `receiveDelayFromRobot` copies the latest values received from the robot by the thread when they are called by the *Phantom Robot Connector*. All copying and setting of values in these classes is done in a thread-safe manner. Mutual exclusion is used in a way suitable for the current platform. The internal thread is run as fast as possible. However, fresh data is available from Simulink only at a rate of 250 Hz, see Chapter 7.

If the LinuxClient should be implemented, it should be functioning in the same way as WindowsServer to prevent any misbehavior.

In Figure 9.3 a third sub class can be found, the VirtualClient. As the name suggests, this class has no TCP/IP connection. Its only responsibility is to appear to the *Phantom Robot Connector* as if though it communicates with an ideally working robot, i.e., the angles sent to the robot will also be the angles received from the robot. This sub class can hence be used both on Linux and Windows platforms and also when no robot is connected, in order to test the graphics etc.

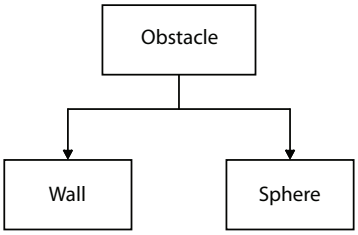

Figure 9.4 The inheritance tree for the Obstacle classes.

9.5 Client Program for Linux

When the Phantom is connected to a computer running Windows the need for a C program developed for Linux emerges. This program has been named *winComm* due to its sole purpose: to enable the Windows computer to communicate with the robot, see Figure 9.1.

The program starts by setting up the TCP socket connection to the server initialized in the *Phantom Robot Connector* by the WindowsServer class. When this is done *winComm* advances and starts to set up the communication with the robot. After all necessary steps for successful communication with the built Simulink model have been taken, *winComm* starts a thread. Its duty is to listen for information sent out by the Simulink model. Helping the thread with this is a few handler functions, one for each out port in the Simulink model that has been defined in the `*.lc` file in Section 5.2.

After this, *winComm* heads into an infinity loop where it conducts the following actions repeatedly: Read from Windows computer, send to robot, receive from robot and finally send to Windows computer. The infinity loop does not halt to wait for fresh data from Simulink. It just sends to Windows the latest data received by the handler functions no matter how old it is. It can seem unnecessary that all this old information is sent, but it makes *winComm* totally independent of the sampling time of the Simulink model. That independency is deemed worth striving for in case the sampling time in the Simulink model should be varied.

9.6 Obstacle Classes

The Obstacle class is the base class to the Wall class and the Sphere class, see Figure 9.4. It exists of only three abstract functions. They are `getForce`, `isInside` and `draw`.

The purpose of the function `isInside` is to decide whether a position in 3D-space should be considered to be inside the virtual object using the Obstacle class. The function `getForce` calculates the force a virtual object will exert if the function `isInside` returns true. The combination of these two functions gives all the information needed to simulate a haptic environment. The final function `draw` is responsible of drawing the object in 3D-space.

The Wall class simulates a virtual wall that lies in the xy-plane, see Figure 9.6. The virtual force experienced by the surroundings is simulated as a spring force, see Figure 9.5. The Sphere class simulates a sphere with a specified radius and center point, see Figure 9.7. For the Sphere class the force is

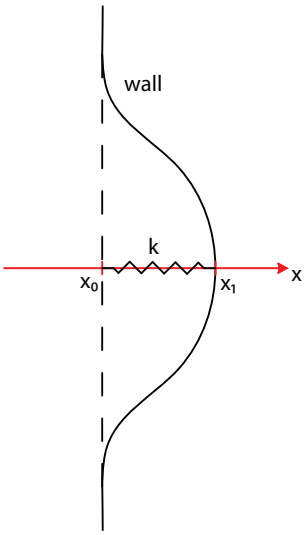

Figure 9.5 The virtual wall will return a force $F = -k(x_1 - x_0)$.

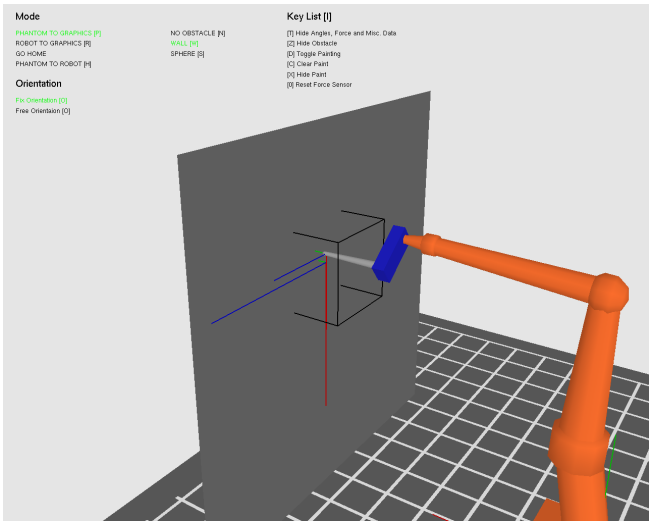

Figure 9.6 The virtual wall. A simple graphical representation of the JR3 force sensor is mounted on the robot.

defined as a spring force normal to the tangent of the surface in the contact point.

9.7 The Haptic Thread

The haptic thread runs at a frequency of approximately 1000 Hz. The choice of a high frequency ensures a stable haptic rendering that is perceived as a natural sense of touch by a human operator, [21]. The haptic rendering takes place in the function `hapticRendering` which is called by the haptic thread in an asynchronous way, see Section 9.9. However the force feedback and the receiving of the state of the Phantom takes place in a special high priority section in `hapticRendering`. In that section it is guaranteed that the state

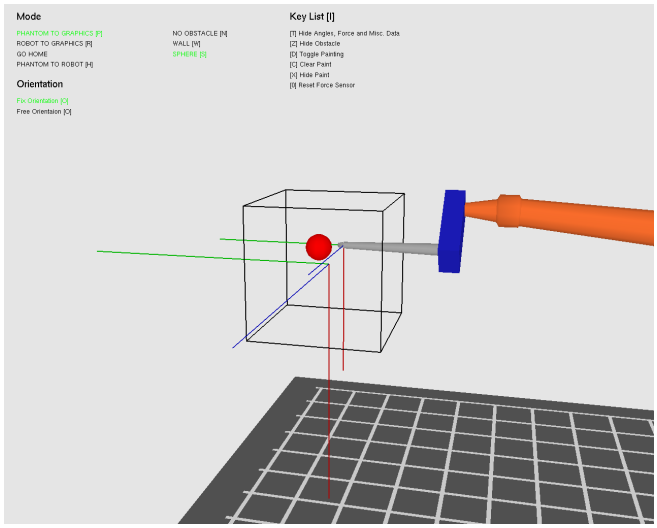

Figure 9.7 The virtual sphere.

of the Phantom remains unchanged, [21]. Here, the current position, velocity and orientation of the Phantom tip is read and the latest force signal received from the robot is sent to the Phantom as force feedback.

Outside the high priority section calls are made through a pointer to a Communicator object to get the latest robot angles and force signals. These values are then saved through synchronous calls to different functions, see Section 9.9.

Thereafter, depending on which mode is active, different pieces of code are executed. In the *robot2graphic* mode calls are made to the synchronous functions `calibrate` and `setStartConfiguration`. The function `calibrate` saves the inverse of the transform collected from the Phantom. This is later on used to solve the jump problem, see Section 4.2. The function `setStartConfiguration` sets the r0TCP frame to be the same as the current TCP frame for the robot.

If the *phantom2robot* mode is active the calculated Phantom angles will be sent to the robot. When the mode *goHome* is active the calculated Phantom angles are changing with constant velocity towards the received robot angles. This is a solution to the problem discussed in Section 7.8. Thus, the Phantom angles are totally independent of the state of the Phantom in the *goHome* mode.

The piece of code executed in the *phantom2graphic* mode is similar as the *phantom2robot* but with the difference that no data communication with the robot occurs. All forces felt by the Phantom are from the virtual environment, e.g., the wall and sphere obstacles.

The jump problem described in Section 4.2 is triggered when switching modes from *robot2graphic* to *phantom2robot*. To solve this the same solution as suggested in Section 4.2 is used.

Finally calls are made to the functions `updateHapticFrequency` and `updateDelay`. The function mentioned first calculates an approximation of the haptic frequency. This is done by measuring the time it takes for the haptic thread to run for a predetermined number of cycles. The latter function does a similar job. It calculates an approximation of the time elapsed during a round

trip back and forth to the robot. The approximation is achieved by measuring the average time for a significate number of round trips. Each round trip is marked by the return of a value sent to the Simulink controller on the robot's main computer. When a value returns another value is sent out. The time it takes until that specific value is received is defined as the time for one round trip. By default the values sent out are shifting between zero and one but the user can change that into a shift between two and three instead by pressing the [0] button, see Appendix B. This is used to reset the force sensor, see Section 7.3.

9.8 The Graphics Thread

In the main function all the initialization concerning the graphics is done, e.g., the window size, the depth settings, the light settings etc. The exact thread frequency for the graphics loop is not critical as in the case of the haptic thread. However it is recommended that the CPU is fast enough for the animation to appear smooth to a human eye.

The graphics thread never communicates with the Communicator object directly. If any values shared with the haptic thread are needed they are collected through calls to synchronous functions, see Section 9.9 and Figure 9.1. All the global variables that are only used in the graphic thread are completely thread safe and can be used without fear of any conflict.

In the graphics loop a call to the function `graphicRendering` is made repeatedly. In `graphicRendering` all things that concern the actual drawing on the screen is done through calls to several functions. Their purpose is to draw everything that should be visible in the graphical window, e.g., the robot, the different coordinate frames, the floor, an obstacle (if any) and the robot's workspace, see Figures 9.6, 9.7, 9.8 and 9.9.

To draw the robot a call is made through the instance of the robot to the function `draw` which draws the robot based on the joint angles. The joint angles used correspond to the Phantom angles if the *phantom2graphic* mode is active but otherwise they correspond to the joint angles for the real robot. There are also calls made to draw some of the different coordinate frames. To draw the Phantom's workspace the function `drawWorkspace` is called. It draws a rough approximation of the space where the Phantom will render a more or less correct force feedback, see Figure 9.8. There is also a lot of information printed as text on the screen. An example of that is the approximation of the the graphic frequency. The graphic frequency is calculated approximately in the function `updateGraphicFrequency` which works in the same way as the function `updateHapticFrequency`, see Section 9.7.

An instance of the Sphere class or the Wall class is used to draw a virtual wall or a virtual sphere depending on which setting is activated by the user.

There is a class called Painter. It can be used to make invisible obstacles, virtual or real, visible. Using synchronous functions, information about the current force and the robot's TCP position is gathered. This information can be used by the Painter class. If the force is higher than a threshold value the instance of the Painter class will use the position and the force to derive a matrix describing a frame where the z-axis is aligned with the force direction. This is used to draw a small blue square perpendicular to the force. All these frames are stored in a vector so that they can be drawn at each graphics

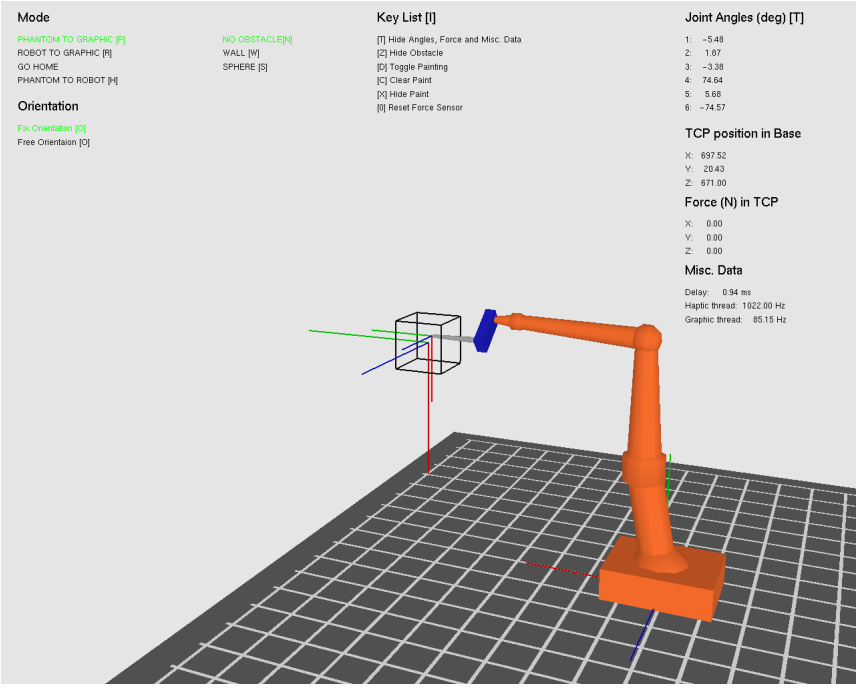

Figure 9.8 The virtual IRB140B near the starting configuration for which the rTCP frame has been drawn. As throughout this report the x, y and z axes correspond to red, blue and green respectively. The black-edged box indicates an approximation of the current workspace.

