ISSN 0280-5316
ISRN LUTFD2/TFRT--5741--SE

Cooperative Robots

Chin Yuan Chong

Department of Automatic Control
Lund Institute of Technology
March 2005

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

Document name
MASTER THESIS

Date of issue
March 2005

Document Number
ISRN LUTFD2/TFRT--5741--SE

Author(s)
Chin Yuan Chong

Supervisor
Rolf Johansson and Anders Robertsson at LTH in Lund

Sponsoring organization

Title and subtitle
Cooperative Robots (Samarbetande robotar)

Abstract
The topic of this master thesis is cooperative robots. The master and slave configuration is to be used for cooperative tasks and the task chosen is to move a flexible beam together. A trajectory is fed to the master robot to move it to the desired positions. The slave robot then follows the movement of the master robot by force control using force measurements from a force sensor.
All experiments are performed in the Robotics Laboratory at the Department of Automatic Control at Lund Institute of Technology. The robots used are an ABB IRB 6 robot and an ABB IRB 2000 robot. The
ABB IRB 6 robot is assigned to be the master robot and the ABB IRB 2000 robot as the slave robot. The ABB IRB 2000 robot is equipped with a force and torque sensor of type JR3 for force control.
The thesis can be divided broadly into three parts. The first part is a background study of the robot systems and kinematics, on path and trajectory generation and also on the various robot force control schemes.
The second part is about optimal trajectory generation. The desired path is first expressed in terms of path index in Cartesian space. It is then converted to joint space with variable step size to ensure that the deviation from the original path is kept within a given limit. The optimal trajectory is then generated using Linear Programming. The third part consists of the application of parallel force/motion control
schemes to the slave robot to enable it to follow the master robot. The motion controller is simply the built-in controllers of the IRB 2000 robot. The force controllers used are direct force, impedance and admittance controllers. The schemes are first tested in Matlab Simulink where the robots are simulated by a first order system for each joint. The force measurements are generated by a contact model which simulates the contact forces and torques between the gripper of the IRB 2000 robot and the beam. The resulting joint values are then sent to a Java visualization program written in the platform Eclipse.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
72

Recipient's notes

Security classification

Contents

Chapter 1

Introduction

1.1 Problem Formulation

The topic of this master thesis is cooperative robots. The master and slave configuration is to be used for cooperative tasks and the task chosen is to move a flexible beam together. A trajectory is fed to the master robot to move it to the desired positions. The slave robot then follows the movement of the master robot through a force control scheme.

To solve this problem, it is divided into several parts:

- A background study of the robot system and kinematics, Chapter 2 and also of robot control schemes which are suitable for cooperative tasks, Chapter 4.
- Path and trajectory generation, Chapter 3.
- Implementation of the robot control schemes in Matlab Simulink and simulations to check the performances of these schemes, Chapter 5.
- Robot experiments to verify the results of the simulations, Chapter 6.

Due to some practical problems which occurred during the thesis work, robot experiments cannot be done to verify the results of the simulations. These problems are discussed briefly in Section 6.1.

1.2 Experimental Platform

All experiments are performed in the Robotics Laboratory at the Department of Automatic Control at Lund Institute of Technology. The industrial robots used are an ABB IRB 2000 robot and an ABB IRB 6

Figure 1.1: The ABB IRB 2000 robot.

robot. The IRB 2000 robot is equipped with a wrist mounted force and torque sensor of type JR3.

1.2.1 ABB IRB 2000 Robot

One of the robots used in the thesis is the 6-DOF (degrees of freedom) ABB Industrial Robot 2000, IRB 2000. It has seven links connected by six joints, as shown in Figure 1.1. There is a mechanical coupling between joint five and six. Joint one, four and six are cylindrical joints while joint two, three and five are revolute joints. Joint one turns the robot by the base, joint two moves the lower arm back and forth, joint three moves the upper arm up and down. Joint four turns the wrist unit, joint five bends the wrist unit around its centre while joint six turns the force sensor and the end-effector which are mounted on the tip of the wrist. Both the force sensor and the end-effector are rotated 230 degrees relative to the positive direction of joint six.

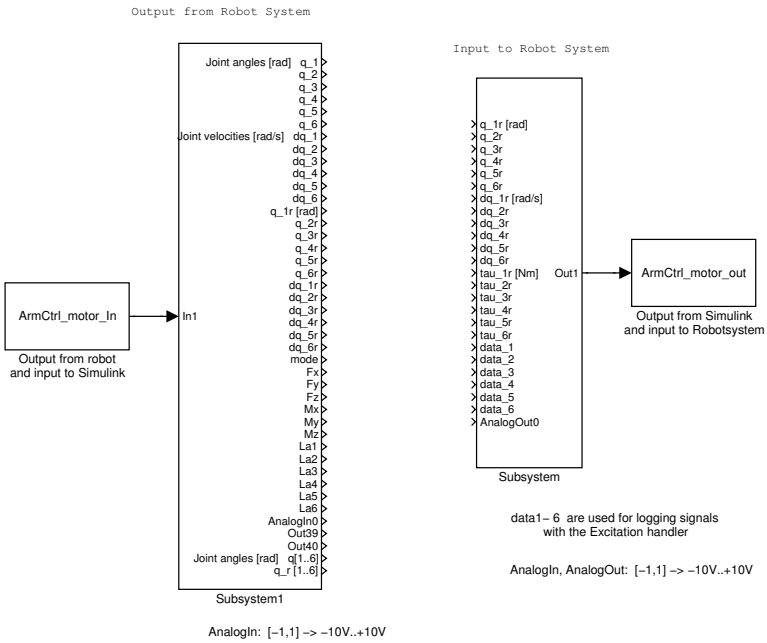


Figure 1.2: The Simulink template used for control scheme implementations

1.2.2 Robot Control System for IRB 2000

All implementations of the control schemes are done in Simulink using the real-time workshop in Matlab. The Simulink template in Figure 1.2 for arm control is available for this purpose.

The Simulink template has twenty five inputs and forty outputs. When designing a controller, the inputs $torque_{ref}$ can be used as input for the control signal and the built-in controllers, as explained in Section 4.1, will be turned off. The outputs data1 to data6 can be used for logging signals. The inputs L1 to L6 can be used to change parameters during run-time. When L5 is set to a value more than 100, the force sensor will be reset. The blocks Input to Simulink and Output to Simulink are S-functions written in C that enable the external modules to communicate with the Simulink inputs and outputs.

A Matlab program, *Exc_handler* is available to feed in excitation signals to the IRB 2000 robot. This program can be used to define position and velocity references to the built-in controllers as well as direct torque inputs. The excitation signals can be step, ramp, sinusoid, noise (PRBS) or arbitrary signals from Matlab workspace. Outputs from the excitation

Figure 1.3: Force Sensor JR3

like the input torques, position measurements, velocity measurements (which are differentiated position measurements), force and torque measurements from the force sensor are also recorded. These signals can then be exported to Matlab workspace for use.

1.2.3 Force Sensor JR3

The force sensor used, see Figure 1.3, is of type 100M40A manufactured by JR3 Inc. It is a wrist sensor and is a mechanical structure instrumented with strain gauges which can measure forces in the x-, y- and z-directions as well as the corresponding torques. It is mounted on the wrist of the robot between the end-effector and joint six and results in a rotation of 230 degrees about the z-axis of joint six. It is DSP-based and has a sample rate of 8kHz. It can measure forces up to 400N and torques up to 40Nm. To protect the force sensor, it is connected to the end-effector via a pneumatic lock linked to the emergency stop.

1.2.4 ABB IRB 6 Robot

The other robot used in the thesis is ABB Industrial Robot IRB 6 as shown in Figure 1.4. Unlike the IRB 2000 robot, it has only five DOF and thus unable to reach all points in the workspace. Hence, given a desired position, the goal pose of the IRB 6 robot might have to be modified to lie in the robot's subspace. The function modorient.m is written for this purpose using the algorithm by Hedenborn and Olsson [6]. On top of that, the working range of joints are much smaller than that for the IRB 2000 robot. This results in a much smaller workspace.

Joint one and five are cylindrical joints while joint two, three and four are revolute joints. There is a mechanical coupling between joint two

Figure 1.4: The ABB IRB 6 robot.

and three. Joint one turns the robot round the base, joint two moves the lower arm back and forth, joint three moves the upper arm up and down. Joint four turns the wrist unit and joint five bends the wrist unit around its centre.

1.2.5 Robot Setup

The setup of the robots in the Robotics Laboratory is as shown in Figure 1.5. The position of the robots relative to each other in the lab has not been calibrated. Rough measurements show that the base of the IRB 2000 robot is located at $\begin{bmatrix} 1.15 & -2.15 & 0 \end{bmatrix}$m according to the IRB 6 robot's base coordinate system. On top of that, the IRB 2000 robot is rotated $\frac{\pi}{4}$ radians about the z-axis of the same coordinate system.

1.3 Simulation Platform

A Java visualisation program written in the platform Eclipse is available at the department. Java classes for the kinematics of the IRB 2000 robot are already available. Java classes for the kinematics of the IRB 6 robot are written in order to use the program for the IRB 6 robot as well.

The robots are placed relative to each other in the same way as they do in the lab. The view of the camera is changed such that both robots

Figure 1.5: Robot Setup in the Robotics Laboratory

can be seen clearly.

As the IRB 2000 robot object available at the department does not have the sensor and the end-effector, a simple cuboid is added to the flange of the robot to represent the translation of 391mm and rotation of 230 degrees due to the sensor and the end-effector. A base is also added to the IRB 6 robot object.

Chapter 2

The Robot System

2.1 ABB IRB 2000 Robot

In the master and slave configuration which will be applied later in the cooperative task, the IRB 2000 robot is assigned to be the slave robot. This is due to the fact that it is equipped with a force sensor which enables force feedback and that it has a much bigger workspace and has six DOF.

Since the IRB 2000 robot is the slave robot and has to follow the trajectory of the master robot, more has to be known about its kinematics. In addition to the forward and inverse kinematics, the velocity kinematics and force transmission have to be considered.

2.1.1 Forward Kinematics

The position and orientation of the end-effector, a gripper in the case of the IRB 2000 robot, can be calculated using forward kinematics when all joint values are known. Using the standard Denavit-Hartenberg convention by Denavit and Hartenberg [5], Figure 2.2, the orientation of the coordinate systems for the robot are defined as shown in Figure 2.1. The link parameters a_i, α_i, d_i and φ_i obtained using this definition can be seen in Table 2.1.1.

The link length, a_i, is the offset distance between the z_{i-1} and z_i axes along the x_i axis. The link twist, α_i, is the angle from the z_{i-1} to the z_i axis about the x_i axis. The link offset, d_i, is the distance from the origin of frame $i-1$ to the x_i axis along the z_{i-1} axis. The joint angle, φ_i, is the angle between the x_{i-1} and the x_i axes about the z_{i-1} axis. S refers to the sensor frame and TCP to the Tool Center Point of the gripper. θ_i are the absolute joint values while φ_i are the relative joint values.

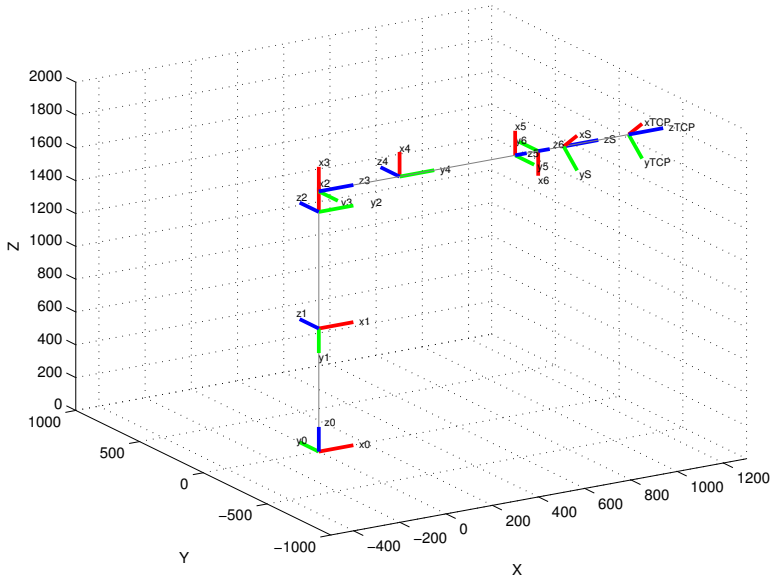


Figure 2.1: Coordinate Systems for IRB2000

link	$a_i[mm]$	$\alpha_i[rad]$	$d_i[mm]$	$\varphi_i[rad]$
1	0	$-\frac{\pi}{2}$	750	θ_1
2	710	0	0	$\theta_2 - \frac{\pi}{2}$
3	125	$-\frac{\pi}{2}$	0	$\theta_3 - \theta_2$
4	0	$\frac{\pi}{2}$	850	θ_4
5	0	$-\frac{\pi}{2}$	0	θ_5
6	0	0	100	$\theta_6 + \pi$
S	0	0	110	$\frac{230}{180}\pi$
TCP	0	0	281	0

Table 2.1: The a_i, α_i, d_i and φ_i parameters for IRB 2000

The transformation matrix ${}^{i-1}_{i}T$ of coordinate system i with respect to coordinate system $i-1$ is obtained by:

$$ {}^{i-1}_{i}T = \text{Rot}_{z,\varphi_i} \cdot \text{Trans}_{z,d_i} \cdot \text{Trans}_{x,a_i} \cdot \text{Rot}_{x,\alpha_i} $$

The resulting transformation matrix can be divided into two parts, the rotation matrix R which represents the orientation of the ith coordinate system relative to that of the $i-1$th coordinate system and the position vector p which represents the position of the origin of the ith coordinate

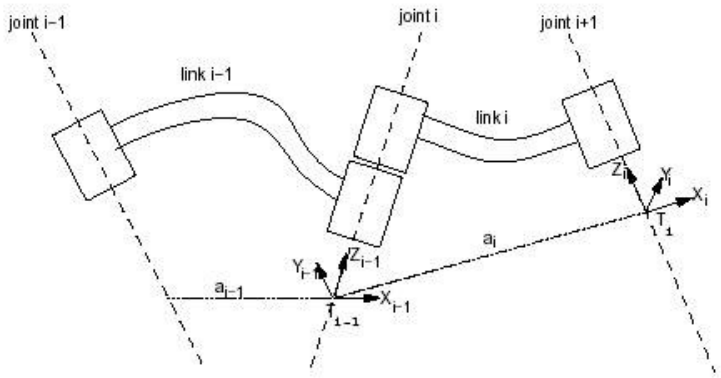


Figure 2.2: The Standard Denavit-Hartenberg Convention

system relative to that of the $(i-1)$th coordinate system.

$$ {}^{i-1}_{i}T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & p_x \\ R_{21} & R_{22} & R_{23} & p_y \\ R_{31} & R_{32} & R_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

To change between the various coordinate systems, the transformation matrices are multiplied. To calculate the forward kinematics, all the transformation matrices are multiplied.

$$ {}^{0}_{TCP}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot {}^{3}_{4}T \cdot {}^{4}_{5}T \cdot {}^{5}_{6}T \cdot {}^{6}_{S}T \cdot {}^{S}_{TCP}T $$

Sometimes, it is more convenient to express the orientation in terms of the unit quaternion since only four values are required as opposed to nine values in a rotation matrix. The unit quaternion is a 4×1 vector which is given by $u = \cos\frac{\theta}{2}$, $v_x = n_x \sin\frac{\theta}{2}$, $v_y = n_y \sin\frac{\theta}{2}$ and $v_z = n_z \sin\frac{\theta}{2}$.