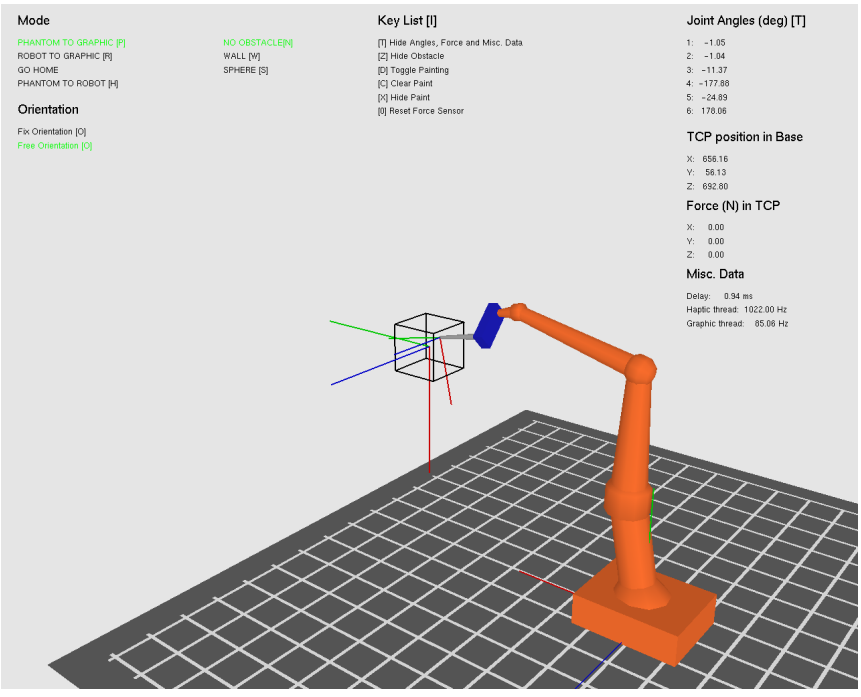

Figure 9.9 Another configuration of the virtual IRB140B. The Phantom has been moved farther away from the hBase frame causing the IRB140 to move farther away from its starting rTCP frame.

update.

The graphics loop also checks if any keyboard key or mouse button has been pressed. One of the functions `mouseFunc`, `inputKey` or `inputKeyboard` is then called depending on which action that took place. This allows the user to modify the program in a predetermined way while running it. For example, the function `lookAt` is controlling the perspective of the graphic scene and is called by the function `inputKey`. Hence it is possible to change the perspective of the virtual robot while running the program, see Section 8.3.

9.9 Synchronous Functions

In *OpenHaptics* the *HDAPI* controls all communication with the Phantom. It provides a scheduler that is responsible of deciding which code that should be run at a specific time. To provide the scheduler with code either one of the functions `hdScheduleSynchronous` or `hdScheduleAsynchronous` can be used. These functions take three arguments: function name, the data sent to the function specified, and finally which priority this call has compared to other scheduled functions. The function `hdScheduleSynchronous` means that the scheduler will try to execute the function provided immediately and without any interruptions. For deeper an more detailed information on how the *OpenHaptics* scheduler works see [21] and [20].

The most important function is the `hapticRendering` which is scheduled as asynchronous and with the highest possible priority. Therefor the `haptic-Rendering` will be called every time in the haptic thread with a frequency of approximately 1000 Hz. The scheduler is also responsible for the functions described in Table 9.1. They are scheduled by `hdScheduleSynchronous` to achieve a thread safe behavior.

Function name	*Description*
`copyRobotAngles`	Copies the real robot angles.
`setRobotAngles`	Saves the real robot angles.
`copyPhantomForce`	Copies the force that is exerted on the Phantom.
`setPhantomForce`	Sets the force that should be exerted on the Phantom. The force will not be exerted during the *robot2graphic* mode.
`copyPhantomAngles`	Copies the robot angles calculated from the Phantom.
`setPhantomAngles`	Saves the robot angles calculated from the Phantom.
`copyhTCP2hBase`	Copies the transformation saved from the Phantom.
`sethTCP2hBase`	Saves the transformation from the Phantom.
`copyCalibration`	Copies the inverse transformation from the Phantom which was saved by calibrate.
`calibrate`	Calibrates, which means to save the inverse of the Phantom transformation during *robot2graphic* mode.
`setStartConfiguration`	Sets the start configuration for the robot which means that hTCP frame is defined.
`copyUserMode`	Copies the currently active mode.
`setUserMode`	Sets the current mode.
`copyPh2GrMode`	Copies the currently active Ph2Gr mode. This is used to decide which, if any, obstacle is active.
`setPh2GrMode`	Defines what obstacle is active.
`copyFixOrientation`	Copies the fix orientation setting.
`setFixOrientation`	Sets the current orientation setting.
`copyHapticFrequency`	Copies the saved estimation of the haptic thread frequency.
`setHapticFrequency`	Saves the estimation of the haptic thread frequency.
`copyDelayValue`	Copies the measured delay for a signal round trip to the robot.
`setDelayValue`	Saves the measured delay for a signal round trip to the robot.
`resetForceSensor`	Resets the force sensor attached to the robot.
`copyResetForceSensor`	Copies the current reset force sensor setting.

Table 9.1 A list of all synchronous functions called by both the graphic and haptic thread.

10. Results

In previous chapters, theory and the control of the robot with the help of a haptic device were described. In this chapter the results of the thesis are presented, i.e., how everything previously described in this report worked when tested. A lot of tests concerning the impedance parameters were done in order to decide the most suitable parameter combination for the impedance control. Something that is also shown is how different parameter settings can lead to other useful implementations, e.g., lead-through or simulating the phenomenon of inertia.

The most interesting result is perhaps how a human operator experiences the environment of the robot, i.e., the haptic behavior for the Phantom-Robot relation. Several tests were conducted to investigate how different obstacles were experienced haptically when touched by the robot. The stiffness of the different materials was also derived in order to see how the stiffness is influencing the haptic behavior.

There was a painting mode implemented in the *Phantom Robot Connector* which can be used both in the virtual and the real world. The painting was investigated in Section 10.5.

10.1 Stiffness of the Materials Used

Haptic tests were performed by touching three different materials with the IRB140B. The objects were the cardboard box seen in Figure 10.15, a small piece of rubber foam attached to the mentioned cardboard box and a piece of cellular plastic also attached to the mentioned cardboard box. It is of interest to have a rough estimation of the stiffness for the three materials. These estimations could help giving an understanding of which impedance control parameters that should be used and how the material stiffness influences the haptic behavior.

By setting the desired position for the robot to be inside an object through inducing a step, the impedance controller will make the robot push on the object in order to reach the set point. To estimate the stiffness parameter of that object, the force can be plotted in relation to the position of the robot's TCP.

A step such as described above was performed on the three materials mentioned earlier, see Figures 10.4, A.5 and A.6. During the tests the force and the robot's TCP position were therefore measured and their relation can be seen in Figures 10.1, 10.2 and 10.3 for the three materials respectively. A straight line was estimated in the area of interest using least-squares and its slope corresponds to the stiffness. The stiffness parameters were calculated to $k_1 = 7.40$ N/mm for the cardboard box, $k_2 = 0.48$ N/mm for the rubber foam and $k_3 = 3.72$ N/mm for the cellular plastic.

Conclusions

The stiffness value of $k_1 = 7.3980$ N/mm for the cardboard box is a very rough approximation because the relation between the force and position is clearly not linear, see Figure 10.1. A reason for the nonlinear behavior may be

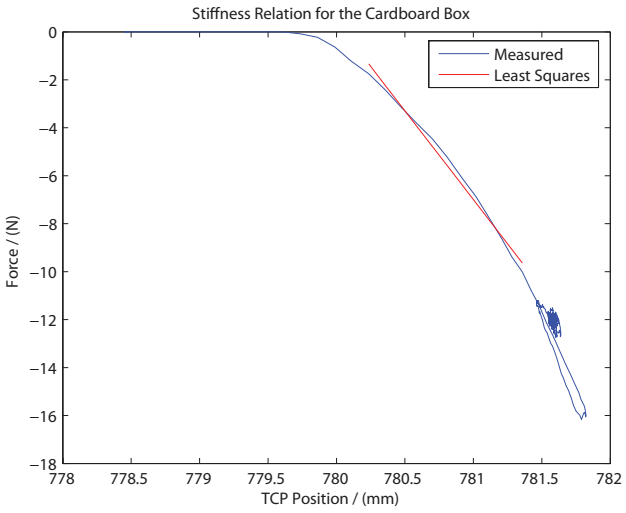

Figure 10.1 The relation between the robot's TCP position and the force exerted on it when trying to reach a point located inside a cardboard box. The blue line is defined by the filtered force signal and the TCP position, both expressed in the robot base frame. The red line is the least-squares estimation for the part of the blue line within the interesting interval.

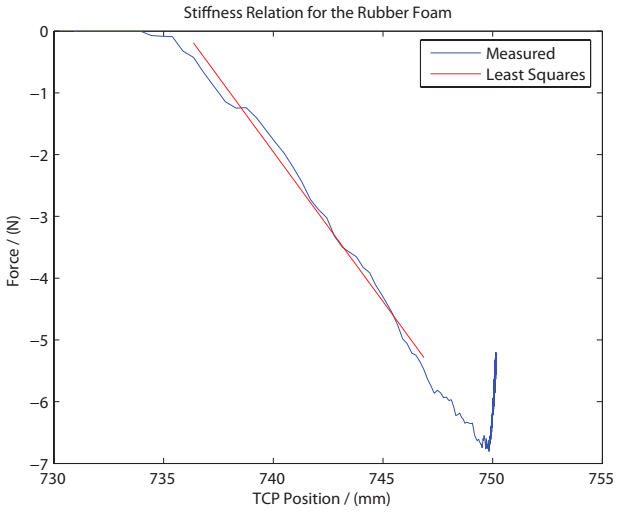

Figure 10.2 The relation between the robot's TCP position and the force exerted on it when trying to reach a point located inside a piece of rubber foam. The blue line is defined by the filtered force signal and the TCP position, both expressed in the robot base frame. The red line is the least-squares estimation for the part of the blue line within the interesting interval.

that the cardboard box was not held perfectly still by its attachment. Another reason is that the box was empty and thereby flexible in a nonlinear way.

For the rubber foam the stiffness value of $k_2 = 0.4848$ N/mm seems more accurate due to the more linear behavior in the force position relation in Figure 10.2. The part of the data that behaves strange is partly due to the piece of rubber foam used being quite small and thin. This caused the material to behave strange if the robot pushed to far inside the piece of rubber foam.