The rotation matrix R can be represented by these four values:

$$ R = \begin{bmatrix} 2u^2 - 1 + 2v_1^2 & 2v_1v_2 - 2uv_3 & 2v_1v_3 + 2uv_2 \\ 2v_1v_2 + 2uv_3 & 2u^2 - 1 + 2v_2^2 & 2v_2v_3 - 2uv_1 \\ 2v_1v_3 - 2uv_2 & 2v_2v_3 + 2uv_1 & 2u^2 - 1 + 2v_3^2 \end{bmatrix} $$

To obtain the unit quaternion from the rotation matrix,

$$ \begin{aligned} u &= 0.5 \cdot \sqrt{R_{11} + R_{22} + R_{33} + 1} \\ v_1 &= 0.5 \cdot \sqrt{R_{11} - R_{22} - R_{33} + 1} \qquad sign(v_1) = sign(R_{32} - R_{23}) \\ v_2 &= 0.5 \cdot \sqrt{R_{22} - R_{11} - R_{33} + 1} \qquad sign(v_2) = sign(R_{13} - R_{31}) \\ v_3 &= 0.5 \cdot \sqrt{R_{33} - R_{11} - R_{22} + 1} \qquad sign(v_3) = sign(-R_{12}) \end{aligned} $$

The function forward2400.m is available at the department to calculate the forward kinematics of the IRB 2000 robot from the base frame to frame six using ABB convention. A simple function for forward kinematics which uses the method described above is used when the forward kinematics from the base frame to the TCP frame is required.

2.1.2 Inverse Kinematics

Inverse kinematics is used to find the values of the joint variables that will place the end-effector at a desired location with a desired orientation relative to the base. The problem of inverse kinematics of the 6-DOF IRB 2000 robot corresponds to solving a set of six nonlinear, transcendental equations with six unknowns (joint variables). There may be no solution, an unique solution or multiple solutions.

As the IRB 2000 robot has a spherical wrist, i.e. the last three joint axes intersect at the same point, WCP (Wrist Center Point), the inverse kinematics problem can be divided into two parts. Firstly, the WCP is calculated given the position and orientation of the TCP. From the wrist position, the first three joint values can be found. Secondly, with the values of the first three joints, the last three joint values can be computed to achieve the given orientation.

Two types of singularities can arise in the computation of inverse kinematics: arm singularity and wrist singularity. Arm singularity occurs when both the x- and y-coordinates of the WCP are close to zero. In this case, joint one can be chosen freely and is set to be zero. Wrist singularity occurs when the rotation of joint five aligns joint four and joint six. This means that joint four and six can be chosen freely as long as the sum of these two joints is zero. Joint four and six are set to zero when a wrist singularity occurs.

The function invkin2400.m is available at the department to calculate the inverse kinematics of the IRB 2000 robot.

2.1.3 Velocity Kinematics

In addition to the kinematic relationship between the joint values and the end-effector's location, the mapping of the joint velocities to the end-effector's linear and angular velocities is also useful. This mapping is specified by the velocity Jacobian of the robot and it changes as the configuration of the robot changes. At singularities, the Jacobian becomes singular and this means that there is some direction in Cartesian

space along which it is impossible to move the hand of the robot no matter which joint rates are selected. There are two types of singularities: workspace boundary singularities which occur when the end-effector is near or at the boundary of the workspace and workspace interior singularities which is the same as the wrist singularity explained in the Section 2.1.2.

The Jacobian can be divided into two parts:

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

where J_v relates the velocity of the joint variables, $\dot{q}$, to the linear velocity, v and J_ω relates the velocity of the joint variables to the angular velocity, ω. This can be expressed as:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J\dot{q}$$

Each column of the Jacobian is obtained from each joint of the robot and the structure of the column depends on the type of joint.

Revolute joints: $J_i = \begin{bmatrix} \vec{z_i} \times (\vec{o_6} - \vec{o_i}) \\ \vec{z_i} \end{bmatrix}$

Prismatic joints: $J_i = \begin{bmatrix} \vec{z_i} \\ \vec{0} \end{bmatrix}$

o_i is the origin of the ith frame in terms of the base frame and z_i is the z-axis of the ith frame in terms of the base frame. Both o_i and z_i can be obtained from the transformation matrices from Section 2.1.1. The velocity jacobian obtained is with respect to relative joint angles. It has to be with respect to absolute joint angles in some applications. This is done by replacing column two by the old column two substracted by column three.

$$^{abs}J = \begin{bmatrix} J_1 & (J_2 - J_3) & J_3 & J_4 & J_5 & J_6 \end{bmatrix}$$

Velocity transmission, i.e. the Cartesian transformation of linear and angular velocities between the different coordinate systems is often necessary in robot control. Given the transformation matrix, ${}^i_jT$, between coordinate systems i and j, where

$${}^i_jT = \begin{bmatrix} {}^i_jR & \vec{p} \\ \vec{0} & 1 \end{bmatrix}$$

the matrix operator which corresponds to taking the cross product with the vector $\vec{p}$ can be obtained

$$ {}^{i}_{j}P\times = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix} $$

The Jacobian for velocity transmission can then be obtained by using this matrix operator and the R matrix according to Craig [4]

$$ {}^{i}_{j}J_{vel} = \begin{bmatrix} {}^{i}_{j}R & 0 \\ {}^{i}_{j}P \times {}^{i}_{j}R & {}^{i}_{j}R \end{bmatrix} $$

2.1.4 Force Jacobians

The force Jacobian is obtained simply by taking the transpose of the velocity jacobian. This Jacobian relates the forces and torques at the end-effector to the joint torques. Similarly, the Jacobian for force transmission between different frames is the transpose of the Jacobian for velocity transmission.

2.2 ABB IRB 6 Robot

The IRB 6 robot is chosen to be the master robot. The forward and inverse kinematics is required for trajectory generation. However, velocity and force Jacobians are not necessary since the force control scheme is not applied on this robot.

2.2.1 Forward Kinematics

The parameters for the IRB 6 robot is given in Table 2.2.1. However, since the IRB 6 robot has only five DOF and has a simpler structure, a more straightforward method of computing the forward kinematics, according to Nilsson [11], will be used. The calculations are based on the definition of the coordinate system according to ABB Robotics [2] which is shown in Figure 2.3.

To get the position of the end-effector, $\vec{v_5}$:

$$ \vec{v_1} = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} $$

$$ \vec{v_2} = D_z(\theta_1) \begin{bmatrix} 0 \\ 0 \\ l_1 \end{bmatrix} $$

link	$a_i[mm]$	$\alpha_i[rad]$	$d_i[mm]$	$\varphi_i[rad]$
1	0	0	0	θ_1
2	0	$\frac{\pi}{2}$	0	θ_2
3	450	0	0	θ_3
4	670	0	0	θ_4
5	0	$-\frac{\pi}{2}$	0	θ_5

Table 2.2: The a_i, α_i, d_i and φ_i parameters for IRB 6

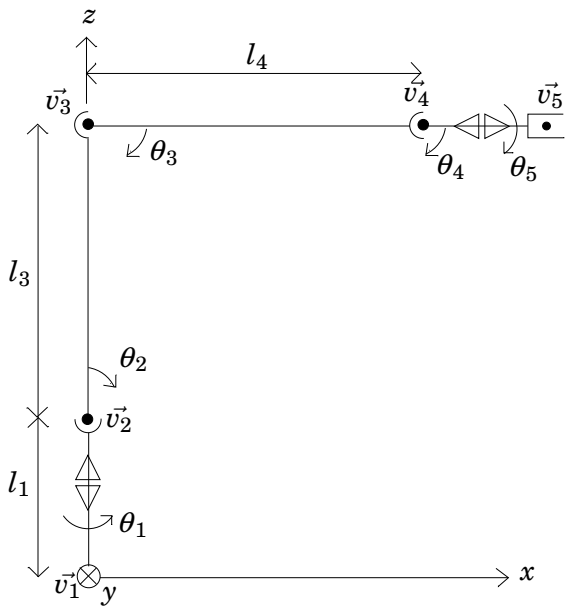


Figure 2.3: Coordinate System for IRB 6

$$\vec{v_3} = \vec{v_2} + D_z(\theta_1)D_y(\theta_2)\begin{bmatrix} 0 \\ 0 \\ l_3 \end{bmatrix}$$
$$\vec{v_4} = \vec{v_3} + D_z(\theta_1)D_y(\theta_3)\begin{bmatrix} l_4 \\ 0 \\ 0 \end{bmatrix}$$
$$\vec{v_5} = \vec{v_4} + D_z(\theta_1)C\begin{bmatrix} f + tool \\ 0 \\ 0 \end{bmatrix}$$

where $b = 157$mm, $l_1 = 700$mm, $l_2 = 450$mm, $l_3 = 670$mm and $f = 95$mm and D_x, D_y and D_z are 3×3 rotation matrix about x-, y- and z-axis respectively. The tool length varies according to the tool that is attached to the flange.

The orientation of the end-effector is given by the rotation matrix

$$R = D_z(\theta_1)D_y(\theta_4)D_x(\theta_5)$$

2.2.2 Inverse Kinematics

According to Nilsson [11], the inverse kinematics of the IRB 6 robot can be calculated as below.

Given the Cartesian position $\vec{v_5} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ of the end-effector, joint one can be obtained by $\theta_1 = \arctan\frac{y}{x}$.

With the value of joint one and the rotation matrix R, matrix C can be computed by $C = D_z(\theta_1)^{-1}R$. From this matrix, joint four and five can be obtained directly by the equations $\theta_4 = \arctan(-\frac{C_{31}}{C_{11}}) + n\pi$ and $\theta_5 = \arctan(-\frac{C_{23}}{C_{22}}) + n\pi$ respectively.

Let $\vec{k} = C\begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$

To find θ_2 and θ_3, v_r which is the length from the base to joint four in the xy plane and z_4 which is the length from the base to joint four in z-direction has to be computed.

$$v_r = \sqrt{x_4^2 + y_4^2} = \sqrt{(x + \sin\theta_1 k_2 - \cos\theta_1 k_1)^2 + (y - \sin\theta_1 k_1 - \cos\theta_1 k_2)^2}$$

$$z_4 = z - k_3$$

The length from joint two to joint four in the xy-plane, v_r', is equal to v_r while the length from joint two to joint four in z-direction, z_4', is equal to

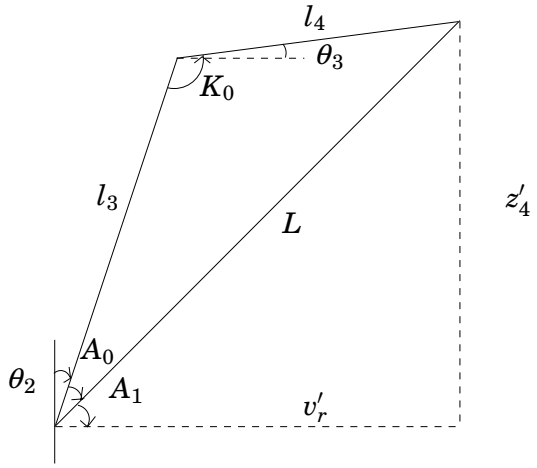


Figure 2.4:

$z_4 - 700$.

According to Figure 2.4,

$$L = \sqrt{(v_r')^2 + (z_4')^2}$$

By cosine theorem,

$$A_0 = \arccos \frac{l_3^2 + L^2 - l_4^2}{2 l_3 L}$$

and by Pythagoras Theorem,

$$A_1 = \arctan \frac{z_4'}{v_r'}$$

With the values of A_1 and A_0, θ_2 can be obtained by

$$\theta_2 = -A_1 - A_0 + \frac{\pi}{2}$$

Using the cosine theorem again,

$$K_0 = \arccos \frac{l_4^2 + l_3^2 - L^2}{2 l_3 l_4}$$

With the value of K_0, θ_3 can be obtained by

$$\theta_3 = -K_0 + \theta_2 + \frac{\pi}{2}$$

2.3 Actuator Space

The joint values obtained from inverse kinematics are in joint space and they have to be converted to actuator space before they can be used. For the IRB 2000 robot, given the gear ratios N_i, the conversion is straightforward except for joint six due to the mechanical coupling between joint five and six.

The conversion is given by $\theta_i = \varphi_i / N_i$ for joint one to five and $\theta_6 = (\varphi_6 + \varphi_5)/N_6$ for joint six. θ_i are the joint values in joint space and φ_i are the joint values in actuator space.

The conversion is more complicated for IRB 6.

$$
\begin{aligned}
\theta_1 &= n_1 \varphi_1 \\
\theta_2 &= a + b - \pi - \arccos \frac{D^2 + E^2 - (\frac{n_2 \varphi_2}{2\pi} + x_{02})^2}{2DE} \\
\theta_3 &= d - \frac{\pi}{2} - \arccos \frac{L^2 + M^2 - (\frac{n_3 \varphi_3}{2\pi} + x_{03})^2}{2LM} \\
\theta_4 &= n_4 \varphi_4 \\
\theta_5 &= n_5 (\varphi_5 + \varphi_4)
\end{aligned}
$$

where n_i are the gear ratios for the IRB 6 robot.

Chapter 3

Trajectory Generation

In order to make a robot move from one position to another, a path has to be generated in order for the motion to be smooth. A path is a geometric representation in either Cartesian space or joint space.

During the motion, the linear velocity and acceleration of the end-effector and the angular velocity and acceleration of the joints of the robot have to stay within certain limits. At the same time, the motion should be completed in the minimum possible time. Thus, a velocity profile of the path is necessary. A path with a velocity profile is called a trajectory.

To generate a trajectory, a path in Cartesian space is first generated and then converted to joint space. Optimization is then done on the path in joint space to get the velocity profile for the trajectory.

3.1 Path Generation in Cartesian Space

To generate a path from arbitrary points in Cartesian space which the robot has to go to, the Cartesian path has to be parameterized. For motion along a straight line from point p_1 to point p_2, the position along the path can be expressed as $p = p_1 + l(p_2 - p_1)$, $0 \leq l \leq 1$.

When considering a three point linear motion, for example moving from point p_1 to point p_3 via point p_2, in order to prevent discontinuities in velocities and acceleration, a zone of radius z_2 is specified such that the robot moves from p_1 to p_2 but starts moving towards p_3 when inside a circle of radius z_2 around p_2, see Figure 3.1.

A number of requirements can be formulated for the behavior of the path inside the zone. A fundamental requirement is that the path is continuous with continuous derivative. Another requirement is that when in-

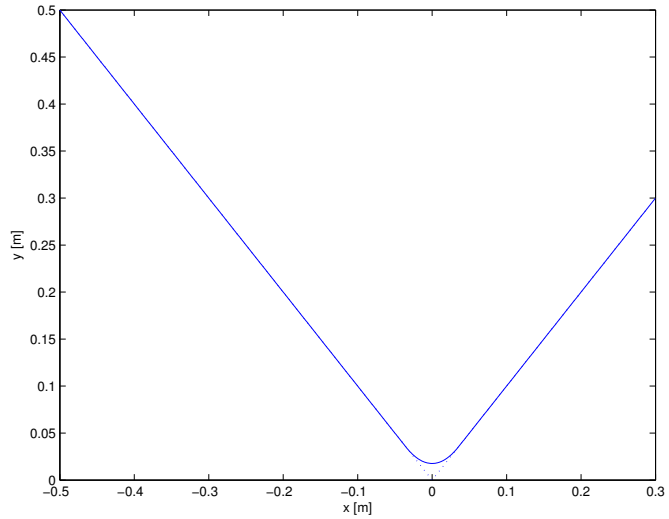


Figure 3.1: Cartesian Path with a zone at point p2.

terconnecting the two linear sections, the path inside the zone must be symmetric.