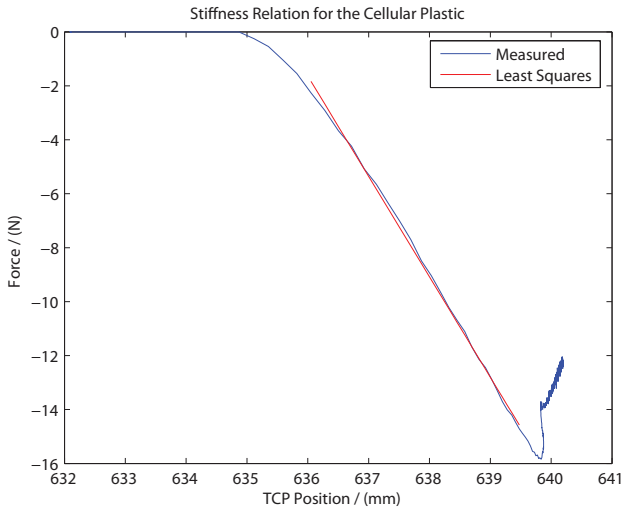

Figure 10.3 The relation between the robot's TCP position and the force exerted on it when trying to reach a point located inside a piece of cellular plastic. The blue line is defined by the filtered force signal and the TCP position, both expressed in the robot base frame. The red line is the least-squares estimation for the part of the blue line within the interesting interval.

In Figure 10.3 the stiffness for the piece of cellular plastic was approximated to $k_3 = 3.7206$ N/mm. The stiffness value lies thereby in between the value for the other two materials used. The piece of cellular plastic was also attached to the cardboard box which introduces some of the nonlinear behavior.

10.2 Choosing the Impedance Control Parameters

The purpose of the impedance control is to avoid large forces and to make the haptic behavior more smooth. The parameters to choose are the M, D and K according to Equation 3.12. Another approach is to use Equations 3.18 and 3.19. Then there are instead the mass, M, the eigenfrequency, ω, and the damping, ζ, to decide. A higher eigenfrequency means that the robot will respond faster to new directions but doing so with more oscillations. A higher damping will make the amplitude of the oscillations decrease faster. The mass decides how the system responds to forces, i.e., a higher mass increases the inertia of the system.

The goal when choosing the parameters was that the robot should be able to handle many different materials. Materials with higher stiffness is harder for the impedance control to work effectively at. Therefore, most of the tests were conducted on the stiffest material at hand, i.e., the cardboard box.

Human Operator Touching a Cardboard Box

The most important factor when deciding which impedance control parameters to choose is how the robot behaves when being controlled by the Phantom. Therefore, a test was conducted where the human operator should try to control the robot in such a way that the Phantom position corresponded to a TCP position inside an obstacle consisting of a the cardboard box in Section 10.1. Furthermore, the operator should try to hold the Phantom as still as possible.

	ω_0 (rad/s)	ζ	M (kg)	D (Ns/m)	K (N/m)
Set 1	8.0	1.0	100	1600	6400
Set 2	8.0	1.0	50	800	3200
Set 3	8.0	0.6	100	960	6400

Table 10.1 The parameter settings used in the different tests.

The cardboard box was positioned with its surface perpendicular to the robot base frame's y-axis. Three tests were performed with the parameter settings seen in Table 10.1. The results can be seen in Figures A.1, A.2 and A.3.

It is hard to draw any decisive conclusions from the test because it is a human that controls the robot. But some general thinking can be done anyway. The force is clearly larger when using the parameters of *Set 1* in comparison to the other set of parameters. This is due to the fact that the user was able to push harder without getting an uncontrollable oscillation in the Phantom. In the tests, forces were also recorded in the x and z directions and that was a consequence of the cardboard box's surface not being perfectly positioned and that the surface is flexible when pushed upon. It is also noticeable that some of the oscillations come from the human operator's inability to hold the Phantom still when forces are feeded back to the Phantom.

Touching a Cardboard Box Automatically

In order to investigate how much the impedance controller itself causes oscillations the same tests as described in Section 10.1 were performed. Only the tests concerning the cardboard box are analyzed here. All three parameter sets defined in Table 10.1 were used. Part of the result is seen in Figures 10.4, 10.5 and 10.6. More data concerning the tests can be seen in Figures A.7, A.8 and A.9.

Using the parameters of *Set 1* gives the smallest oscillations, this despite the fact that it handles the biggest forces. The larger mass makes the inertia bigger and more insensitive to fast force changes. However, the drawback of this is of course that it is not desired that the robot totally ignores forces. Using *Set 2* parameters gives a similar result as for *Set 1* but with bigger oscillations that never disappear which makes this set of parameters unwanted. In the last set of parameters, *Set 3*, the damping is lowered and the behavior of the system is almost identical to when using *Set 2* but with slightly larger oscillations.

Final selection of Impedance Parameters

After considering the tests described earlier in this section, the impedance parameters in *Set 1* were chosen. This because of the overall performance such as taking care of big forces and in the same time minimizing the oscillations. The haptic behavior worked acceptably with these parameters and the robot followed the Phantom in a good manner in free motion, acting quickly and precisely. The parameters are not necessarily optimal, but they define a system that works fairly well in the situations tested and is perceived as natural by a human operator holding the Phantom, though it is hard to prove the latter statement.

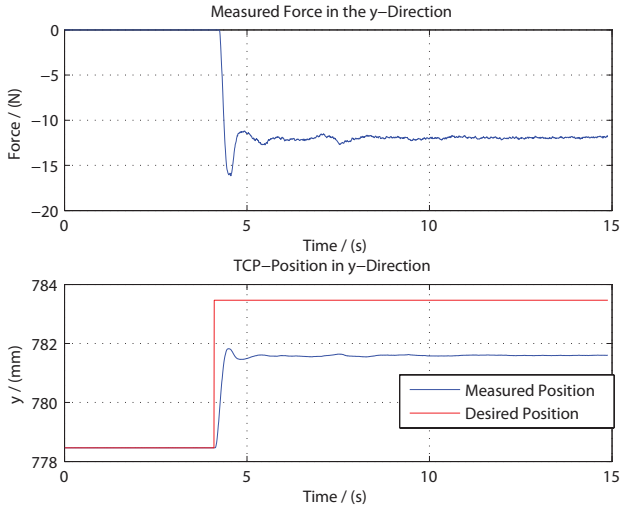

Figure 10.4 The two plots show how the force and TCP position in the robot base frame's y-direction change through time when a step occurs in the desired y-position. The desired position is set to be inside the cardboard box. The blue line is the actual robot position and the red line is the desired position. The set of impedance parameters used is *Set 1*.

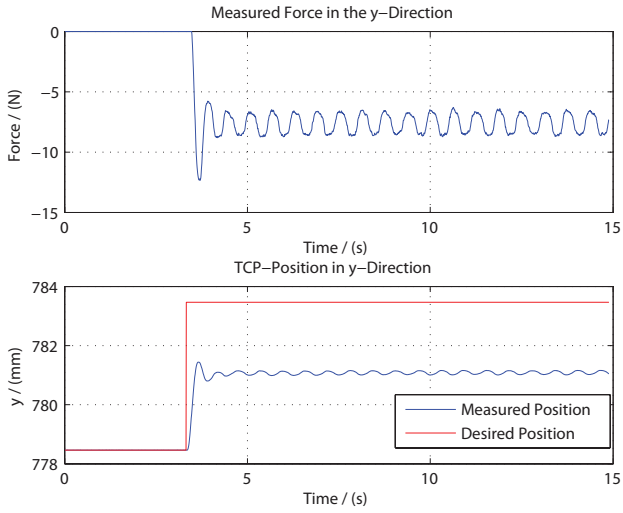

Figure 10.5 The two plots show how the force and TCP position in the robot base frame's y-direction change through time when a step occurs in the desired y-position. The desired position is set to be inside the cardboard box. The blue line is the actual robot position and the red line is the desired position. The set of impedance parameters used is *Set 2*.

10.3 Impedance Control

The impedance control makes the robot behave like a spring-mass-damper system. In this section the behavior of the robot due to the impedance control is investigated.

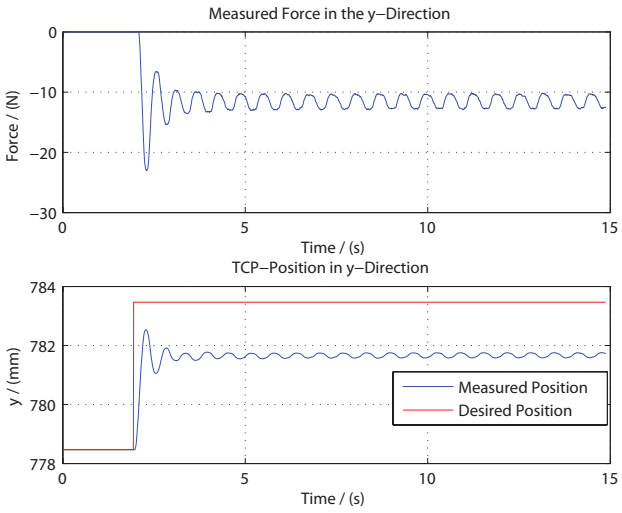

Figure 10.6 The two plots show how the force and TCP position in the robot base frame's y-direction change through time when a step occurs in the desired y-position. The desired position is set to be inside the cardboard box. The blue line is the actual robot position and the red line is the desired position. The set of impedance parameters used is *Set 3*.

Step in the Desired Position

Under the assumption that the robot follows the position reference perfectly the robot can be viewed upon as a spring-mass-damper system. The theoretical step result can be derived, i.e., making a step in the desired position reference gives the where the robot position should be. Three experiments were conducted where a step in the desired position reference for the three different sets of impedance parameters in Table 10.1 was performed. The steps were in the y direction of the robot base frame. The desired velocity reference was assumed to be zero, this in order to avoid calculating the discrete derivative for a step which would have given strange results. The step experiments together with the corresponding theoretical calculations are presented in Figure 10.7. Notice that the theoretical values are the same for the parameter sets *Set 1* and *Set 2*.

In Figure 10.7 it can be seen that the reaction from the robot is a bit slow. The explanation is that it takes some time for the robot to follow the reference values sent from the impedance control. This due to the real inertia and motor limitations for the robot IRB140B. The damping is set to one for the *Set 1* parameters and that forces the robot to avoid any oscillations when following the step. When the damping is set to 0.6 the system behaves faster but with the consequence that it oscillates.

Force Impulse

A force impulse was generated on the robot's TCP in order to see if the robot would behave like the spring-mass-damper system defined. In contrast to earlier tests the desired position is hold fixed but the generated force impulse makes the impedance control to move the robot in opposite direction to the force direction. If the force is only applied for a short time the robot will try to go back to its original equilibrium state. The equilibrium state is the desired position, due to the spring characteristics.

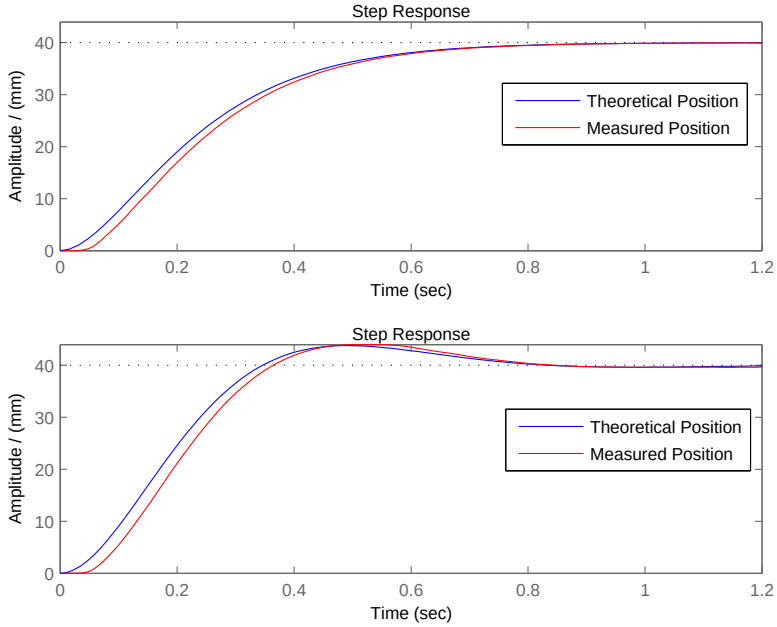

Figure 10.7 The two plots show how the TCP position changes through time when a step occurs in the desired y-position in comparison to the theoretical step response derived from the impedance control used. The upper plot is for the impedance parameters of *Set 1* and the lower for *Set 3*. The parameters of *Set 2* gives precisely the same result as plot one when no forces are acting on the robot.