The path is represented by sections, $P_j(l)$, $l \in [-l_{z1}, l_{z2}]$ and j is the section number. Since different sections can have different lengths, the derivative of $p_{j,x}(l)$ with respect to l is not always a continuous function in the point where we go from section $j-1$ to section j. This is why the sections are grouped according to Figure 3.2.

For the second section, l_{z1} is the value of l where $p_2 + l(p_3 - p_2)$ intersects the zone and $l_{z2} = 1$. Hence the range of l within the zone is $[-l_{z1}, l_{z1}]$ and $[l_{z1}, 1]$ outside the zone.

A second order polynomial is used to represent the path within the zone. A more detailed description of path representation can be found in Norrlöf [13]. Path generation in Cartesian space is done using the Path Generation Toolbox for Matlab by Nyström and Norrlöf [14].

3.2 Path Generation in Joint Space

After the path representation in Cartesian space is obtained, the next step is to transform it to joint space. This is in general not possible to do analytically. Instead, the path must be transformed using the inverse kinematics of the robot at discrete points. Cubic splines are used to ob-

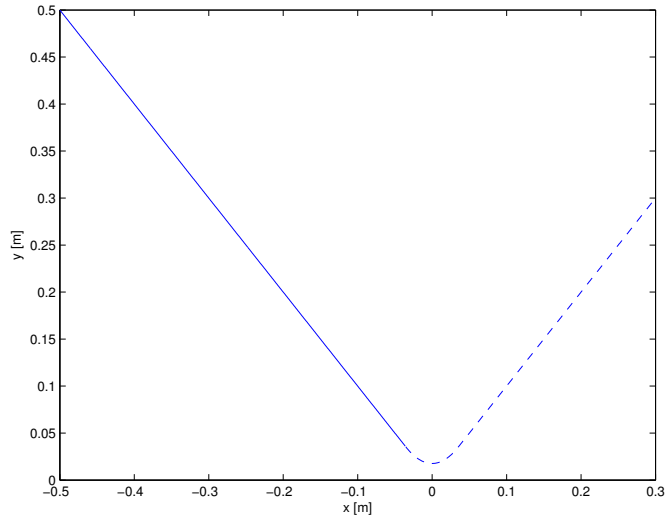


Figure 3.2: Representation of the path in sections. Solid line is section 1, dashed line is section 2.

tain a path in joint space that approximates the true path.

To ensure that the deviation from the true path stays within a given limit, variable step size is used. If the error is within limits, the step size is doubled for the next step to get faster conversion. If not, the step size is halved and the step repeated.

The iterative algorithm for spline interpolation according to Norrlöf [13] is as follow:

1. Compute the inverse kinematic solution at P(l) and $P(l+\delta_l)$, resulting in $\phi(l)$ and $\phi(l+\delta l)$.

2. Find an estimate of $\frac{d\phi(l)}{dl}$ and $\frac{d\phi(l+\delta_l)}{dl}$ using difference approximation.

3. Compute the coefficients in the cubic spline

   $\hat{\phi}(l) = \frac{2(\phi_k-\phi_{k+1})+\delta_k(\phi'_k+\phi'_{k+1})}{\delta_k^3}(l-l_k)^3 - \frac{3(\phi_k-\phi_{k+1})+\delta_k(2\phi'_k+\phi'_{k+1})}{\delta_k^2}(l-l_k)^2 + \phi'_k(l-l_k)$

   where $l_k \leq l \leq l_{k+1}$

4. Evaluate the resulting spline, $\hat{\phi}(l)$, in $l+\delta_l/2$

$$\| T(\hat{\phi}(l+\delta_l/2)) - P(l+\delta_l/2) \| < \gamma$$

where $P(\cdot)$ is assumed to be the path representation in Cartesian space.

a. If the inequality is fulfilled, keep the spline and let

$$l = l + \delta_l, \delta_l = 2\delta_l$$

b. else let

$$\delta_l = \delta_l / 2$$

5. Go to 1.

In cooperative tasks, it is sometimes necessary to have a specific orientation for the tool. However, in the toolbox, the function to convert from Cartesian space to joint space, cart2jsp.m, returns only the 3×1 position vector.

This function is modified to generate values for all six joints for the IRB 2000 robot and five joints for the IRB 6 robot with the possibility to specify the orientation. The orientations of the two robots are chosen depending on the type of task they are performing. However, the orientations have to be fixed throughout the trajectory.

3.3 Optimized Trajectory Generation

After the geometric path in joint space is obtained, the next step is to generate a trajectory which enables the robot to move along the path in minimum amount of time using a limited amount of energy and keeping limits on the maximum linear velocity and acceleration and angular velocity and acceleration, see Åkerblad [22].

This results in an optimization problem.

$$\begin{array}{ll} \max_a & a \\ \text{subject to} & a_{min} \leq a \leq a_{max} \\ & 0 \leq v \leq v_{max} \\ & \dot{\phi}_{min} \leq \dot{\phi} \leq \dot{\phi}_{max} \\ & \ddot{\phi}_{min} \leq \ddot{\phi} \leq \ddot{\phi}_{max} \end{array}$$

The variables expressed in terms of a_i are then substituted into the optimization problem given above and the reformulated optimization problem is obtained.

$$\begin{array}{ll} \max_a & a_i \\ \text{subject to} & a_{min} \leq a \leq a_{max} \\ & 0 \leq v_{i-1}^2 + 2a_i \Delta l_i \leq v_{i,max}^2 \end{array}$$

$$
\begin{aligned}
&\vdots \\
&0 \leq v_{i-1}^2 + 2\sum_{j=i}^{n} a_j \Delta l_j \leq v_{n,max}^2 \\
&0 \leq (\tfrac{d\Phi}{dl_i})^2 (v_{i-1}^2 + 2a_i \Delta l_i) \leq v_{i,max}^2 \\
&\vdots \\
&0 \leq (\tfrac{d\Phi}{dl_i})^2 (v_{i-1}^2 + 2\sum_{j=i}^{n} a_j \Delta l_j) \leq v_{n,max}^2 \\
&\ddot{\phi}_{i,min} \leq \tfrac{d^2\Phi}{dl_i^2}(v_{i-1}^2 + 2a_i \Delta l_i) + \tfrac{d\Phi}{dl_i} a_i \leq \ddot{\phi}_{i,max} \\
&\vdots \\
&\ddot{\phi}_{n,min} \leq \tfrac{d^2\Phi}{dl_i^2}(v_{i-1}^2 + 2\sum_{j=i}^{n} a_j \Delta l_j) + \tfrac{d\Phi}{dl_i} a_i \leq \ddot{\phi}_{n,max}
\end{aligned}
$$

Since the reformulated optimization problem has only acceleration a_i as variable and a_i is linear in both the cost function and the constraints, it can be implemented as a linear program and solved using the linear programming solver in the Optimization Toolbox in Matlab.

However, it is desirable for the linear velocity to go to zero at the end of the trajectory. One solution is to make the linear velocity go to zero linearly at the end by decreasing the maximum velocity linearly for the last five samples. This method results in a slightly slower trajectory.

Another solution is to use a symmetric method. Optimization is done from the first and last samples towards the sample in the middle. This results in a faster and symmetrical trajectory. However, this method requires that the whole path is known beforehand and thus not suitable for online trajectory generation.

For both solutions, the limits for linear velocity, linear acceleration, angular velocity and angular acceleration are fixed for all time samples except for the linear velocity at the end in the first solution.

These two solutions are tested on a simple trajectory and the limits are set to be as below throughout the trajectory. The resultant linear velocity and acceleraton is shown in Figure 3.3 for the first solution and in Figure 3.4 for the second solution.

$$
\begin{aligned}
-2 \leq a \leq 2 \\
0 \leq v \leq 0.25 \\
-4 \leq \dot{\phi} \leq 4 \\
-2 \leq \ddot{\phi} \leq 2 \\
n = 10 \\
\Delta l_c = 0.01
\end{aligned}
$$

The trajectory obtained is then converted to actuator space as described in Section 2.3. To obtain the reference angular velocity for the trajectory, the joint values are differentiated and low-pass filtered.

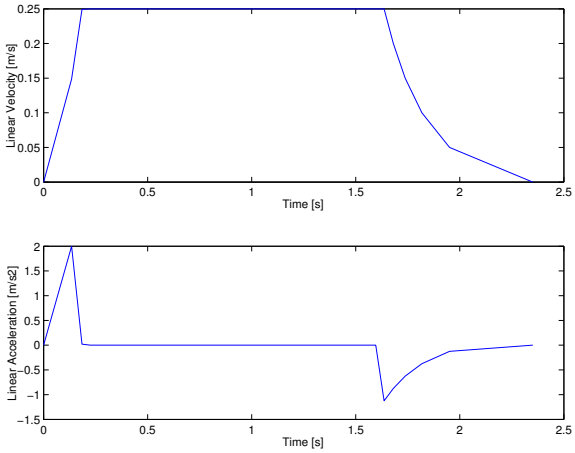


Figure 3.3: The linear velocity and linear acceleration along the test path when the first solution is used.

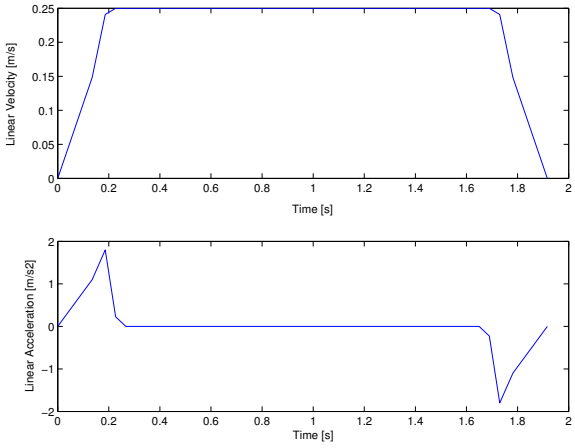


Figure 3.4: The linear velocity and linear acceleration along the test path when the second solution is used.

Chapter 4

Robot Control Schemes

4.1 Position-Velocity Control

In the standard configuration after startup the robot system runs a cascaded position- and velocity- PID (Proportional, Integral and Derivative) controller for each of the six joints, with the velocity loop as the innermost loop and the position control loop as the outermost loop. The controller for a single joint is shown in Figure 4.1. The velocity signal used in the velocity loop is the position signal differentiated and low-pass filtered.

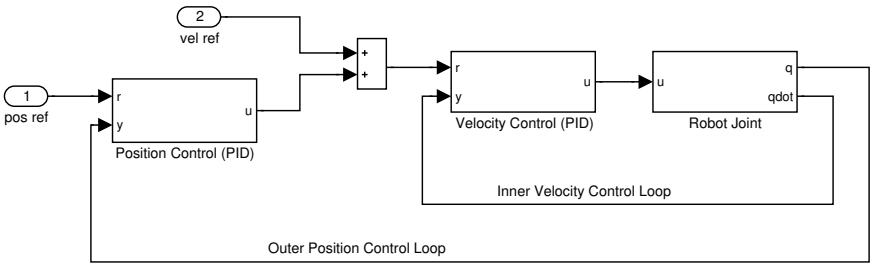


Figure 4.1: The structure of the built in controller for the control of each joint.

4.2 Stiffness Control

When the robot is in contact with the environment, the end-effector's position and orientation are constrainted along certain task-space directions and a suitable compliant behavior of the robot is required to accommodate the interaction. Stiffness control, or compliance control, can be used to achieve this purpose.

Stiffness control can be either passive or active. In passive stiffness control, the end-effector is equipped with a mechanical device composed of springs and dampers. In active stiffness control, PD control is applied on the position error. The smaller the gain used in the controller, the more compliant the robot will be with regard to the environment. The contact forces are treated as load disturbances and no force measurements are required for this scheme.

4.3 Force Control

In the execution of certain tasks, for example polishing a window, not only does the robot have to follow a trajectory, it also has to follow a force trajectory. This can be achieved by combining a force control scheme with a motion control scheme. Different force control schemes that will be discussed in this section are direct force control, impedance control and admittance control. The built-in position/velocity controllers will be used for the motion control part.

4.3.1 Direct Force Control

The first force control scheme that will be discussed here is the direct force control scheme. The force loop consists of a PI-controller acting on the force error between the desired and the actual contact forces and torques as shown in Figure 4.2.

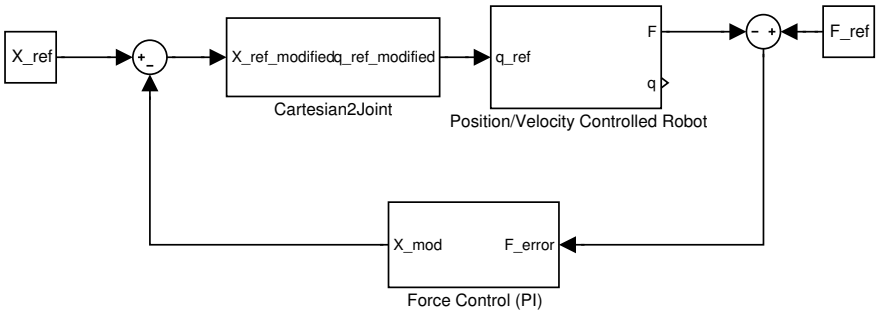


Figure 4.2: The structure of the position-based direct force controller according to Siciliano and Villani [18].

The controller will then return X_{mod} which is the modification required for the trajectory in Cartesian space. The modified trajectory is then obtained by $X_{ref} - X_{mod}$ and is converted to joint space and fed to the position-controlled robot.

The parameters for the PI-control should be chosen such that the force loop dominates over the position loop so that the force error goes to zero at the expense of the position error.

4.3.2 Impedance Control

The aim of impedance control is to achieve a dynamic relationship between the motion of the end-effector and the contact forces and torques, i.e. the mechanical impedance rather than attempting to control either of these variables alone.

This can be represented as:

$$F = K(X_{ref} - X) + D \cdot \frac{d}{dt}(X_{ref} - X) + M \cdot \frac{d^2}{dt^2}(X_{ref} - X)$$

where F is the forces and torques at the robot's end-effector, X and X_{ref} are the actual and reference Cartesian positions and orientations, respectively. The constant matrices K, D and E represent stiffness, damping and inertia, respectively.

The structure of the position-based impedance controller according to Zeng and Hemami [21] is shown in Figure 4.3. Instead of having the position error as input, contact forces and torques are used as input. The transfer function used is thus inverse impedance.

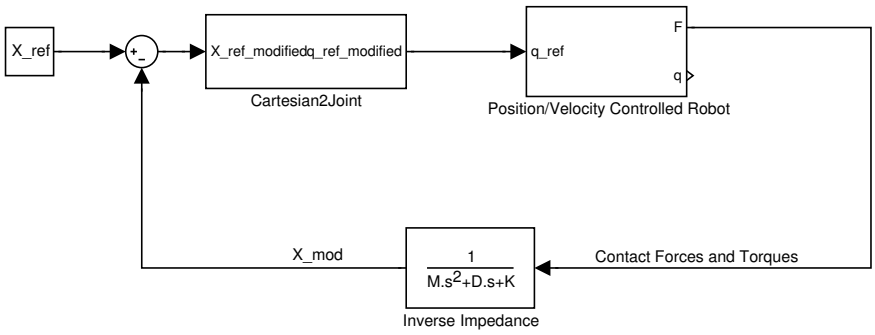


Figure 4.3: The structure of the position-based impedance controller according to Zeng and Hemami [21].