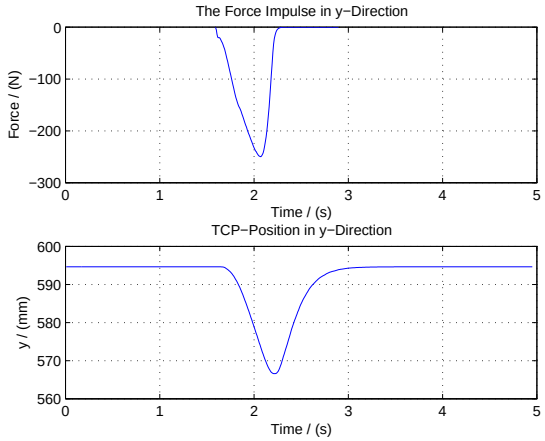

Figure 10.8 The robot's TCP position and the force impulse generated by a human pushing on the TCP in the y-direction. The parts of the position and force presented are in the robot base frame's y-direction. The set of impedance parameters used is *Set 1*.

The test described was performed, with the chosen impedance parameters of *Set 1*, where the robot was pushed in the robot base frame's y-direction. The result is seen in Figure 10.8.

Because of the inertia due to the impedance control and the speed of the robot motors there is a time delay before the force impulse affects the robot in any significant way. The force impulse showed in Figure 10.8 is also delayed

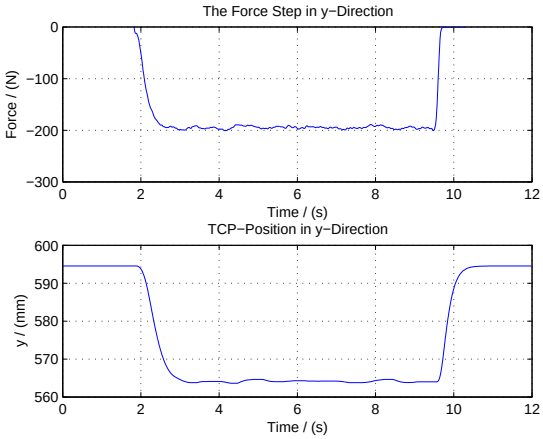

Figure 10.9 The robot's TCP position and the force step generated by a human pushing on the robot TCP with a constant force. The position and force presented is the part directed in the y-direction. The set of impedance parameters used is *Set 3*.

in comparison to the real force due to the time delay introduced by the force sensor and the force filter implemented in the Simulink model.

The damping parameter used was defined to one, and that prevents the robot from oscillating. As seen in Figure 10.8 the robot position returns to the desired position a little bit slow and that is the prize the robot will have to pay to avoid oscillations.

Another force test was also performed where the robot was influenced by a force step generated by a human hand. The result is presented in Figure 10.9. The robot stays in the new equilibrium position which is decided by the stiffness parameter, K, in the impedance controller. The robot shows small oscillations in the position but they are likely to stem from difficulties for the human to generate a constant force.

Lead-Through

That the human operator can move the robot in any wanted direction by holding the end effector is called *lead-through*. The lead-through behavior can be achieved as a byproduct of the implemented impedance controller by setting the influence by the stiffness parameter, K, in Equation 3.12 to zero. The robot will then behave in similar way as an object laying still in water. If pushed in any direction it will go there but it will slow down due to the damping and eventually stop moving.

A test where the stiffness effect was turned off was conducted. The human operator tried, by holding the end effector with the hands, to make the robot move in some kind of circular path. The path for the robot's TCP is seen in Figure 10.10. The impedance parameters used were equivalent to $M = 10$, $D = 31$ and $K = 0$. The result implies that the robot follows the calculated path good at least for moderate movements. By lowering the mass and damping it would be considerably easier to move the robot which could be preferable.

Lead-through is achieved when the stiffness effect is removed from the impedance control. By also setting the damping to zero the robot will simulate how the phenomenon inertia works. The robot then behaves as Newton's first and second law imply. If the robot is pushed it will continue with the given

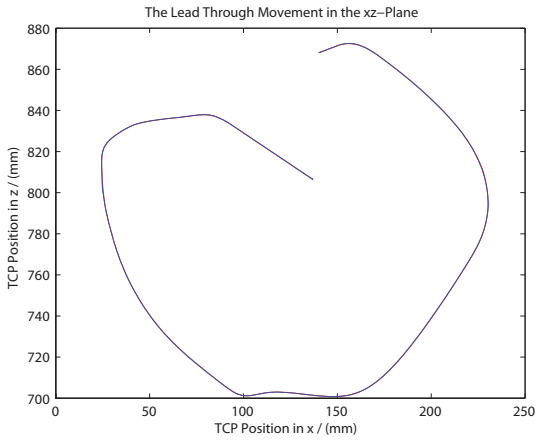

Figure 10.10 The human operator leads the robot through a circular path in the xz-plane. The impedance parameters were set to $M = 10$, $D = 31$ and $K = 0$. In the figure there are two lines, one red and one blue. The red is where the impedance controller wants the robot to be and the red where the robot actually is.

velocity for an infinite time. However, due to physical restraints for a robot, it will eventually stop.

10.4 Controlling the IRB140B With the Phantom

This section will describe the results concerning the master and slave relation, i.e., the Phantom's ability to control the IRB140B. There will also be a retrospect of earlier presented results concerning the haptic behavior.

Different Modes

The Figure 10.11 shows how the robot's TCP position changes when different program modes in the *Phantom Robot Connector* are active. The same test is also seen in Figure A.10 which shows how the joint angles change. During the test, first the *robot2graphic* mode was active, then the *phantom2robot* mode was activated and the operator moved the Phantom around. Lastly, the *Go Home* was chosen which lead back to the *robot2graphic*.

Notice in Figure 10.11 that the *Go Home* mode in the *Phantom Robot Connector* doesn't make the robot's TCP position move along the shortest path to its initial position. Furthermore, it is important that no obstacles are positioned on the path. Because of the fact that the impedance controller has no impact on the robot during the *Go Home* mode the robot will not take any consideration to forces felt by the force sensor.

Fast Phantom Movements

It is of interest to know how well the robot will follow the Phantom position if the Phantom is moved fast. Therefore, a test of this was conducted where the impedance parameter set used was *Set 1*. The human operator tried to move the Phantom as fast as possible without harming it. In Figure 10.12 the movements made in the xz-plane are shown. How the TCP position in the y-direction changes through time is seen in Figure 10.13. The robot joint angles in this test can be seen in Figure A.11.

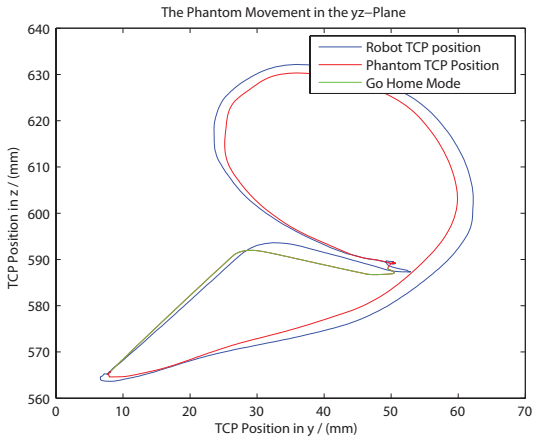

Figure 10.11 The path the robot's TCP position will take when being controlled by the Phantom. It also shows how the *Go Home* routine makes the robot move. The path plotted is a projection on the xz-plane, (the robot was attempted to be controlled by the Phantom so that it would only move in the xz-plane).

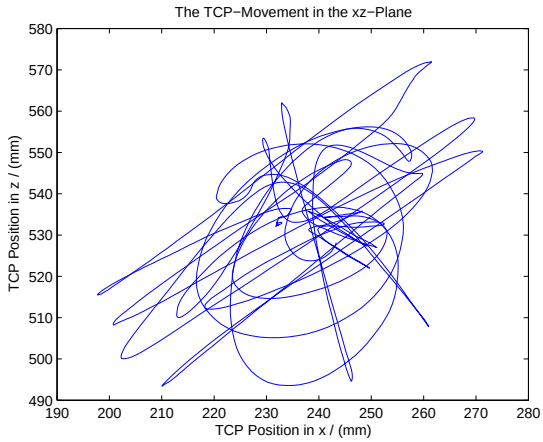

Figure 10.12 The path the robot's TCP position takes when being controlled by the Phantom with fast movements.

The robot's fast movements make the force sensor register its own inertia which is seen in Figure 10.14. The force plotted is the filtered force signal after it has gone through a dead zone of 2.0 N. The figure shows that the inertia is working as a disturbance in the impedance controller for fast movements but that it is negligible for slow movements.

10.5 Virtual Painting

How the virtual painting in the *Phantom Robot Connector* works for a real world object in comparison to virtual objects is described in this section.

First the painting was tested for a real world object. The robot start configuration was set according to Figure 10.15. The obstacle used was a cardboard box and the robot TCP was only a few centimeters from the box and with the

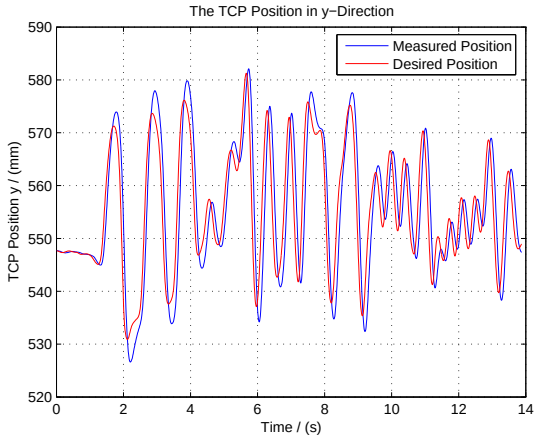

Figure 10.13 How the robot's TCP position in the y-direction changes through time when the robot is controlled by the Phantom with fast movements.

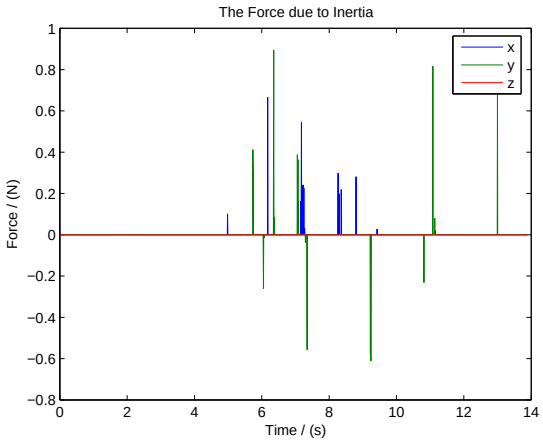

Figure 10.14 The force due to the inertia for the force sensor. The force plotted is the filtered force signal that has gone through a dead zone of 2.0 N. The blue, green and red colors stand for the force measured in the x, y and z-directions respectively.

attack direction pointing towards it. The cardboard box was placed so that the surface was perpendicular to the robot base frame's y-direction.

The actual test was performed by setting the mode in *Phantom Robot Connector* to *phantom2robot* and with the painting enabled. The Phantom was then moved in such a way that the box was touched and the contour made clear. The result of the test is seen in Figure 10.16. Notice the shape of the virtual paint in comparison to the shape of the real box.

A similar test but with virtual obstacles was also conducted. During these tests the *phantom2graphic* mode was active instead of the *phantom2robot* mode. The start relation between the wall and the robot was similar to the one in the previously mentioned test. In order to pretend that the wall was real it was made invisible. The result of the painting of the virtual wall can be viewed in Figure 10.17. Notice that the wall was placed so its surface perpendicular to the x-direction, in comparison to the previous test. However, this has no effect for the outcome other than that the paint is located differently in 3D-space.

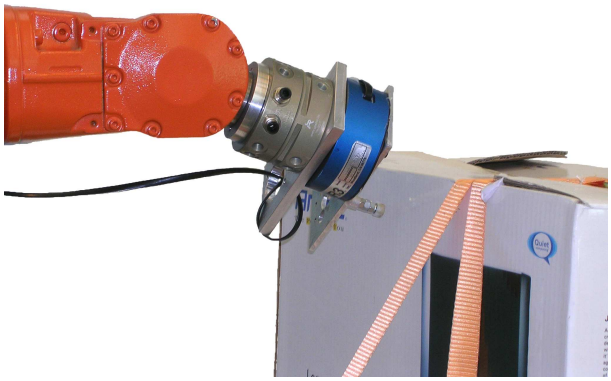

Figure 10.15 The setup during the experiment where the virtual painting was investigated.

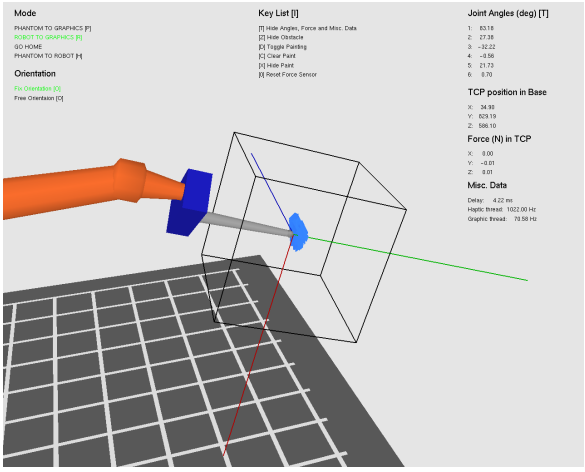

Figure 10.16 The virtual robot corresponding to the real robot in Figure 10.15. The blue colored paint indicates that the robot has made contact with a real-world object.

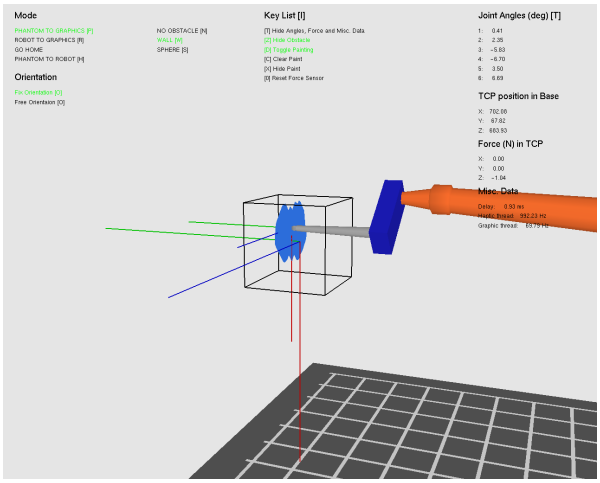

Figure 10.17 The virtual robot touching and painting the virtual wall. The blue colored paint indicates where the virtual robot felt contact forces.

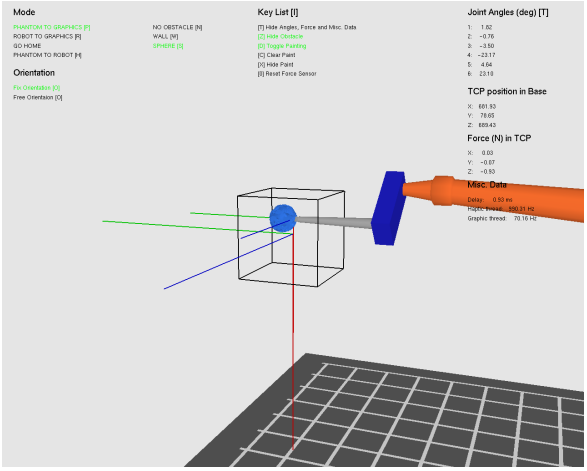

Figure 10.18 The virtual robot touching and painting the virtual sphere. The blue colored paint indicates where the virtual robot felt contact forces.

The virtual sphere was also made invisible and tested upon. However, this result has no equivalent in a real world situation as some parts of the robot were moved inside the sphere when the backside of the sphere was touched and painted, this due to the Phantom tip just being mapped to one single point on the robot and with no consideration taken to other parts of the robot. The test with the virtual sphere can be found in Figure 10.18.

10.6 Haptic Behavior

It is preferable when controlling a robot with a haptic device that large movements by the haptic device give small movements for the robot. This behavior will give the user a delicate control over the robot's movements. Therefore, the movements of the Phantom was mapped to correspond to a factor 0.5 of these of the IRB140B. A smaller factor might have given the operator even smoother control of the robot but the choice made was due to that the limited workspace of the Phantom would have given a very small workspace for the IRB140B. It would had been hard to touch and recognize different obstacles if this choice had not been made.

Earlier results like the one in Figure A.1 suggests that some oscillations are due to the inability of the human operator to keep the Phantom still when forces are felt by the force sensor. This is especially a problem when touching a very stiff object because of the sudden force peak that occurs. The time delays are also a big part of the problem because if the user moves the robot into an object it takes some time before the human senses the force and the longer it takes the further inside the object the robot would be and the larger the force peak will get. If the time delays would be negligible the problem would probably have been smaller. However, making the time delays smaller is hard and an easier way of improving the performance could be to decrease the workspace for the IRB140B in relation to the Phantom's workspace. This would lower the robot speed and thereby the amplitude of the contact force peaks measured by the force sensor. But as mentioned earlier this solution may not be optimal due to other dilemmas.

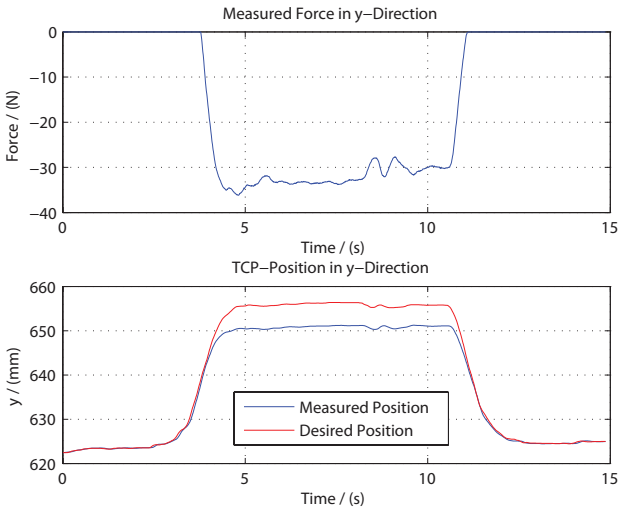

Figure 10.19 The two plots show how the force and TCP position in the robot base frame's y-direction change through time when the Phantom is controlled by a human user. The robot is moved in such a way that it touches a piece of cellular plastic. The blue line is the robot position and the red line is the desired position. The impedance parameters used is from *Set 2* in Table 10.1.

The solution used in this thesis in order to give the human operator a chance of handling the force feedback was to lower the force sent to the Phantom with a predefined factor. The factor was chosen to be 20, i.e., 20 N measured by the force sensor would be felt as 1 N by the user. The drawback of this solution is that objects with low stiffness would almost not be felt at all by the human operator.

There are bigger oscillations for the IRB140B when the material touched is the cardboard box than for example the piece of rubber foam. The problem when touching the rubber foam is that it is hard to feel the contour due to the small contact forces. Therefore, a test was performed where the material touched was instead a piece of cellular plastic. The impedance parameters used during the test was *Set 1*. Cellular plastic was used because the oscillations are smaller than for the cardboard box and that the contour is easier felt by the human operator than in the rubber foam case.

In one test the user tried to hold the Phantom so that the IRB140B was pressed into the object with a constant force. The test results are presented in Figure 10.19. The figure shows that the force signal is quite stable and that no oscillations to speak of are present. This suggests that the impedance parameters in *Set 1* is a good choice when the material to be touched is one of the three used in the tests described in this chapter. The haptic feeling is distinct, which is shown by the sharpness of the force signal in Figure 10.19 and that the force curve follows the movement from the Phantom well. Further data from the test can be seen in Figure A.4.

Conclusions

There are many factors that affect the haptic feeling for the human operator. There are practical limitations due to the IRB140B's motors, the Phantom's motors, the force sensor, signal delays, etc. There are other limitations due to other considerations, e.g., the contact force controller implemented in order

to avoid high forces or having a large robot workspace in order to have the ability to feel the contour of a large object.

The parameters used for the impedance controller was *Set 1*. Evidently there are other parameter settings that would had worked good, perhaps even better than the one chosen here. In order to give the human operator a good perception of the surroundings of the IRB140B through the sense of touch (haptic feedback) the ideal thing would be to let the robot follow the Phantom perfectly. However, due to the practical limitations earlier mentioned this is not possible. Additional limitations come from the implemented impedance controller. It is important that the user will have the most influence on the robot's behavior to achieve a good haptic behavior.

11. Summary

The target of this thesis was to implement a haptic interface for a contact force controlled industrial robot. The haptic device used was a Phantom Premium A from *SensAble Technologies, Inc.*, see Section 2.1, and the industrial robot was an IRB140B from *ABB*, see Section 3.1. The haptic interface can be divided into two parts, one theoretical where a mathematical mapping from the current state of the haptic device to a corresponding robot state was done and one practical part where the mathematical mapping was implemented through programming in C++ and C and by using Matlab's Simulink. The theoretical mapping was done in a general way and could hence be used for other master and slave systems than the Phantom and the IRB140B.

A computer program called *Phantom Robot Connector* was written in C++ with the responsibility to communicate with the Phantom directly, see Chapters 8 and 9. The program determines a new set of joint angles for the IRB140B based on the current state of the Phantom and will set the force feedback signal to the Phantom. It also has a graphics part that features a virtual robot that is a simplified version of the IRB140B, but with all joints and links represented correctly. The graphical robot has its own standard Denavit-Hartenberg representation.

The Simulink model created, see Chapter 7, was transformed into C code and downloaded to the robot's axis computer. It receives values from the *Phantom Robot Connector* and the server belonging to the force sensor mounted on the robot. In the model the contact force controller that was to be implemented in this thesis was created. It is an impedance controller with both position and velocity references. They are based on the new set of robot joint angles suggested by the *Phantom Robot Connector*. The impedance controller can be altered during runtime to enable lead-through of the robot, see Section 10.3. Parameters in the Simulink model can be altered during runtime through a graphical user interface (GUI), see Chapter 5.

11.1 Main Results

To test the full implementation of the system in this thesis, a series of different tests were performed, see Chapter 10. The most important tests and results are presented in this section.

Impedance Controller

The impedance controller finally selected is described in Section 10.2. It is well damped and has a quite large mass. The latter will decrease the effects of the quantization and resolution problems with the force sensor used, see Section 3.3. A step response in the desired position for the IRB140B compared to that of the theoretical spring-mass-damper system can be seen in Figure 10.7. A step was also performed so that the robot will attempt to reach a position inside a piece of cellular plastic. The result can be seen in Figure 10.19.