The desired impedance behavior is a trade-off between the trajectory error and force error. If the motion of the robot is not constrained, i.e. there is no contact force or torque, the robot will move to the reference position.

The main drawback of impedance control is that the force is controlled indirectly by changing the values of the impedance parameters which makes it hard to follow a force trajectory.

4.3.3 Admittance Control

Mechanical admittance is defined as

$$Y = \frac{v_f}{F}$$

where v_f is the end-effector velocity and F is the contact forces, both at the point of interaction. A large admittance corresponds to a rapid motion induced by applied forces; while a small admittance represents a slow reaction to contact forces.

Taking in account position and acceleration perturbations as well and converting to the frequency domain the mechanical admittance can also be expressed as:

$$\frac{X(s)}{F(s)} = \frac{Y_m(s)}{s} = (K + Ds + Ms^2)^{-1}$$

where the constant matrices K, D and M represent stiffness, damping and inertia as in Section 4.3.2.

The underlying concept of compliant motion control using admittance is to take a position-controlled robot as a baseline system and to make the necessary modifications of the admittance to this system in order to enable the execution of constrained tasks, according to Seraji [17]. The structure of the position-based admittance controller can be seen in Figure 4.4.

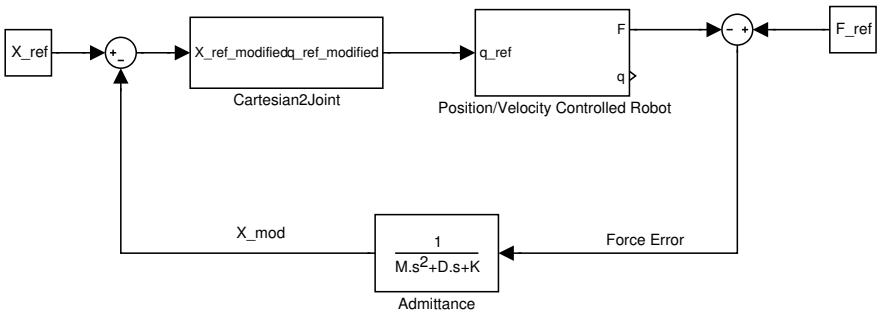


Figure 4.4: The structure of the position-based admittance controller, according to Seraji [17].

This scheme is very similar to the position-based impedance controller discussed in Section 4.3.2. The main difference is that the force error is used as input instead of contact forces. This means that this scheme focuses more on the force error.

However, the main drawback of this control scheme is that the response can either be sluggish or even become unstable if the admittance is not correctly matched.

Chapter 5

Robot Simulations

Before running the actual experiments, simulations are done in Matlab Simulink. In order to do simulations, there are several things which have to be done first. These include:

- The two robots have to be simulated. As both robots are PID position-controlled internally, the IRB 2000 robot is modelled as six decoupled first order systems and the IRB 6 robot is modelled as five decoupled first order systems.
- The Robot Force Control Blockset which is available at the department contains useful Simulink blocks. These blocks have to be modified and verified before they can be used. The blocks available are shown in Appendix B.
- The reference trajectories for the IRB 6 robot and the IRB 2000 robot have to be generated, Section 5.1.
- Gravity calibration has to be done to find the sensor offsets, the Center of Gravity (COG) and weight of the end-effector of the IRB 2000 robot, Section 5.2. With these values, the gravity compensation block from the Robot Control Blockset can then be used in the control scheme.
- A gravity model has to be made to compute the resultant gravitational forces in the sensor frame when the robot is in different configurations, Section 5.3.
- A contact model has to be made to simulate the contact forces and torques that arise from the cooperative task, Section 5.4.
- Simulated sensor readings can then be generated from the contact forces, the gravitational forces, the sensor offsets and sensor noise.

5.1 Trajectory Planning

The position references for the IRB 6 robot is generated using the method described in Chapter 3. The path for the IRB 6 robot is shown in Figure 5.1.

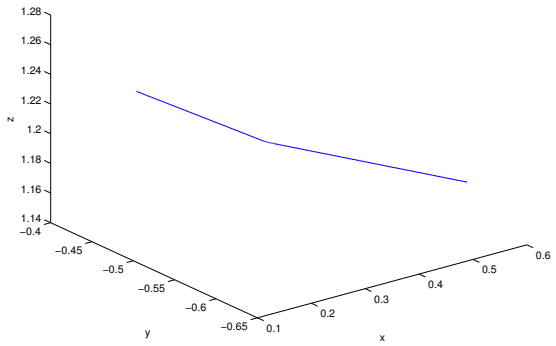


Figure 5.1: The Cartesian path of the end-effector of the IRB 6 robot.

To get better performance, position and velocity references of the IRB 2000 robot are also fed into the force control system. However, it is not possible to get these references directly as in the case of the IRB 6 robot as the position references for the IRB 2000 robot will depend on the IRB 6 robot and also on the beam in between the end-effectors of the two robots.

Instead, the Cartesian position and orientation of the end-effector of the IRB 2000 robot are computed assuming that there is a beam of length 1.0m between the two end-effectors and the orientation of the end-effector of the IRB 2000 robot relative to that of the IRB 6 robot is set to be

$$R = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

This is done using a Simulink model as shown in Figure C.1. The trajectory for the IRB 2000 robot is then checked using the Java visualization program in Eclipse and then tested on the actual robot. It turns out that the limits on the joints are smaller than what are used to generate the trajectory. This is due to the mechanical stops placed on the robot.

The second section of the original trajectory for the IRB 6 robot is then

removed and the trajectory for the IRB 2000 robot regenerated using the same model except for the inclusion of a Matlab function, check-joint, which ensures that the joint values stay within the mechanical limits. The new trajectory is then checked in Eclipse and then on the actual robot. The new position references for both robots are shown in Figure 5.2.

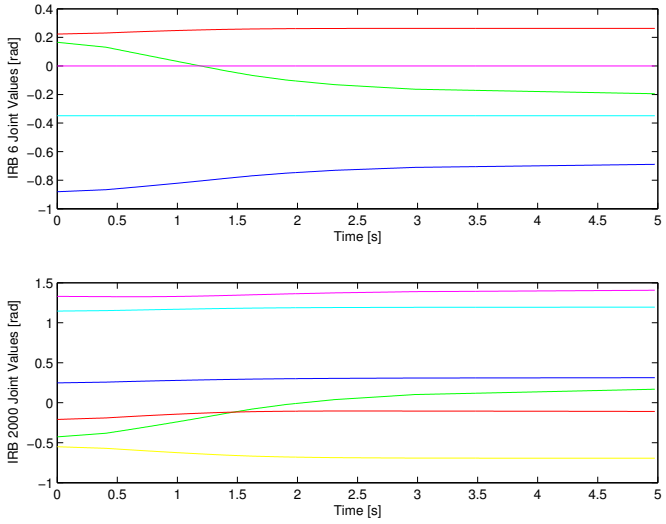


Figure 5.2: New reference joint values for the IRB 6 robot above and for the IRB 2000 robot below.

5.2 Gravity Calibration

Gravity calibration on the IRB 2000 robot has to be done in order to compensate for the effect of gravitational forces due to the weight of the end-effector and also to include the effect of gravitational forces in the simulated force measurements.

To do calibration, the force sensor is first reset when the robot is in home position, that is, when all the joint values are zero. Joint five is then set to vary from -90 degrees to 90 degrees in a sinusoidal waveform and measurements from the force sensor are recorded using the *Exc_handler*.

Joint six is then set to -50 degrees such that the x-axis of the end-effector is pointing vertically upwards when joint one to joint five are zero and the experiment is repeated. Lastly, joint six is set to -140 degrees such

that the y-axis is now pointing vertically upwards and the experiment is repeated.

The force sensor returns force measurements F_x, F_y and F_z which are the forces along x-, y-, and z-axis, respectively and torque measurements M_x, M_y and M_z which are the torques about x-, y- and z-axis respectively.

The values of the sensor offsets can be obtained by taking the mean values of the sensor measurements where the expected force or torque is zero assuming that the weight of the end-effector is the only force acting on the sensor. The offsets obtained in N for forces and Nmm for torques are $\begin{bmatrix} 39.3229 & -39.6195 & 0 & 6318.82 & 4388.85 & -443.42 \end{bmatrix}$.

The weight of the end-effector can be obtained directly from the F_z measurements where joint five is about 90 degrees or -90 degrees since the gravitational force is then acting only along z-axis. The value obtained is 54.58N. Using $g = 9.81$, the mass is found to be 5.564kg.

The COG of the end-effector is found by taking the torque measurements which are non-zero after taking into account the offsets and dividing this compensated values by the weight of the end-effector. The COG is found to be $\begin{bmatrix} 10.8151 & 0 & 136.3124 \end{bmatrix}$ in mm in sensor frame.

These values are then doubled checked by passing the force and torque measurements from the experiments through the gravity compensation block. The resulting forces and torques should be zero if the compensation is perfect. However, an error of 12% is obtained. This can be due to the effect of the weight of the wires hanging from the end-effector to the middle of link five of the robot.

5.3 Gravity Modelling

The effect of the weight of the end-effector has to be taken into account in the simulation of sensor readings. To do that, a gravity model, as shown in Figure C.2, is made using values from the gravity calibration.

The orientation of the coordinate frame of COG is defined to be the same as that of the base coordinate frame. The position of the COG in the base coordinate frame is found by multiplying the transformation matrices of the various frames. The transformation matrix from the sensor frame to the COG frame is computed and and the weight acting at the COG can then be converted to the sensor frame by force transmission.

To verify that the gravity model is correct, sensor offsets are added to the generated values for the gravitational forces at the sensor frame and compared to the actual readings from the sensor. Figure 5.3 shows the simulated forces and torques and the actual sensor readings.

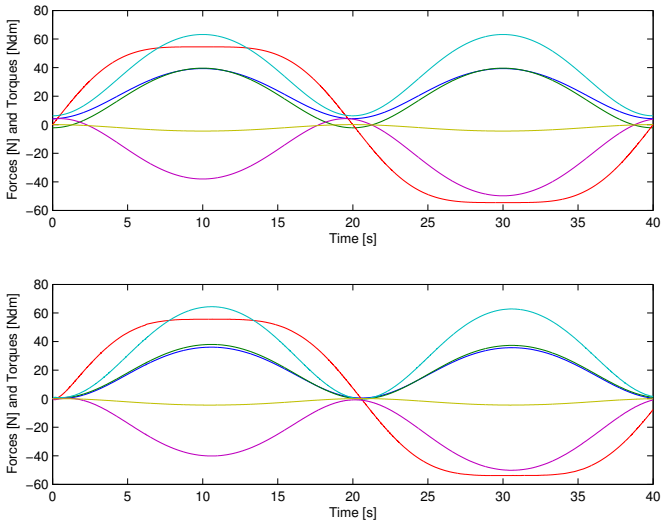


Figure 5.3: The simulated gravitational forces and torques above and the actual sensor readings (filtered) below

5.4 Contact Model

The contact model is implemented in Matlab Simulink as shown in Figure C.3 to simulate the contact forces and torques that can arise from the cooperative task.

The position and orientation of the end-effector of the IRB 6 robot is expressed in terms of the coordinate frame of the end-effector of the IRB 2000 robot. To do that, the transformation matrices of the various frames are multiplied together.

The beam is considered to have the same orientation as the end-effector of the IRB 6 robot and extends along the x-axis of the coordinate frame of the end-effector of the IRB 6 robot. Thus, the transformation matrix from the end-effector of the IRB 2000 robot to the end of the beam at the IRB 2000 robot can be found.

The contact forces and torques are modelled as follows:

$$\begin{aligned} F_x &= K_x \cdot \Delta x + D_x \cdot \Delta \dot{x} \\ F_y &= K_y \cdot \Delta y + D_y \cdot \Delta \dot{y} \\ F_z &= K_z \cdot \Delta z + D_z \cdot \Delta \dot{z} \\ M_x &= K_\gamma \cdot \Delta \gamma \\ M_y &= K_\beta \cdot \Delta \beta \\ M_z &= K_\alpha \cdot \Delta \alpha \end{aligned}$$

The spring and damping constants depend on the flexible beam. For the simulations, these values are set to be:

$$\begin{aligned} K_x, K_y, K_z &= 4\,\mathrm{N/mm} \\ D_x, D_y, D_z &= 0.1\,\mathrm{Ns/mm} \\ K_\gamma, K_\beta, K_\alpha &= 10^6\,\mathrm{Nmm/rad} \end{aligned}$$

These contact forces and torques are then converted to be in the sensor frame by force transmission.

5.5 Controllers

The various force control schemes discussed in Section 4.3 are implemented in Matlab Simulink. The models can be found in the Appendix C. The activation of the force control scheme is done via a set of six parameters which represent the x-, y- and z-translation and the α-, β- and γ-rotation of the TCP.

If force control is not activated, the position and the velocity references will not change. This corresponds to not having external control on the robot. Position and velocity control is done only by the built-in controllers of the robot which are discussed in Section 4.1.

The sensor readings are first gravity-compensated and then transformed to the contact frame to get the contact forces and torques. These values are then used as input to the impedance controller. In the case of the direct force and admittance controller, the force error which is the force reference minus the contact forces and torques is used as input.

For the direct force controller, the parameters for the PI-controller are as follows:

$$\begin{aligned} K_c &= \begin{bmatrix} 0.5 & 0.5 & 0.5 & 10^{-5} & 10^{-5} & 10^{-5} \end{bmatrix} \\ T_i &= \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.5 & 0.5 & 0.5 \end{bmatrix} \end{aligned}$$

The tracking time for the antiwindup part is set to be the same as T_i.

For impedance control, the inertia matrix M is set to be null matrix while the damping matrix D and the stiffness matrix K are set to be diagonal with the diagonal elements as follows:

$$\begin{aligned} D_{diag} &= \begin{bmatrix} 0.01 & 0.01 & 0.01 & 5 \cdot 10^2 & 5 \cdot 10^2 & 5 \cdot 10^2 \end{bmatrix} \\ K_{diag} &= \begin{bmatrix} 0.7 & 0.7 & 0.7 & 10^5 & 10^5 & 10^5 \end{bmatrix} \end{aligned}$$

The stiffness parameters are chosen such that the stiffness of the system is lower than that of the flexible beam so that the end-effector of the IRB 2000 robot will comply to the movement of the flexible beam. The damping parameters are chosen such that the system will be well-damped and yet not too slow.

For admittance control, both M and K are set to be null matrix while D is set to be diagonal with the diagonal elements as follows:

$$D_{diag} = \begin{bmatrix} 0.7 & 0.7 & 0.7 & 5 \cdot 10^4 & 5 \cdot 10^4 & 5 \cdot 10^4 \end{bmatrix}$$

The effect of the stiffness, damping and inertia parameters are different from the previous case since the input is now the force error and not the contact forces and torques. It is sufficient to have only the damping term. The smaller the damping term is, the greater the effect of external forces on the end-effector is. The choice of the damping terms is a compromise between stability and performance.

The output of the force controllers are then the necessary modifications to the Cartesian position and orientation which is converted to joint space using the model as shown in Figure C.8 and inverse kinematics. The position and velocity references are then modified accordingly and the new references are fed to the position-controlled robot.