Controlling the IRB140B with the Phantom

Using the impedance controller described above, the robot was controlled by an operator holding the Phantom. In Figure 10.11 the robot's TCP position is

shown for a sequence where the robot is at rest, the control by the Phantom is started, the Phantom is moved around and the control of the robot is stopped to make it return to its original configuration. The figure hence shows how the control of the robot through the Phantom is commenced and ceased without any bumps in the robot's position.

The robot has a speed limit it can not exceed. If so, the brakes will be locked automatically. The question is if the human operator holding the Phantom can move it so quickly that the robot will exceed the speed limit? In Figure 10.12 the result of a test where the operator moved the Phantom very fast is seen. The robot did not exceed the speed limit.

Haptics

To test the haptic behavior of a master and slave system is not a trivial task as there is no simple way to express how good the performance is in the eyes of the user. Something that could be measured is however how the robot will react if it is pressing against an obstacle and do this under two different circumstances. First, to let a human operator induce the step through the Phantom and to apply the force feedback, i.e., to run the full system. Second, to do it by inducing an automatic step in the desired position in the impedance controller and not listening to the Phantom or sending any force feedback to it. In Figure A.1 the first case is shown and in Figure 10.4 the second can be seen. There will clearly be a difference in behavior according to the two figures which has to be explained by the user's inability to hold the Phantom perfectly still, and by that keeping the desired position constant, when the force feedback kicks in.

Time Delay

The round trip time from the *Phantom Robot Connector* to the robot and back again is lying just above 4 ms in average. As that almost corresponds to the sample time of the Simulink controller, which runs at 250 Hz, it is assumed that the network communication does not decrease the performance of the system that much.

11.2 Future Work

Due to the limited time span of a master thesis, some issues will remain unresolved. There are some things that would have been added, improved or redone in a different way, had there been more time.

Extra Functionalities

Some extra functionalities that are not featured in this thesis could be added, e.g., a solid lead-through implementation that would enable the operator to lead the robot to the place where the haptic experiments will take place. As of now, lead-through comes as an extra bonus with the impedance controller, see Section 10.3, through clever choices of some parameters in the impedance controller that can be set via the GUI. It would be very handy for an operator if the robot would not have to be jogged into a suitable position, but could instead be lead there by hand before any experiments.

To further extend the workspace of the robot while being controlled by the Phantom a function for repositioning the robot during control with the

Phantom could be implemented. This would be a much faster way to move the workspace for the robot than to re-jog it. Introducing this would maybe take a bit of reconstruction of the current version of the *Phantom Robot Connector* but is not an impossible task.

The Influence of Time Delays

An area that was not investigated was the occurrence of time delays in the control loop. Data is being sent over a network and the Simulink model runs at a slower pace than the *Phantom Robot Connector*. These are both circumstances that imply that a time delay will appear. Time delays in master and slave systems can, and often will, result in instability of the system [4]. Especially when a human operator is involved in the loop.

As can be seen in Chapter 10 even the best found parameter settings for the impedance controller would not make the system entirely without any oscillations if a human operator would hold the Phantom, see Figure A.1. It would be of interest to see if a time delay analysis would lead to better performance in this case. A possible solution that could be implemented is found in [4].

Miscellaneous Things to Improve

Of course, there exist a number of things and functions that could be improved. For example the free orientation option in the *Phantom Robot Connector* will not work on the real robot as well as the fix orientation. The risk of an *overspeed error* in the robot system is highly increased in the orientation of the Phantom is considered. If the mapping would be done differently, let's say a full 360 degree twist of the *pen* would instead correspond to only a 180 or a 90 degree twist of the robot's sixth joint, the risk of an *overspeed error* would decrease but it would perhaps instead feel awkward to the user when controlling the robot. Therefore the fix orientation was focused on. Due to that fact, the Simulink model was not created in such a way that the free orientation mode would work as good as the fixed. Hence, some things in the Simulink model would need to be updated if the free orientation mode is to be improved in the future.

Another thing that leaves room for improvements is the graphic representation of the IRB140B. It has all the links correct and the joints in the right places, but it could be designed to look more like the real robot.

The Painter class allows the user to paint any virtual or real objects felt with the Phantom. The current implementation is far from perfect and there are many things to improve, e.g., how to store the points where to paint between cycles in the graphics loop, when to add a new point and how to paint the points added.

12. Acknowledgements

First of all, we would like to express our great gratitude to our supervisor Anders Robertsson. Always nice and positive minded he is ready to help out in his trademark buoyant mood. He has shown great interest in our work and has always been eager to understand exactly how we have tried to solve a problem and thereafter done his best to help us where we are, instead of where we might be, as easily happens in other supervising situations. Frankly, it is hard to imagine a better supervisor for this thesis than Anders.

In the beginning of this thesis a lot of work with the Phantom and the 3D-graphics was carried out at the Virtual Reality Lab at Ingvar Kamprad Design Centrum. Helping us getting started there both with the programming itself and with other practical issues, and thereby acting as kind of a co-supervisor for a while, was Joakim Eriksson who we would like to thank a lot for that. We would also like to thank Joakim for letting us bring a computer from the Virtual Reality Lab, to which the Phantom was connected and installed on, to the Robotics Lab at the Department of Automatic Control.

Another person involved in moving the Phantom to the Robotics Lab is Kirsten Rassmus-Gröhn from Certec, the department to which the Phantom originally belongs to. Thank you for your approval and assistance in the relocation of the Phantom.

Two persons from the Department of Automatic Control that also deserves special thanks are Leif Andersson and Anders Blomdell. When the Phantom was to be installed at the Robotics Lab, some materiel issues arose. Leif Andersson was then a great help and provided us with the paraphernalia needed to set up a proper workstation. Leif has also helped us with some questions concerning LaTeX, as we both were novices when it came to that area at the beginning of the thesis.

Anders Blomdell has helped out with different problems primarily concerning the network programming part of the thesis, e.g., how the communication with the downloaded Simulink controller should be done, to install the force sensor server and to do maintenance work on the logging of data from the Simulink controller.

In all of the above mentioned, Anders Robertsson has also taken a great part. Both when it comes to contacting people and taking initiatives in matters but also when it comes to helping us with any kind of issues that may have come up, be it with LaTeX, programming, thoughts on problem solving, haptics, robotics or any other general concerns we might have had during this journey. So once more, thank you Anders.

Finally we would like to address a thank you to all the other persons besides the above mentioned that have helped us getting where we got with this master thesis. Also to all of you who have followed our work and asked questions and shown interest in our achievements, thank you.

References

[1] ABB. *www.abb.com*.

[2] ABB Automation Technologies AB, Robotics. *Product Specification – Articulated Robot 3HAC9041-1*, 2004.

[3] ABB Automation Technologies AB, Robotics. *Operator's Manual – IRC5 with FlexPendant*, 2005.

[4] Robert J. Anderson and Mark W. Spong. Bilateral control of teleoperators with time delay. *IEEE Transactions on Automatic Control*, 34(5):494–501, 1989.

[5] Christopher Brown and Michael Barr. Introduction to endianness. *Embedded Systems Programming*, pages 55–56, January 2002.

[6] John J. Craig. *Introduction to Robotics – Mechanics and Control*. Addison-Wesley Publishing Company, 1986.

[7] Isolde Dressler. *Force control interface for ABB S4*. LTH.

[8] Fredrik Eriksson and Marcus Welander. *How to Connect the Phantom with the IRB140B*, 2009.

[9] Xuejian He and Yonghua Chen. Six-degree-of-freedom haptic rendering in virtual teleoperation. *IEEE Transactions on Intstrumentation and Measurement*, 57(9):1866–1875, 2008.

[10] N. Hogan. Impedance control: An approach to manipulation, part i – theory. *J. Dynam. Syst. Meas. Control*, vol. 107, 1985.

[11] N. Hogan. Impedance control: An approach to manipulation, part ii – implementation. *J. Dynam. Syst. Meas. Control*, vol. 107, 1985.

[12] N. Hogan. Impedance control: An approach to manipulation, part iii – applications. *J. Dynam. Syst. Meas. Control*, vol. 107, 1985.

[13] IPR (Intelligent Peripherals for Robots). *www.ipr-worldwide.de*.

[14] Dong-Soo Kwon Jee-Hwan Ryu and Blake Hannaford. Stable teleoperation with time-domain passivity control. *IEEE Transactions on Automatic Control*, 20(2):365–373, 2004.

[15] JR3, Inc. *www.jr3.com*.

[16] Rajni V. Patel Mahdi Tavakoli, Arash Aziminejad and Mehrdad Moallem. High-fidelity bilateral teleoperation systems and the effect of multimodal haptics. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 37(6):1512–1528, 2007.

[17] Novint Technologies, Inc. *home.novint.com*.

[18] Tomas Olsson. *High-Speed Vision and Force Feedback for Motion-Controlled Industrial Manipulators*. PhD thesis, Department of Automatic Control, Lund University, Sweden, May 2007.

[19] Open GL. *www.opengl.org*.

[20] SensAble Technologies, Inc. *OpenHaptics Toolkit – version 2.0 – API Reference.*

[21] SensAble Technologies, Inc. *OpenHaptics Toolkit – version 2.0 – Programmer's Guide.*

[22] SensAble Technologies, Inc. *www.sensable.com.*

[23] Mark W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley & Sons, Inc., 1989.

[24] Nicolas Turro, Oussama Khatib, and Eve Coste-Maniere. Haptically augmented teleoperation. *Proceedings of the 2001 IEEE International Conference on Robotics & Automation, Seoul, Korea*, pages 386–392, 2001.

A. Data Collected During Experiments with the IRB140B

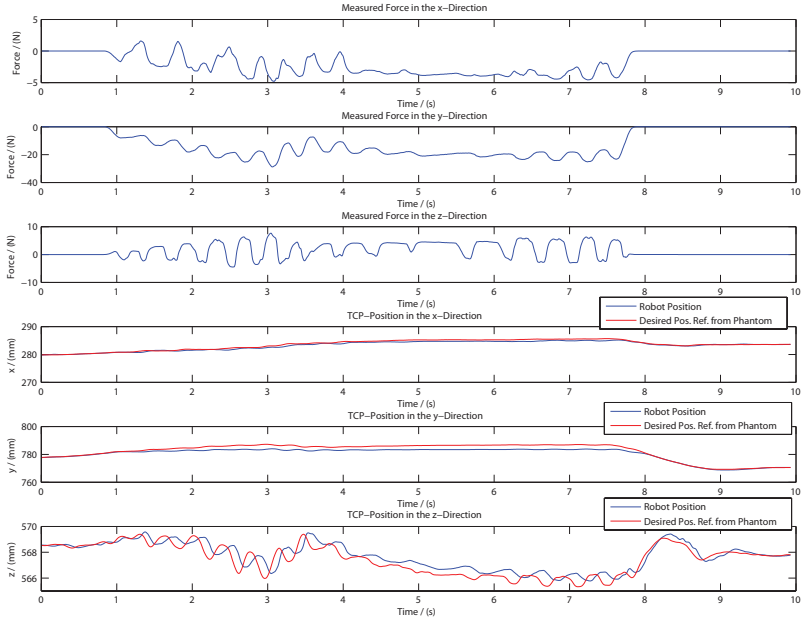

Figure A.1 How the force and TCP position change through time when a human operator tries to hold the phantom still when the desired robot position is inside the cardboard box. The impedance parameters used is *Set 1*.

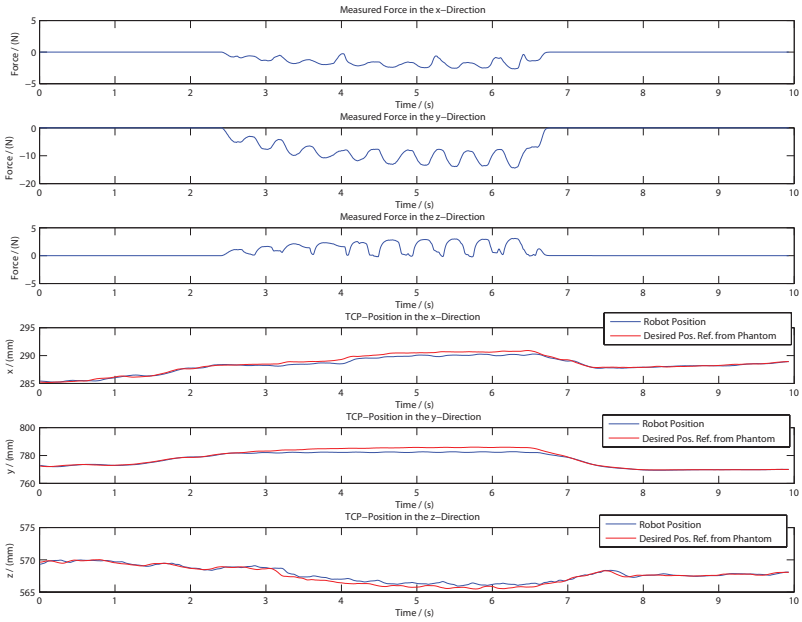

Figure A.2 How the force and TCP position change through time when a human operator tries to hold the phantom still when the desired robot position is inside the cardboard box. The impedance parameters used is *Set 2*.

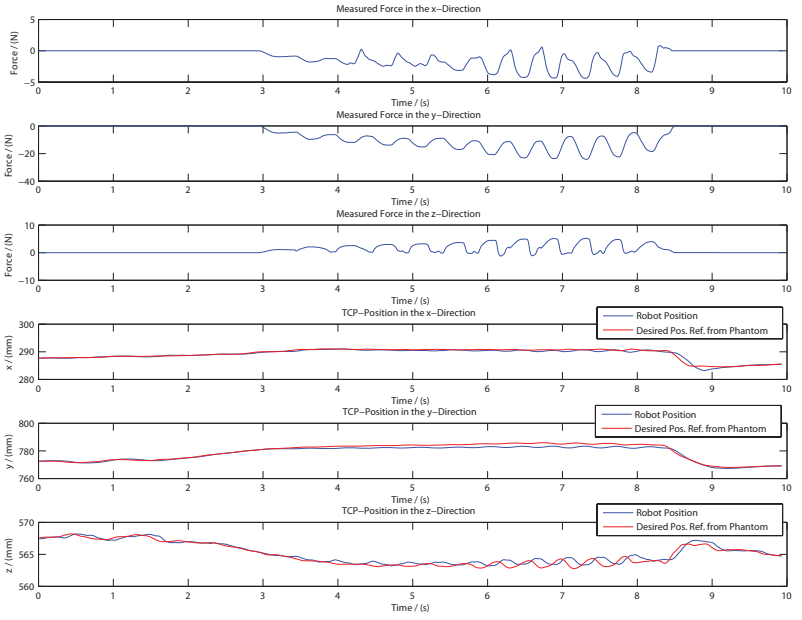

Figure A.3 How the force and TCP position change through time when a human operator tries to hold the phantom still when the desired robot position is inside the cardboard box. The impedance parameters used is *Set 3*.

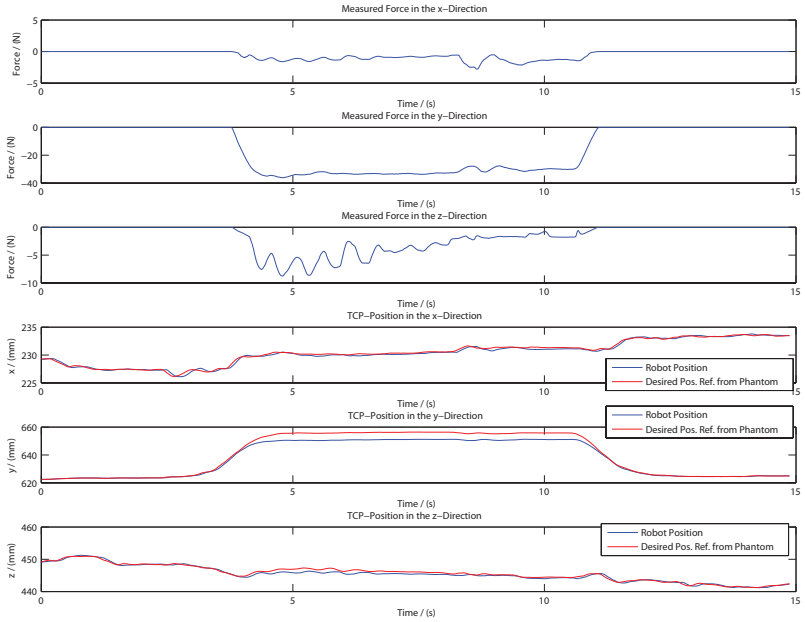

Figure A.4 How the force and TCP position in the robot base frame's y-direction change through time when the Phantom is controlled by a human user. The robot is moved in such an way that it touches the cellular plastic. The blue line is the robot position and the red line is the desired position. The impedance parameters used is *Set 2*, from Table 10.1.

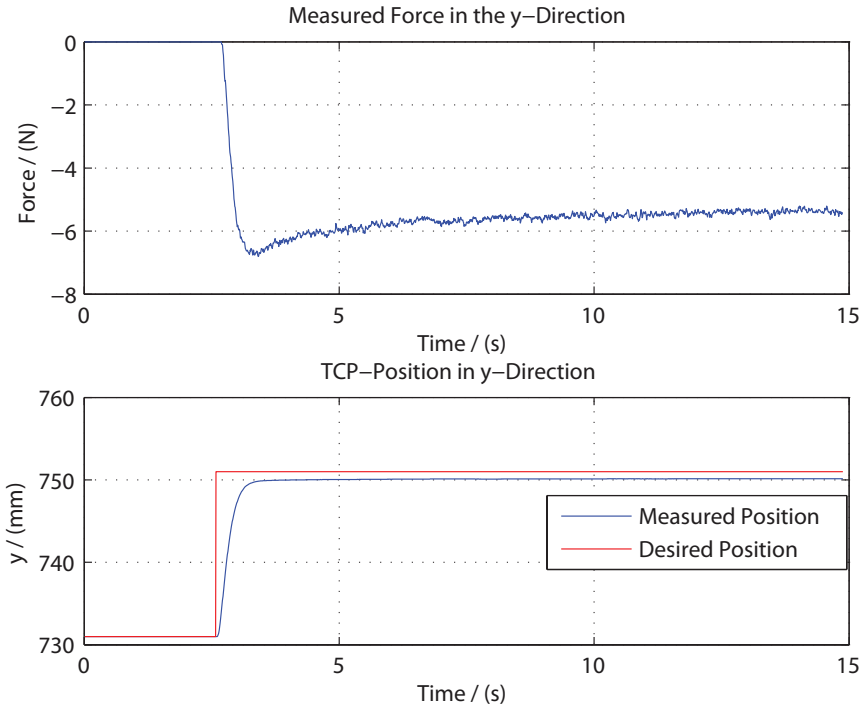

Figure A.5 How the force and TCP y-position change through time when a step occurs in the desired y-position. The desired position is set to be inside the rubber foam. The impedance parameters used is *Set 1*.

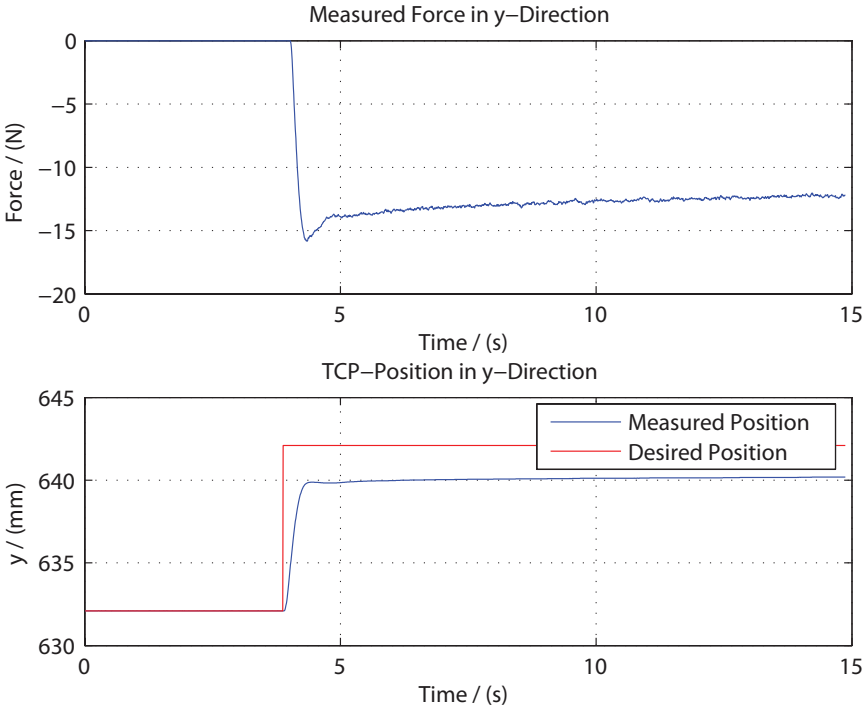

Figure A.6 How the force and TCP y-position change through time when a step occurs in the desired y-position. The desired position is set to be inside the cellular plastic. The impedance parameters used is *Set 1*.

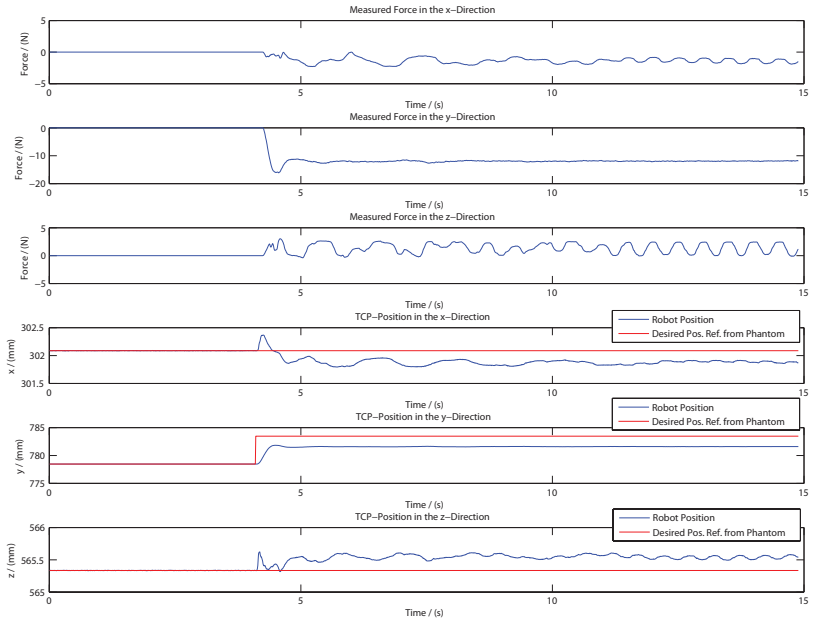

Figure A.7 How the force and TCP position change through time when a step occurs in the desired y-position. The desired position is set to be inside the cardboard box. The impedance parameters used is *Set 1*.