5.6 Simulation Results

A java plotter block is included in the Simulink model such that the joint values from the Matlab simulations are sent to Eclipse as soon as they are generated. The start position of the simulation is shown in Figure 5.4.

First, simulation is run with force reference $F_{ref} = \vec{0}$ using pure position control, direct force control, impedance control and admittance control. The results of the simulations can be seen in Figure 5.5 to 5.8.

Figure 5.4: The start position of the robots for simulation.

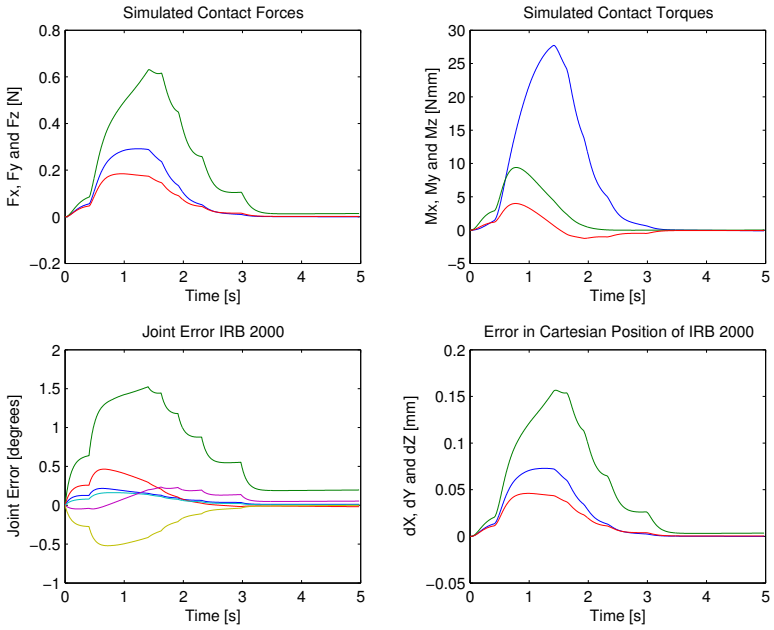


Figure 5.5: Simulation results for position control and $F_{ref} = \vec{0}$.

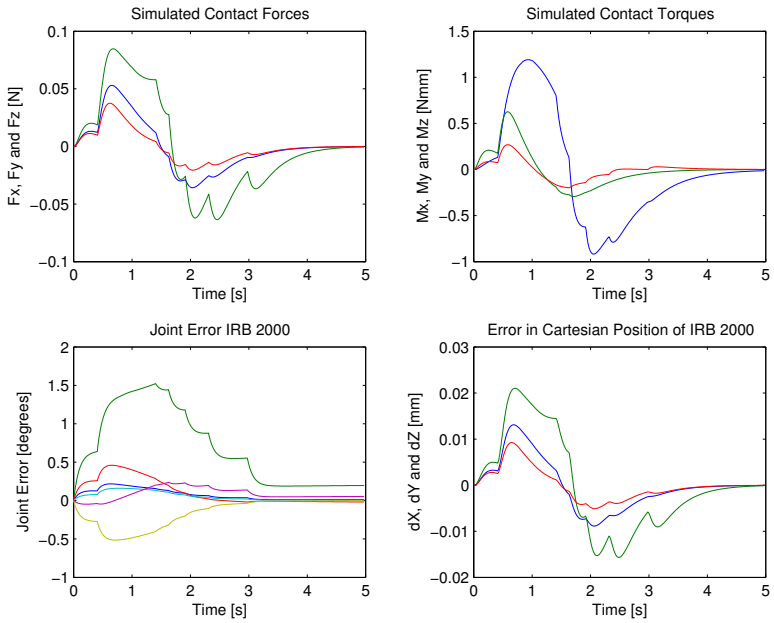


Figure 5.6: Simulation results for direct force control and $\vec{F}_{ref} = \vec{0}$.

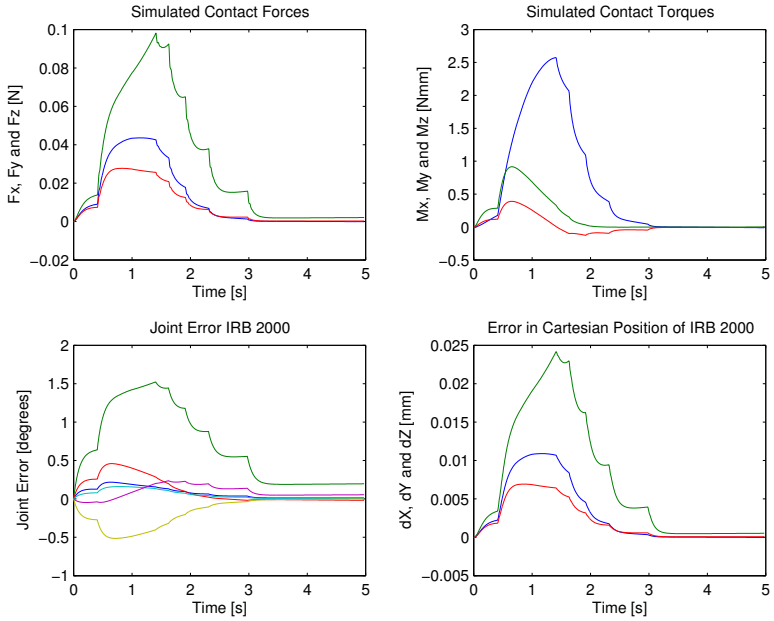


Figure 5.7: Simulation results for impedance control and $\vec{F}_{ref} = \vec{0}$.

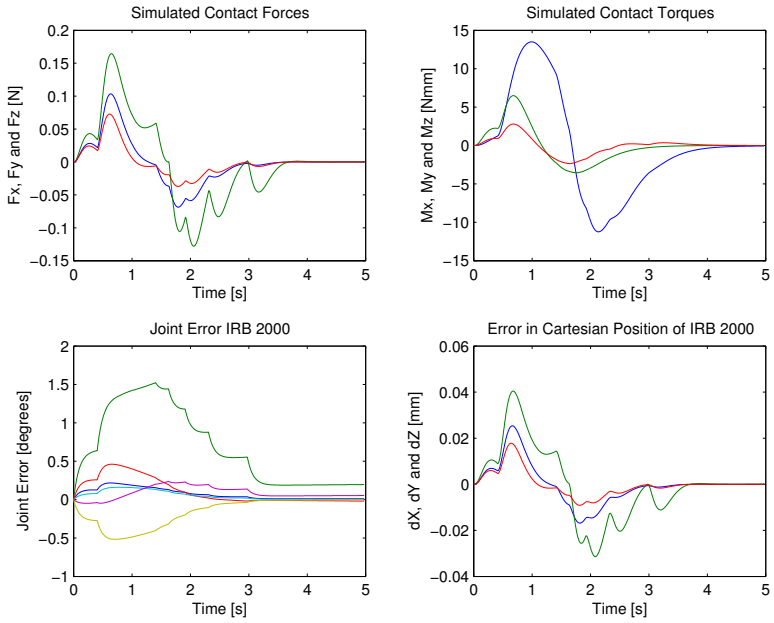


Figure 5.8: Simulation results for admittance control and $F_{ref} = \vec{0}$.

The contact forces and torques for pure position control and impedance control display characteristics of a first order system while that for direct force control and admittance control display characteristics of a second order system. The simulation of the robot joints by first order systems probably causes the first order behavior while the second order behavior is due to the integrator in the force control schemes.

All the force control schemes show an improvement over pure position control. Among the force control schemes, direct force control is most able to keep the forces and torques near to zero while admittance control is least able to do so. A bigger admittance will improve the steady state performance greatly but it will also cause oscillations in the transient.

To investigate how well these schemes can handle disturbances, a force of 10N is added to the contact force in the z-axis at 2.5 s for a duration of 0.05s. The same force reference is used. The results of the simulation can be seen in Figures 5.9 to 5.12.

The presence of the disturbance does not change the results for pure position control since there is no feedback of forces and torques. However, this can result in damages to the end-effector if the robot is made very stiff with respect to the environment.

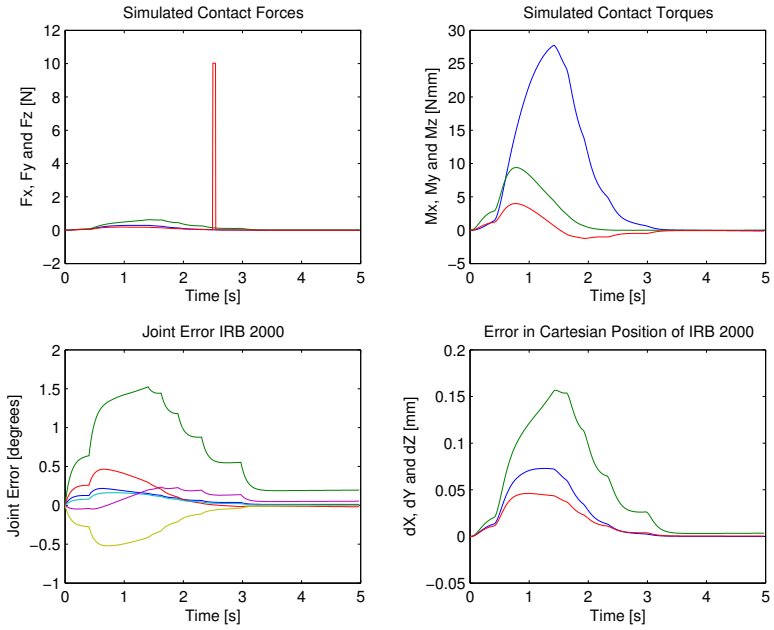


Figure 5.9: Simulation results for position control, $F_{ref} = \vec{0}$ and with disturbance.

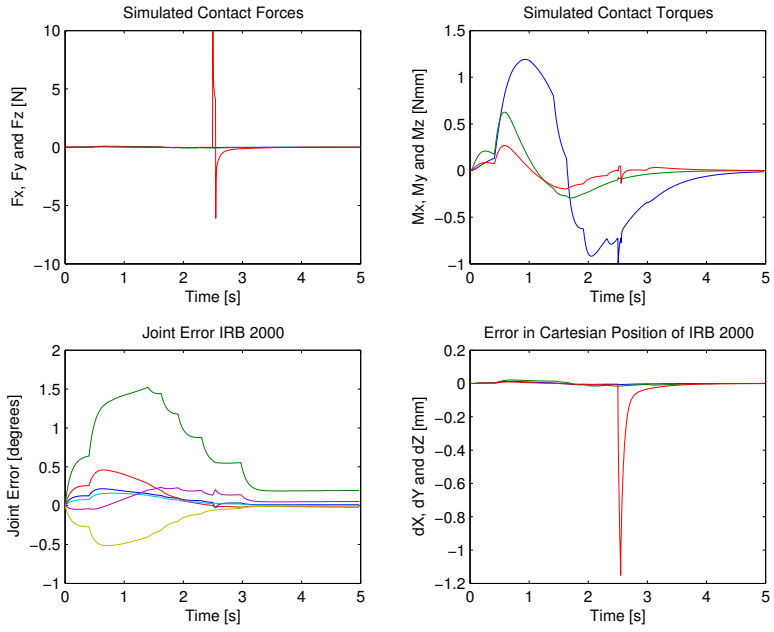


Figure 5.10: Simulation results for direct force control, $F_{ref} = \vec{0}$ and with disturbance.

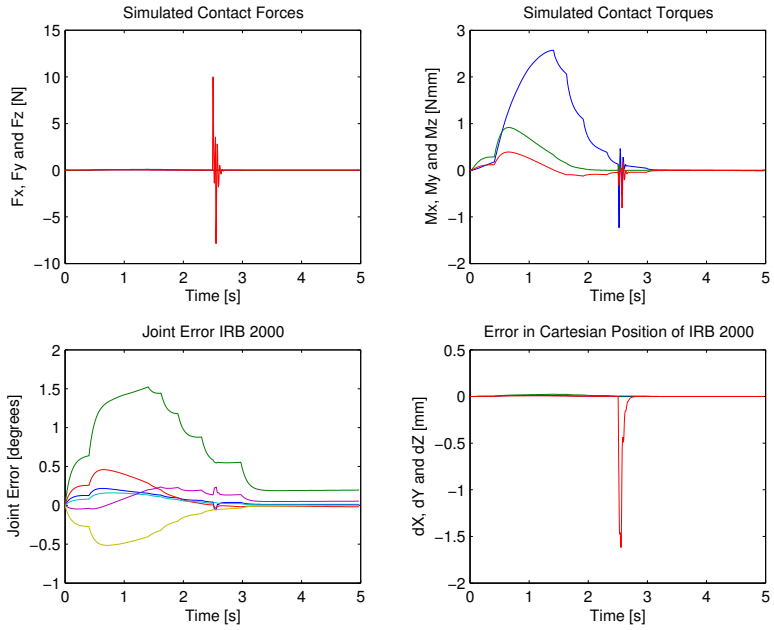


Figure 5.11: Simulation results for impedance control, $F_{ref} = \vec{0}$ and with disturbance.

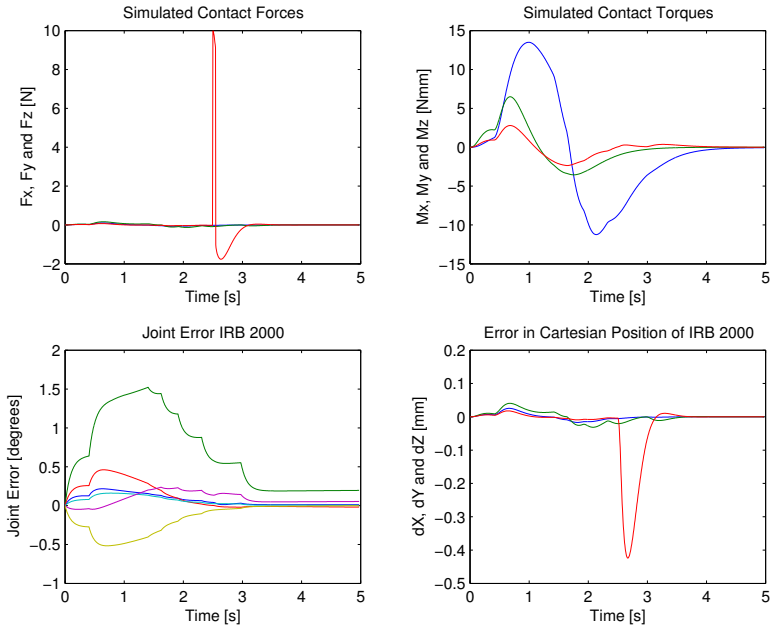


Figure 5.12: Simulation results for admittance control, $F_{ref} = \vec{0}$ and with disturbance.

As for the force controllers, the sudden increase in F_z results in a bigger error in the Cartesian position of the end-effector but the additional error is quickly removed by the controllers. The admittance controller is able to keep the additional error smallest although it is also the slowest to remove it. The impedance controller gives the largest error and oscillations also appear due to the disturbance.

The force reference is then changed to be $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$ N for the forces and $\begin{pmatrix} 200 & 100 & 50 \end{pmatrix}$ Nmm for the torques. Simulations are run with and without disturbances to check the ability of the direct force controller and admittance controller to follow a force reference and handle disturbances at the same time.

The results of the simulations without disturbances are shown in Figures 5.13 and Figure 5.14. Both control schemes are able to track the force reference very well. The admittance controller reacts slower to the force reference. A bigger admittance will decrease response time but it also results in oscillations in the transient.

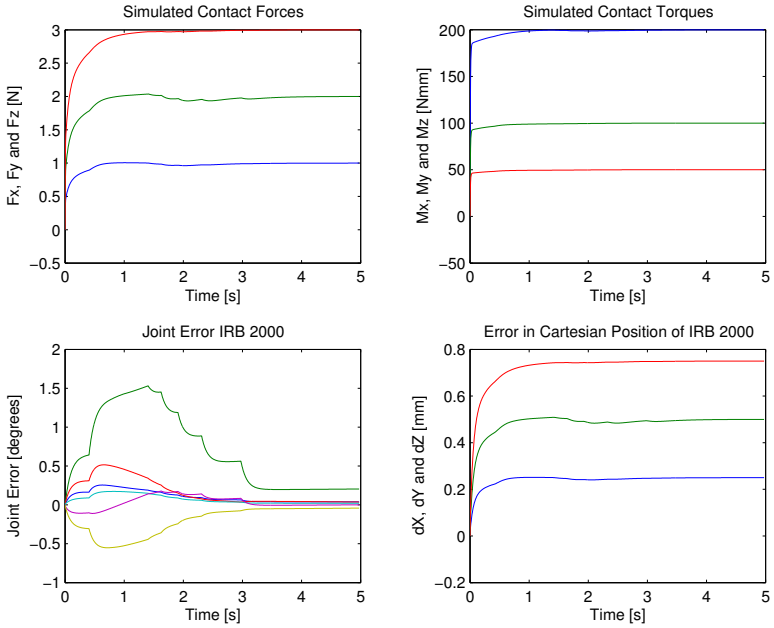


Figure 5.13: Simulation results for direct force control with non-zero F_{ref}.

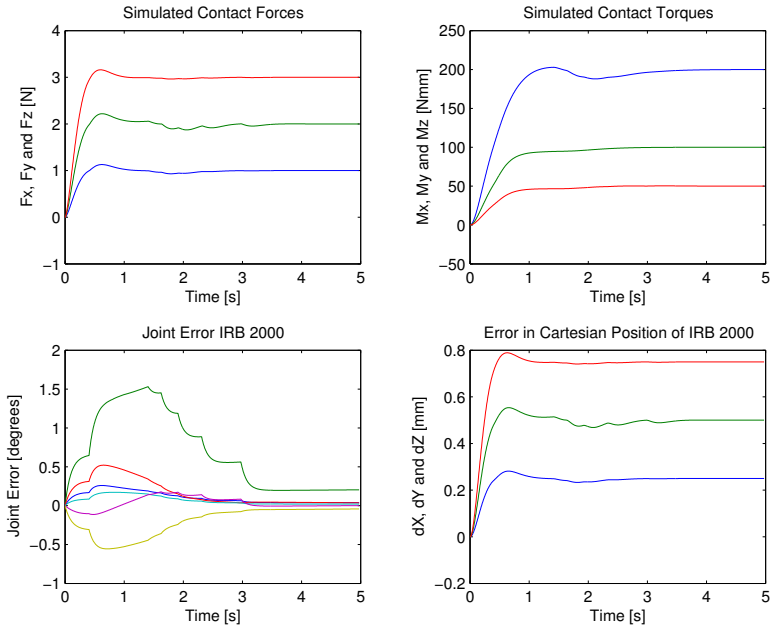


Figure 5.14: Simulation results for admittance control with non-zero F_{ref}.

The results of the simulations with disturbances are shown in Figures 5.15 and Figure 5.16. With the presence of disturbances, the performance of the controllers are not affected. They are able to remove the effect of the disturbance as in earlier simulations.

Simulations are then run for the direct force and admittance controller with a varying force reference. The results of the simulations are shown in Figures 5.17 and Figure 5.18. The direct force controller responds quickly to the change in force reference while the admittance controller is a little slower.

From the results of the simulations, direct force control seems to be the best for force tracking and has the fastest response. Admittance control is the most robust to disturbances. The response of impedance control has no second order behavior.

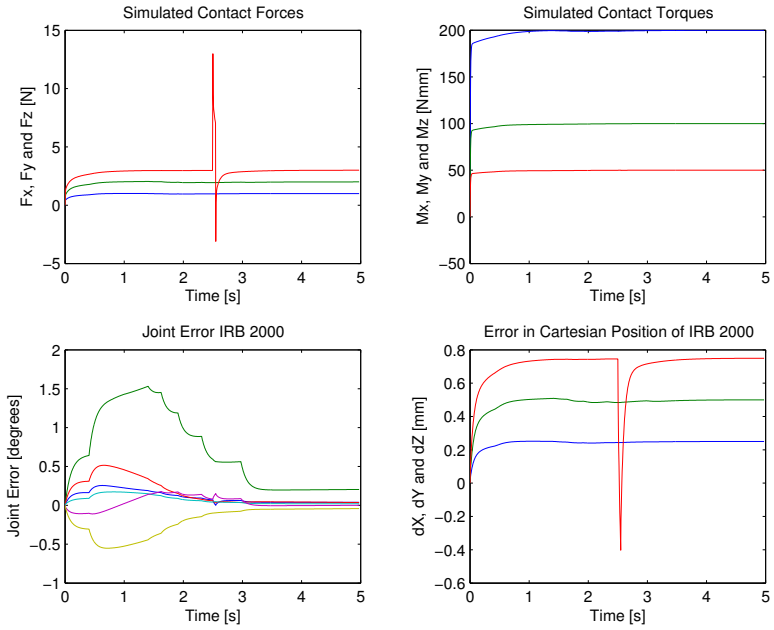


Figure 5.15: Simulation results for direct force control with non-zero F_{ref} and disturbance.

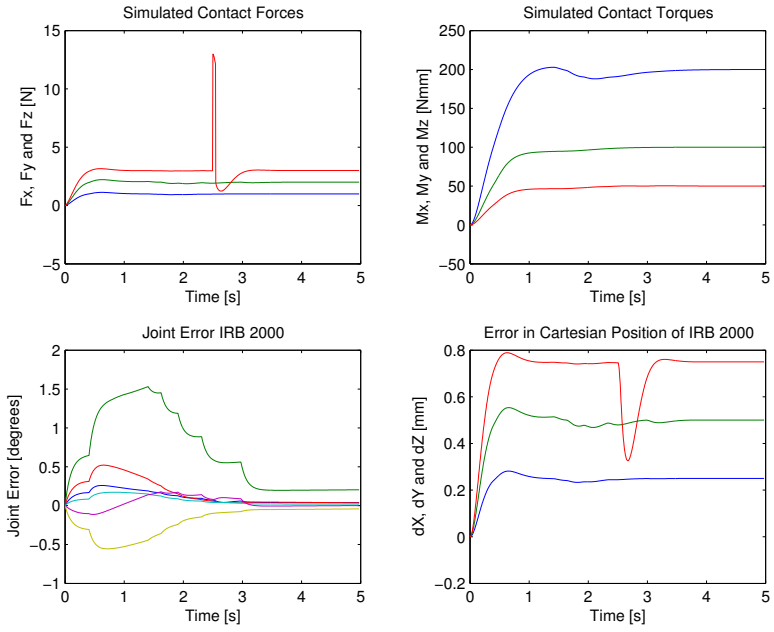


Figure 5.16: Simulation results for admittance control with non-zero F_{ref} and with disturbance.

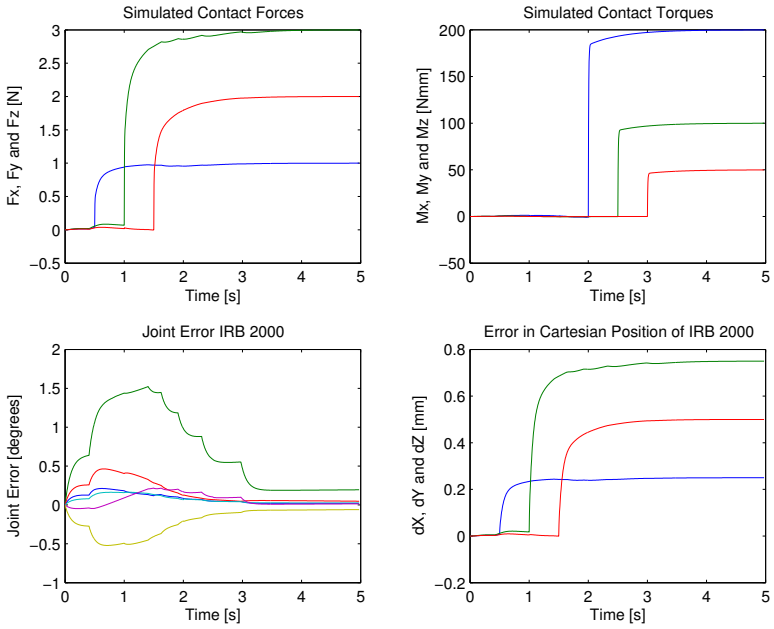


Figure 5.17: Simulation results for direct force control with varying F_{ref}.

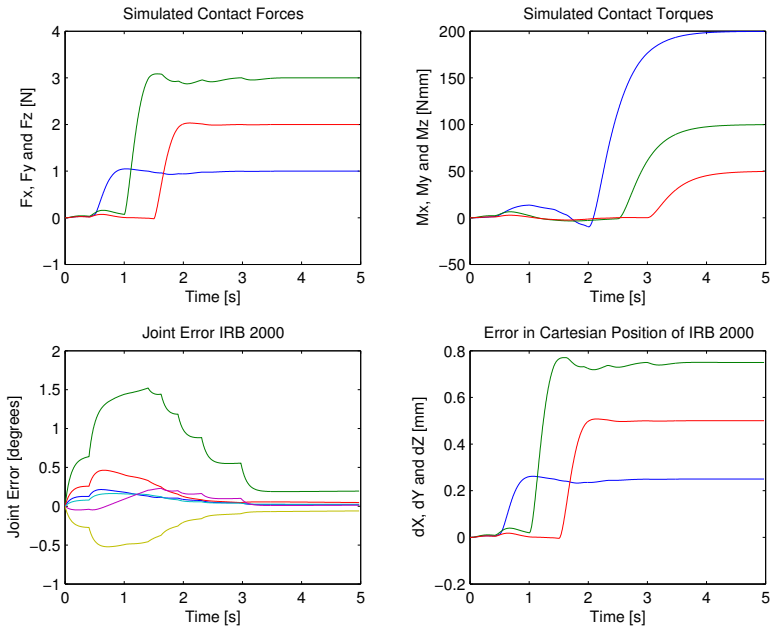


Figure 5.18: Simulation results for admittance control with varying F_{ref}.

Chapter 6

Robot Experiments

6.1 Problems

Some practical problems are encountered during the thesis work:

- The gripper cannot be activated due to some hardware problems. When the grippern is activated, it applies a suction force on the object the robot is gripping by compressed air such that the object is held rigidly.
- A new Java-based RTAI-Linux platform for the basic level motor control of the IRB 6 robot is developed during the Fall Semester 2004. Due to unforeseen circumstances, the completion of the platform is delayed. Thus, the IRB 6 robot cannot be run for the experiments.
- Platforms and compilers are changed at the department at the end of 2004 and this results in some problems with the compilation and download of the arm-control models to the robot system.

Due to the fact that the IRB 6 robot cannot be run, experiments cannot be done to verify the results of the simulation in Section 5.6.

6.2 Experiments

Although experiments cannot be run at the moment, the Simulink model for running the experiments is prepared for further use. The controller is first put into the arm control template in Figure 1.2.

The continuous time integrator in the direct force and admittance controller is replaced by a discrete time integrator with reset, see Figure C.9. This is to enable the reset of the integrator at the start of the experiment.

The force measurements are converted to be right-hand oriented and the values for torques are converted to be in Nmm instead of Ndm. The modified force measurements are then filtered to remove noise. A dead-zone is added to the force measurements to prevent drift. The resulting model is shown in Figure C.10.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The aim of the thesis has been to control a pair of robots in a cooperative task using a master and slave configuration. A parallel force/motion control scheme is used. The force controllers used are direct force, impedance and admittance controllers while motion control is done by the built-in controllers of the robot.

From the simulations, it is found that the direct force controller tracks the force reference best and gives the quickest response. The admittance controller is the most robust to external disturbances. The impedance controller is the simplest but it cannot be used to track a specific force reference.

7.2 Future Work

One obvious area of future work is to run robot experiments to verify the results of the simulations. The contact model in the simulations does not take into account friction and other non-linearities. Modifications might be necessary for the controllers to work on the real robots.

Another possible area of future work is the improvement of the optimal trajectory generation. The Matlab function for conversion of the path from Cartesian space to joint space can be generalized such that it can be used for different robot models as long as the inverse and kinematics of the robot are defined.

Currently, the orientation of the robot's end-effector can be set but it

has to be fixed throughout the trajectory. Modification can be made such that orientation can be changed during the trajectory as well.

Limits in linear and angular velocity and linear and angular acceleration are fixed throughout the optimization. The implementation of the Linear Programming solution can be improved such that the limits can be changed throughout the trajectory.

Bibliography

[1] *Product Manual IRB 2000* ABB Robotics, 1991

[2] *Product Manual IRB 6* ABB Robotics, 1983

[3] S. Chiaverini, B. Siciliano and L. Villani *A Survey of Robot Interaction Control Schemes with Experimental Comparison* IEEE/ASME Transactions on Mechatronics, Vol. 4, No. 3, Sept 1999

[4] J. J. Craig *Introduction to Robotics, Mechanics & Control* Addison Wesley, 1986

[5] J. Denavit, R.S. Hartenberg *A Kinematic Notation for Lower Pair Mechanisms Based On Matrices* Journal of Applied Mechanics, pp. 215-221, 1955

[6] P. Hedenborn and M. Olsson *Simulation Based Improvements of a Service Robot for Disabled People* In Proceedings of Robotikdagar. pp. 117-126, Linköping, Sweden, 1997

[7] D. Henriksson *Observer-based Impedance Control in Robotics* Master Thesis, Department of Automatic Control, Lund Institute of Technology, Nov 2000

[8] S. Liljenborg, A. Olsson *Identify a Surface with Robot Force Control* Master Thesis, Department of Automatic Control, Lund Institute of Technology, Nov 2000

[9] Mathworks, Inc. *Mathworks* http://www.mathworks.com/, 2005

[10] R. M. Murray, Z. Li, S. S. Sastry *A Mathematical Introduction to Robotic Manipulation* CRC Press, 1994

[11] K. Nilsson *Personal Correspondence* 2004

[12] S. Y. Nof *Handbook of Industrial Robotics* John Wiley& Sons, 1999

[13] M. Norrlöf *On Path Planning and Optimization Using Splines* Technical Report LiTH-ISY-R-2490, Department of Electrical Engineering, Linköping University, 2003

[14] M. Nyström, M. Norrlöf *PGT-A Path Generation Toolbox for Matlab* Technical Report LiTH-ISY-R-2542, Department of Electrical Engineering, Linköping University, 2003

[15] T. Olsson *Feedback Control and Sensor Fusion of Vision and Force* Licentiate Thesis, Department of Automatic Control, Lund Institute of Technology, 2004

[16] J. Rix *Controlling IRB6 with Java* Instruction paper, Department of Automatic Control, Lund Institute of Technology, 2005

[17] H. Seraji *Adaptive Admittance Control: An Approach to Explicit Force Control in Compliant Motion* IEEE International Conference on Robotics and Automation, Vol. 4, pp 2705-2712, 1994

[18] B. Siciliano, L. Villani *Robot Force Control* Kluwer Academic Publishers, 1999