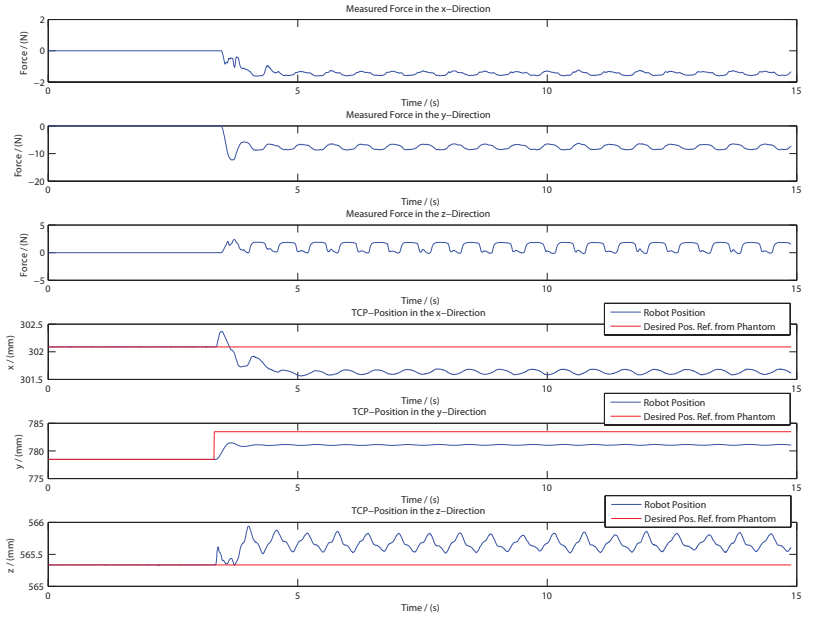

Figure A.8 How the force and TCP position change through time when a step occurs in the desired y-position. The desired position is set to be inside the cardboard box. The impedance parameters used is *Set 2*.

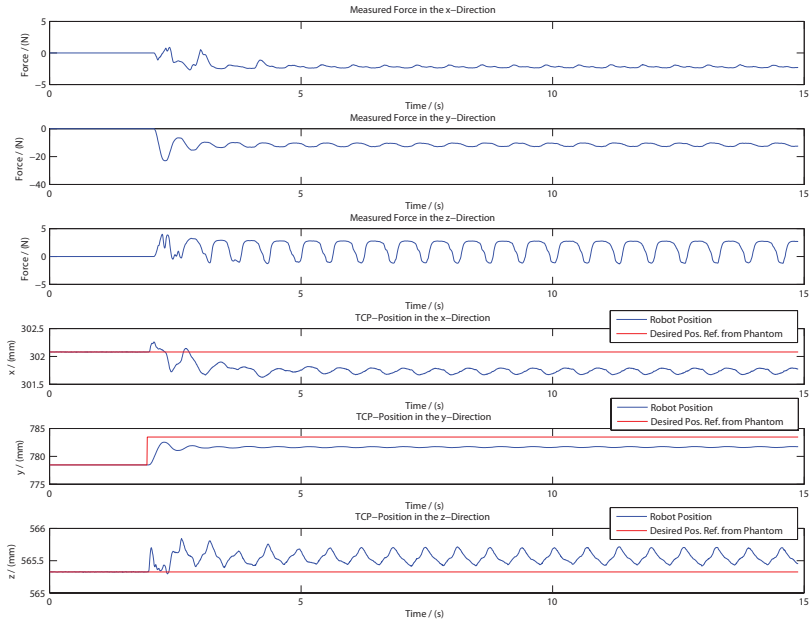

Figure A.9 How the force and TCP position change through time when a step occurs in the desired y-position. The desired position is set to be inside the cardboard box. The impedance parameters used is *Set 3*.

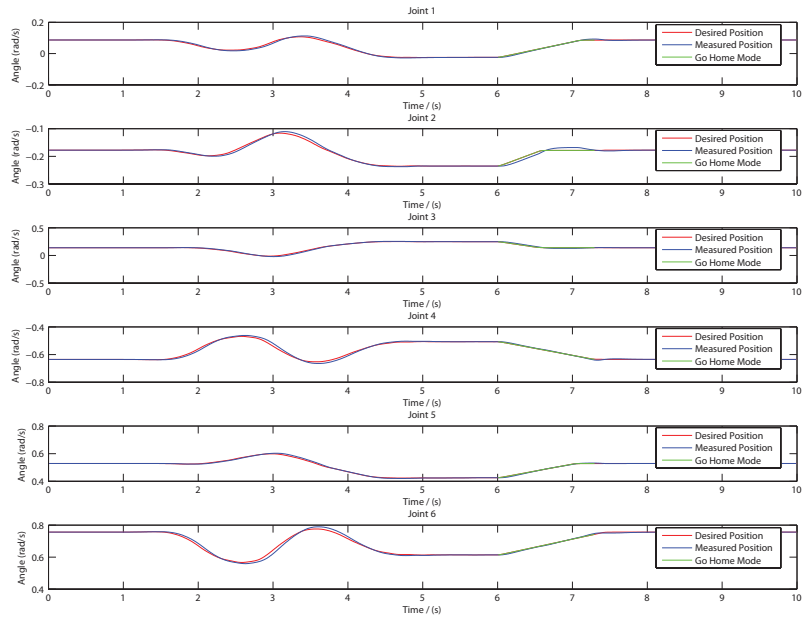

Figure A.10 The joint angles when a human operator moves the Phantom and different modes in *Phantom Robot Connector* are activated. The blue line is the actual robot position and the red line is the joint angles on where the Phantom want's the robot to be.

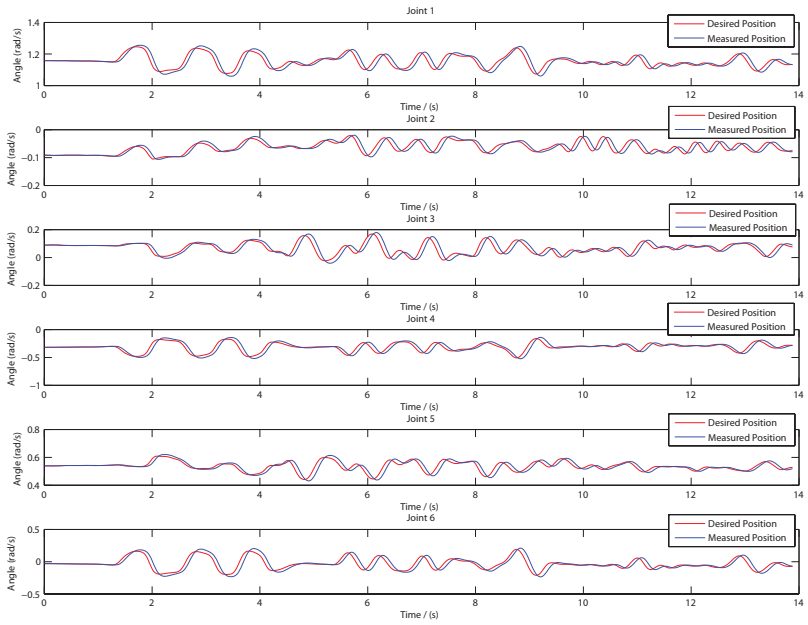

Figure A.11 The joint angles when a human operator moves the Phantom in a fast way. The blue line is the actual robot position and the red line is the joint angles on where the Phantom want's the robot to be.

B. Phantom Robot Connector Key List

Key	*Function*
[P]	Changes to *phantom2graphic* mode. Only available in *robot2graphic* mode.
[R]	Changes to *robot2graphic* mode.
[H]	Changes to *phantom2robot* mode. Only available in *robot2graphic* mode.
[O]	Changes between fix orientation and free orientation. Only available in *robot2graphic* mode.
[W]	Makes a virtual wall appear. Replaces the virtual sphere, if present. Only available in *phantom2graphic* mode.
[S]	Makes a virtual sphere appear. Replaces the virtual wall, if present. Only available in *phantom2graphic* mode.
[N]	No obstacle. Removes any virtual obstacle currently present.
[Z]	Hide/view the virtual obstacle currently present. Hiding a virtual obstacle will not affect the forces from it.
[0]	Resets the robot's force sensor.
[D]	Paint the surface of an obstacle, virtual or real. Paints whenever $\|F_p\| > 0.2$, where F_p is the force on the Phantom.
[C]	Clears the all paint.
[X]	Hide/view the paint. This does not remove the paint as [C] does.
[I]	Show a list of commands.
[T]	Hide/View the data i.e., the angles, the force and the miscellaneous data column.
[F]	Toggles full screen.
[←] and [→]	Move horizontally around a sphere to change the view perspective.
[↑] and [↓]	Move vertically around a sphere to change the view perspective.
[Page Down/Up]	Zoom in and out towards the robot.
[Esc] or [Q]	Exit the *Phantom Robot Connector*

Table B.1 A list of all keyboard buttons used in the *Phantom Robot Connector*.

C. Simulink

Impedance Control Embedded Matlab Function Code

```
function acc = fcn(posRef,pos,velRef,vel,force,M,D,K,...
    posDeltaLimit,impSwitch)
% Calculates the desired acceleration for the system to...
    behave such as a
% simple spring-mass-damper system with parameters M, D...
    and K.

limit = posDeltaLimit;
deltaPos = pos-posRef;

for i=1:3
    if abs(deltaPos(i)) < limit;
        deltaPos(i)=0;
    elseif deltaPos(i) > 0
        deltaPos(i) = deltaPos(i) - limit;
    else
        deltaPos(i) = deltaPos(i) + limit;
    end
end

if impSwitch<0.5
    acc = [0 0 0]';
else
    acc = (force - D*(vel-velRef) - K*deltaPos)/M;
end
```

Impedance Parameters Embedded Matlab Function Code

```
function [D,K] = fcn(omega,z,M)
%Sets the impedance parameters based on the ...
    characteristic polynomial.
% 0=s^2+2*z*w*s+w^s <=> 0=M*s^2+D*s+K

D = 2*M*z*omega;
K = M*omega*omega;
```

Update Position Referens Embedded Matlab Function Code

```
function tcp2base_new = fcn(tcp2base,posRef_new)

tcp2base_new = tcp2base;
tcp2base_new(4) = posRef_new(1);
tcp2base_new(8) = posRef_new(2);
tcp2base_new(12) = posRef_new(3);
```

C.1 Simulink Library Blocks

Get TCP Position Embedded Matlab Function Code

```
1  function rTCPposition = fcn(rTCP2rBase)
2  % Extracts the position from a given row-major ↩
       transformation matrix.
3
4  rTCPposition = [rTCP2rBase(4) rTCP2rBase(8) rTCP2rBase↩
       (12)]';
```

Angle Fix Embedded Matlab Function Code

```
1  function y = fcn(u)
2  % Redefines the third joint angle so that it is ↩
       expressed relative the
3  % horizontal plane.
4
5  y = u;
6  y(3) = u(3)+u(2);
```

Inverse Angle Fix TCP Position Embedded Matlab Function Code

```
1  function y = fcn(u)
2  % Redefines the third joint angle so that it is no ↩
       longer expressed
3  % relative the horizontal plane.
4
5  y = u;
6  y(3) = u(3)-u(2);
```

Forward Kinematics – arm joints -> tcp

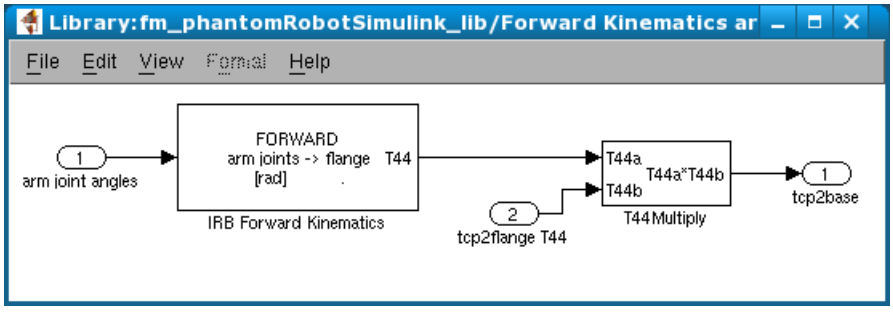

Figure C.1 This subsystem will return the transformation from the robot's TCP frame to the robot's base frame.