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

[20] Sun Microsystems, Inc. *Java Sun Technology* http://java.sun.com/, 2005

[21] G. Zeng and A. Hemami *An Overview of Robot Force Control* Robotica, Vol. 15, pp 473-482, 1997

[22] M. Åkerblad *Optimized Path Following* Technical Report, Never Published, Department of Electrical Engineering, Linköping University, 2004

Appendix A

Matlab Codes

A.1 Codes for Kinematics

```
% FORWARD6
%
% Computes the forward kinematics of IRB6 given the joint values
% and tool-length.
%
%   T44 = FORWARD6(JOINTS,TOOL_LENGTH)
%
% The transformation matrix T44 is returned.

function T44 = forward6(joints,tool_length)

if nargin < 2,
  tool_length = 0;
end

% Check whether joint values are within limits
if joints(1) > 170*pi/180,

    error('Joint One out of range'); return;
elseif joints(1) < -170*pi/180,
    error('Joint One out of range'); return;
end
if joints(2) > 40*pi/180,
    error('Joint Two out of range'); return;
elseif joints(2) < -40*pi/180,
    error('Joint Two out of range'); return;
end
if joints(3) > 40*pi/180,
    error('Joint Three out of range'); return;
elseif joints(3) < -25*pi/180,
    error('Joint Three out of range'); return;
end
if joints(4) > 90*pi/180,
    error('Joint Four out of range'); return;
elseif joints(4) < -90*pi/180,
    error('Joint Four out of range'); return;
end
if joints(2)-joints(3) > 40*pi/180,
    error('Joint TwoThree out of range'); return;
elseif joints(2)-joints(3) < -40*pi/180,
    error('Joint TwoThree out of range'); return;
```

```
end

L1 = 700; L3 = 450; L4 = 670; wrist2Flange = 95;
xi = [wrist2Flange+tool_length 0 0]';

v1 = [0 0 157]';
v2 = v1+rotzm(joints(1))*[0 0 L1]';
v3 = v2+rotzm(joints(1))*rotym(joints(2))*[0 0 L3]';
v4 = v3+rotzm(joints(1))*rotym(joints(3))*[L4 0 0]';
C = rotym(joints(4))*rotxm(joints(5));
v5 = v4+rotzm(joints(1))*C*xi;
R = rotzm(joints(1))*C; % Rotation matrix
T44 = [R,v5;0 0 0 1];

% INVKIN6
%
% Calculates the inverse kinematics of IRB6.
%
%   JOINTS = INVKIN6(T44,TOOL_LENGTH)
%
function joints = invkin6(T44,tool_length)

if nargin < 2,
    tool_length = 0;
end
wrist2Flange = 95;

v5 = T44(1:3,4);
B = 157; L1 = 700; L3 = 450; L4 = 670;
xi = [tool_length+wrist2Flange 0 0]';
x = v5(1); y = v5(2); z = v5(3);

j1 = atan2(y,x);
if j1 > 170*pi/180,
    error('Joint One is out of range'); return;
elseif j1 < -170*pi/180,
    error('Joint One is out of range'); return;
end

C = inv(rotzm(j1))*T44(1:3,1:3);
k = C*xi; k1 = k(1); k2 = k(2); k3 = k(3);
vr = sqrt((x+sin(j1)*k2-cos(j1)*k1)^2+(y-sin(j1)*k1-cos(j1)*k2)^2);
v_r = vr; % L2 = 0;
z_4 = z-k3-L1-B;
L = sqrt(v_r^2+z_4^2);
A0 = acos((L3^2+L^2-L4^2)/(2*L3*L));
A1 = atan2(z_4,v_r);
j2 = -A1-A0+pi/2;

if j2 > 40*pi/180,
    error('Joint Two is out of range'); return;
elseif j2 < -40*pi/180,
    error('Joint Two is out of range'); return;
end

K0 = acos((L4^2+L3^2-L^2)/(2*L3*L4));
j3 = -K0+j2+pi/2;

if j3 > 40*pi/180,
    error('Joint Three is out of range'); return;
elseif j3 < -25*pi/180,
    error('Joint Three is out of range'); return;
```

```
end
if j2-j3 > 40*pi/180,
    error('Joint TwoThree is out of range'); return;
elseif j2-j3 < -40*pi/180,
    error('Joint TwoThree is out of range'); return;
end

j4 = atan2(-C(3,1),C(1,1));
j5 = atan2(-C(2,3),C(2,2));

if j4 > 90*pi/180,
    error('Joint Four is out of range'); return;
elseif j4 < -90*pi/180,
    error('Joint Four is out of range'); return;
end

joints(1) = j1;
joints(2) = j2;
joints(3) = j3;
joints(4) = j4;
joints(5) = j5;

%
% DERJOINTS
%
% Gives the angular velocities of the joints given the sampling time h.
% If no sampling time is given, h = 5e-3;
%
%   DJOINTS = DERJOINTS(JOINTS,H)
%

function djoints = derjoints(joints,h)

[A,B,C,D] = tf2ss([1 0],[0.01 1]);
Gss = ss(A,B,C,D);

t = 0:h:(length(joints(1,:))-1)*h;

for i = 1:length(joints(:,1)),
    y0 = (joints(i,1)-joints(i,2))/h;
    x0 = (10*joints(i,1)-y0)/100;
    djoints(i,:) = lsim(Gss,joints(i,:),t,x0)';
end
```

A.2 Codes for Trajectory Generation

```
% CART2JSP6
%
% Transforms the Cartesian path or section RPATH to cubic
% spline functions in joint space. The path or section NRPATH,
% which is equal to RPATH but with a description of the path
% in joint space attached, is returned. Two breakpoints are used.
% Only joint four and five are used to represent the orientation.
% They can be chosen but it will be fixed throughout the
% path. If no orientation is specified, they will be set to zero.
%
%   NRPATH = CART2JSP6(RPATH,MAXERROR,DELTA,J4,J5)
%
function nrpath = cart2jsp6(rpath,maxerror,delta,j4,j5)
```

```
if nargin < 4,
  j4 = 0;
  j5 = 0;
elseif nargin == 4,
  j5 = 0;
end

if isfield(rpath,'descr')
    k = length(rpath.descr);
    sec = rpath.descr;
else
    k = 1;
    sec = rpath;
end;

DELTA = delta;
C = rotym(j4)*rotxm(j5);

for j = 1:k % each section is transformed
    clear evalzone
    lc(1) = 0;
    first = 1;
    lc(2) = lc(1)+delta;
    if lc(end) > sec(j).range(end) %length between the breakpoints too large
        lc(end) = sec(j).range(end);
        DELTA = lc(end)-lc(1);
        lc(2) = lc(1)+DELTA;
    end;
    h = 10e-7;
    pg = evalsec(sec(j),lc(1),[],1);
    q(:,1) = invkin6C(1000*pg,C)'; % Modified invkin fcn for irb 6
    pg = evalsec(sec(j),lc(1)+h,[],1);
    q1 = invkin6C(1000*pg,C)';
    dq(:,1) = (q1-q(:,1))/h;

    while lc(1) < (sec(j).range(end)-10*eps)
        pg = evalsec(sec(j),lc(2),[],1);
q(:,2) = invkin6C(1000*pg,C)';
        pg = evalsec(sec(j),lc(end)-h);
        q2 = invkin6C(1000*pg,C)';
        dq(:,2) = (q(:,end)-q2)/h;
        tmp_jsp = completesp(lc,q,dq);

        %the path error is calculated for the spline tmp_jsp
        sp = evalsp(tmp_jsp,lc(1)+DELTA/2);
[a,b] = forward6C(sp');   % Modified forward kin fcn for irb 6
        P = evalsec(sec(j),lc(1)+DELTA/2);
        er = norm(a/1000-P);

        if er > maxerror  %path error too large
        DELTA = DELTA/2;
        lc(2) = lc(1)+DELTA;
        else
  if first
    jsp   = tmp_jsp;
    first = 0;
  else
    jsp=appendsp(jsp,tmp_jsp);
  end
  DELTA  = 2*DELTA;
  lc(1)  = lc(end);
  q(:,1) = q(:,end);
```

```
  dq(:,1)= dq(:,2);
  if lc(1) < sec(j).range(end)
    if lc(1)+DELTA > sec(j).range(end)
      DELTA = sec(j).range(end)-lc(1);
    end
    lc(2) = lc(1)+DELTA;
  end
end
    end
    if isfield(rpath,'descr')
      rpath.descr(j).jsp = jsp;
    else
      rpath.jsp = jsp;
    end
end
nrpath = rpath;


% LINEARPROG5
%
% Find the optimal linear acceleration given constraints in maximum
% and minimum linear accelerations, maximum linear velocity,
% maximum and minimum angular accelerations and maximum and minimum
% angular velocities. For the IRB6 robot.
% Limits = [vmax amin amax qdotlim qddotmin qddotmax]
%
%   X = LINEARPROG5(N,VPREV,QLC,Q2LC,LIMITS,DLC)
%
function x = linearprog5(n,vprev,qlc,q2lc,limits,dlc)

qddotmin = limits(5);
qddotmax = limits(6);
amin = limits(2)*ones(n,1);
amax = limits(3)*ones(n,1);
vmax = limits(1); qdotlim = limits(4);

f = zeros(n,1); f(1) = -1;

A1 = eye(n);
for j =1:n-1,
  A1 = A1+diag(ones(n-j,1),-j);
end
A1 = -2*dlc*A1;
b1 = (vprev^2)*ones(n,1);

A2 = -A1;
b2 = (vmax^2)*ones(n,1)-b1;

A3 = (qlc(1)^2)*A2;
b3 = (qdotlim^2)*ones(n,1)-(qlc(1)^2)*b1;

A4 = (qlc(2)^2)*A2;
b4 = (qdotlim^2)*ones(n,1)-(qlc(2)^2)*b1;

A5 = (qlc(3)^2)*A2;
b5 = (qdotlim^2)*ones(n,1)-(qlc(3)^2)*b1;

A6 = (qlc(4)^2)*A2;
b6 = (qdotlim^2)*ones(n,1)-(qlc(4)^2)*b1;

A7 = (qlc(5)^2)*A2;
b7 = (qdotlim^2)*ones(n,1)-(qlc(5)^2)*b1;
```

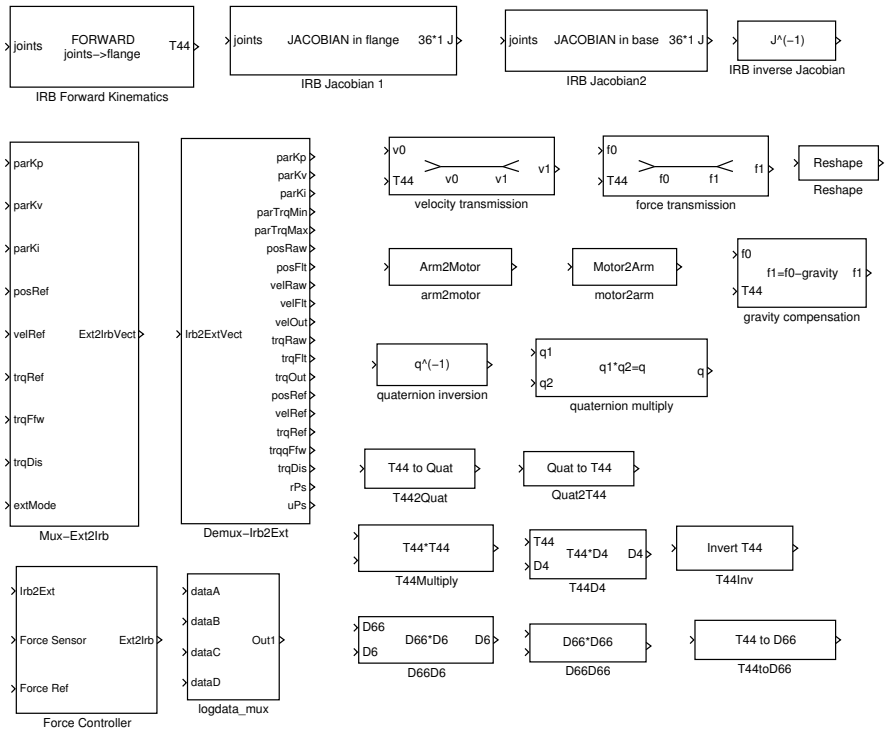
```
A9 = q2lc(1)*A1;
A9(:,1) = A9(:,1)-qlc(1);
b9 = -(qddotmin)*ones(n,1)+q2lc(1)*b1;

A10 = -A9;
b10 = (qddotmax)*ones(n,1)-q2lc(1)*b1;

A11 = q2lc(2)*A1;
A11(:,1) = A11(:,1)-qlc(2);
b11 = -(qddotmin)*ones(n,1)+q2lc(2)*b1;

A12 = -A11;
b12 = (qddotmax)*ones(n,1)-q2lc(2)*b1;

A13 = q2lc(3)*A1;
A13(:,1) = A13(:,1)-qlc(3);
b13 = -(qddotmin)*ones(n,1)+q2lc(3)*b1;

A14 = -A13;
b14 = (qddotmax)*ones(n,1)-q2lc(3)*b1;

A15 = q2lc(4)*A1;
A15(:,1) = A15(:,1)-qlc(4);
b15 = -(qddotmin)*ones(n,1)+q2lc(4)*b1;

A16 = -A15;
b16 = (qddotmax)*ones(n,1)-q2lc(4)*b1;

A17 = q2lc(5)*A1;
A17(:,1) = A17(:,1)-qlc(5);
b17 = -(qddotmin)*ones(n,1)+q2lc(5)*b1;

A18 = -A17;
b18 = (qddotmax)*ones(n,1)-q2lc(5)*b1;

A = [A1;A2;A3;A4;A5;A6;A7;A9;A10;A11;A12;A13;A14;A15;A16;A17;A18];
b = [b1;b2;b3;b4;b5;b6;b7;b9;b10;b11;b12;b13;b14;b15;b16;b17;b18];

x = linprog(f,A,b,[],[],amin,amax);

% PATH2TRAJ
%
% Transforms the joint space path or section NRPATH to TRAJ which
% is equal to NRPATH but with a velocity profile attached in the
% field TDI.
% limits give the limitations on the linear velocity, linear
% acceleration, angular velocity and angular acceleration.
% Default values: limits = [0.25 -2 2 4 -2 2]; n = 10; dlc = 0.02;
%
%   TRAJ = PATH2TRAJ(NRPATH,LIMITS,N,DLC)
%
function traj = path2traj(nrpath,limits,n,dlc)

if nargin < 2,
    limits = [0.25 -2 2 4 -2 2];
    n = 10; dlc = 0.02;
elseif nargin == 3,
    dlc = 0.02;
elseif nargin == 2,
    dlc = 0.02; n = 10;
end
```

```
if isfield(nrpath,'descr')
    k = length(nrpath.descr);
    sec = nrpath.descr;
    if sec(1).jsp.dim == 6,
        irb2000 = 1;
    else
        irb2000 = 0;
    end
else
    k = 1;
    sec = nrpath;
    if sec.jsp.dim == 6,
        irb2000 = 1;
    else
        irb2000 = 0;
    end
end

for j = 1:k,
    i_end = floor(sec(j).range(end)/dlc)+1;
    pos = evalsp(sec(j).jsp,0:dlc:(i_end-1)*dlc);
    djsp = dersp(sec(j).jsp);
    dpos = evalsp(djsp,0:dlc:(i_end-1)*dlc);
    ddjsp = dersp(djsp);
    ddpos = evalsp(ddjsp,0:dlc:(i_end-1)*dlc);
    if j == 1,
        v(1) = 0;
        dt(1) = 0;
    else
        if irb2000 == 1,
            x = linearprog6(n,v_end,dpos(:,1),ddpos(:,1),limits,dlc);
        else
            x = linearprog5(n,v_end,dpos(:,1),ddpos(:,1),limits,dlc);
        end
        a(1) = x(1);
        v(1) = sqrt(v_end^2+2*a(1)*dlc);
        if a(1) == 0,
            dt(1) = dlc/v(1);
        else
            dt(1) = -v_end/a(1)+sign(a(1))*sqrt((v_end/a(1))^2+2*dlc/a(1));
        end
    end
    for i = 2:i_end,
        if irb2000 == 1,
            x = linearprog6(n,v(i-1),dpos(:,i),ddpos(:,i),limits,dlc);
        else
            x = linearprog5(n,v(i-1),dpos(:,i),ddpos(:,i),limits,dlc);
        end
        a(i) = x(1);
        v(i) = sqrt(v(i-1)^2+2*a(i)*dlc);
        if a(i) == 0,
            dt(i) = dlc/v(i);
        else
            dt(i) = -v(i-1)/a(i)+sign(a(i))*sqrt((v(i-1)/a(i))^2+2*dlc/a(i));
        end
    end
    if j == k,
        vmax = limits(1);
        for i = 1:5,
            limits(1) = (5-i)*vmax/5;
            if irb2000 == 1,
```

```
                x = linearprog6(n,v(i_end-6+i),dpos(:,i_end-5+i),ddpos(:,i_end-5+i),limits,dlc);
            else
                x = linearprog5(n,v(i_end-6+i),dpos(:,i_end-5+i),ddpos(:,i_end-5+i),limits,dlc);
            end
            a(i_end-5+i) = x(1);
            v(i_end-5+i) = sqrt(v(i_end-6+i)^2+2*a(i_end-5+i)*dlc);
            if a(i_end-5+i) == 0,
                dt(i_end-5+i) = dlc/v(i_end-6+i);
            else
                dt(i_end-5+i) = -v(i_end-6+i)/a(i_end-5+i)+ sign(a(i_end-5+i))*
sqrt((v(i_end-6+i)/a(i_end-5+i))^2+2*dlc/a(i_end-5+i));
            end
        end
    end
    t = cumsum(dt);

    % Resampling
    time = 0; m = 2; h = 0.005;
    tdi(1) = 0;
    for i = 1:i_end-1,
        while time < t(i+1),
            tdi(m) = dlc/(t(i+1)-t(i))*h+tdi(m-1);
            m = m+1;
            time = time+h;
        end
    end

    if isfield(nrpath,'descr')
        nrpath.descr(j).tdi = tdi;
    else
        nrpath.tdi = tdi;
    end
    v_end = v(end);
    clear v; clear tdi;
    clear a; clear dt; clear t;
    clear pos; clear dpos; clear ddpos;
end

traj = nrpath;
```

Appendix B

Robot Force Control Blockset



Appendix C

Simulink Models

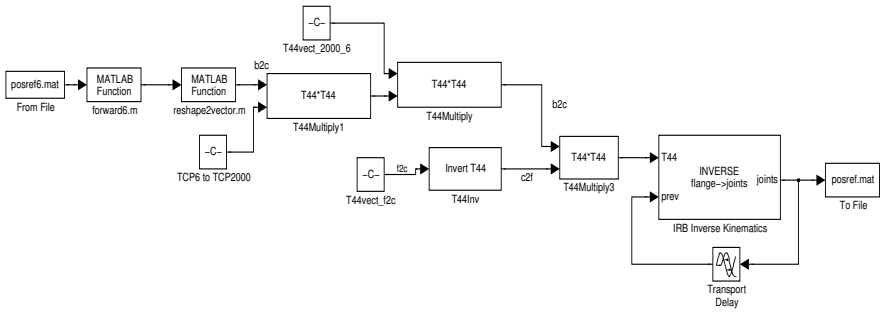


Figure C.1: Simulink model to generate reference trajectory for IRB2000

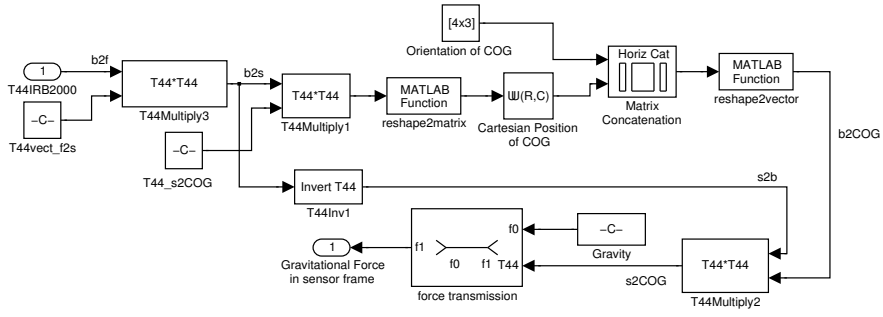


Figure C.2: Model to find the effect of gravity at the force sensor.

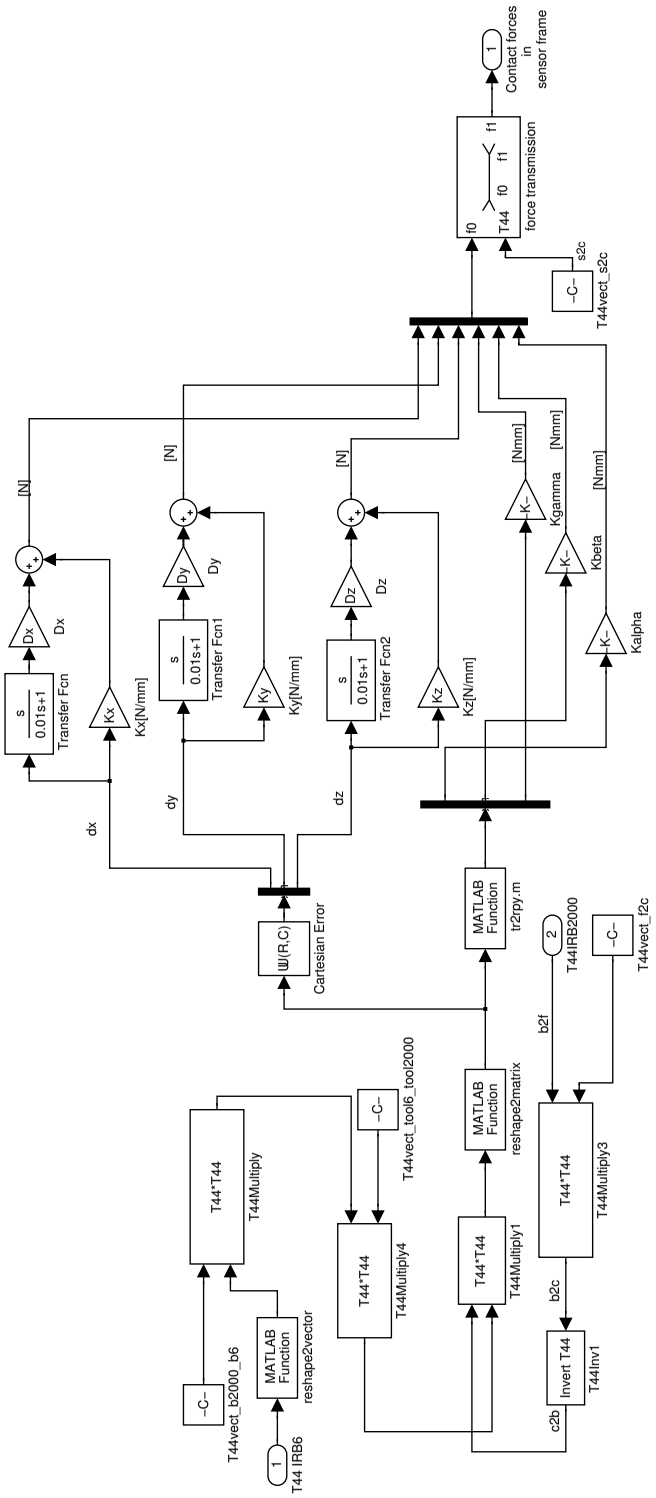


Figure C.3: Simulink model to simulate the contact forces

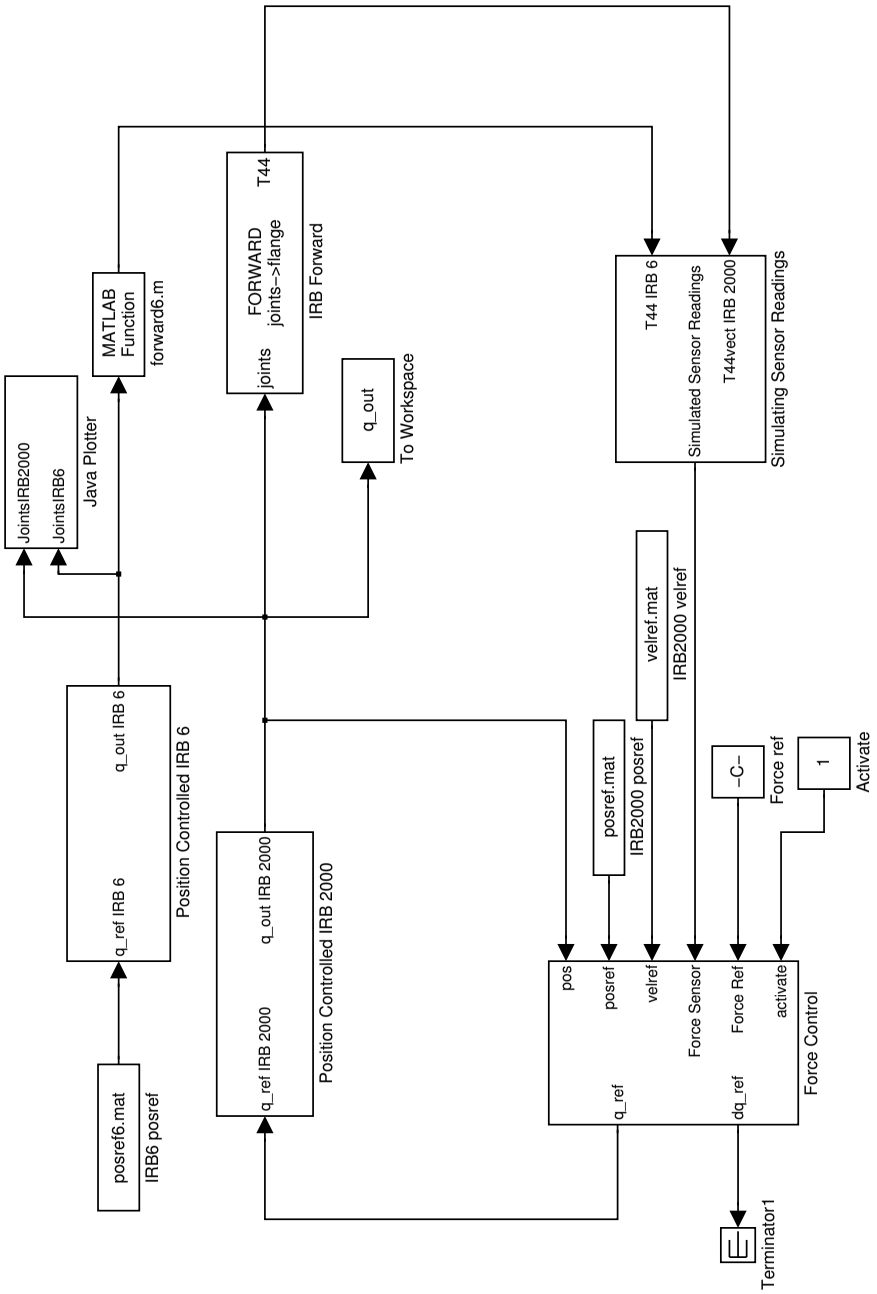


Figure C.4: Simulink model for simulations.

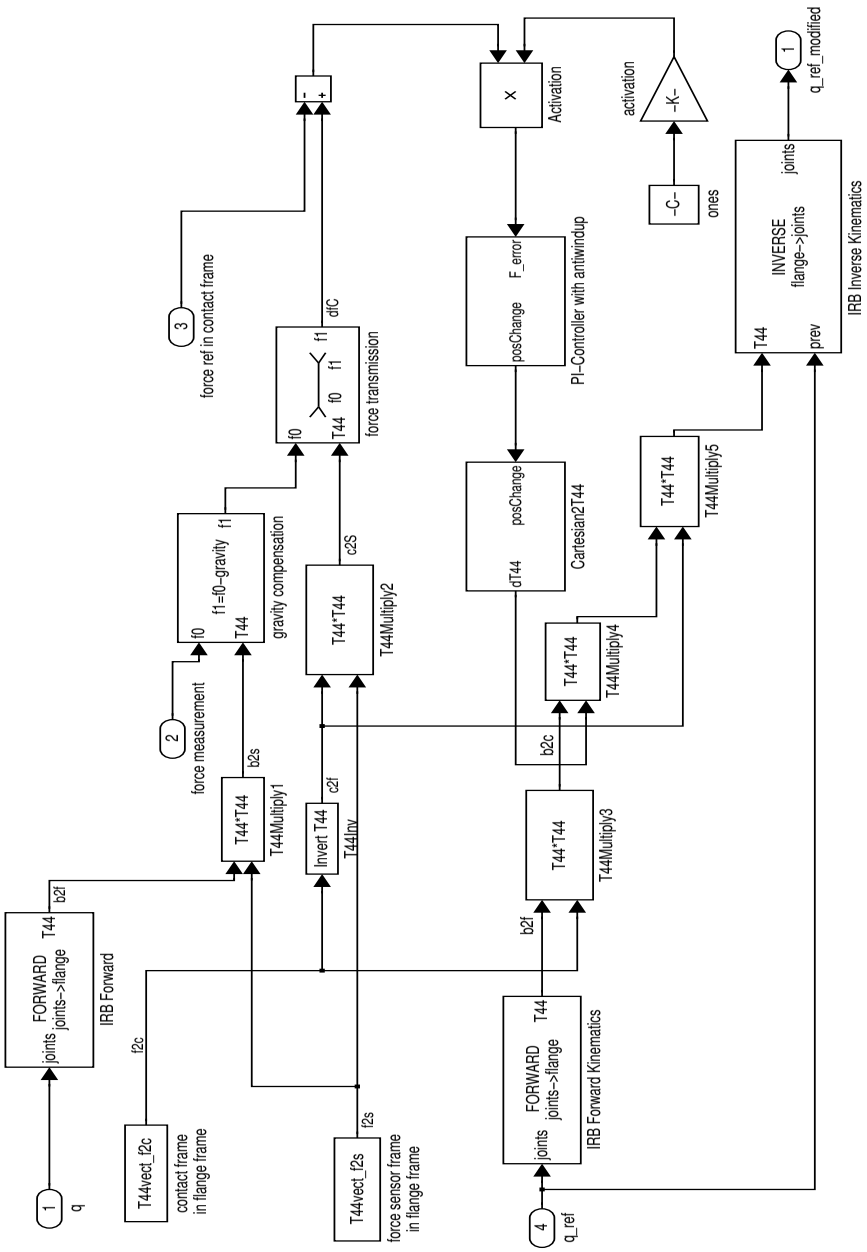


Figure C.5: Simulink model for Direct Force Control.

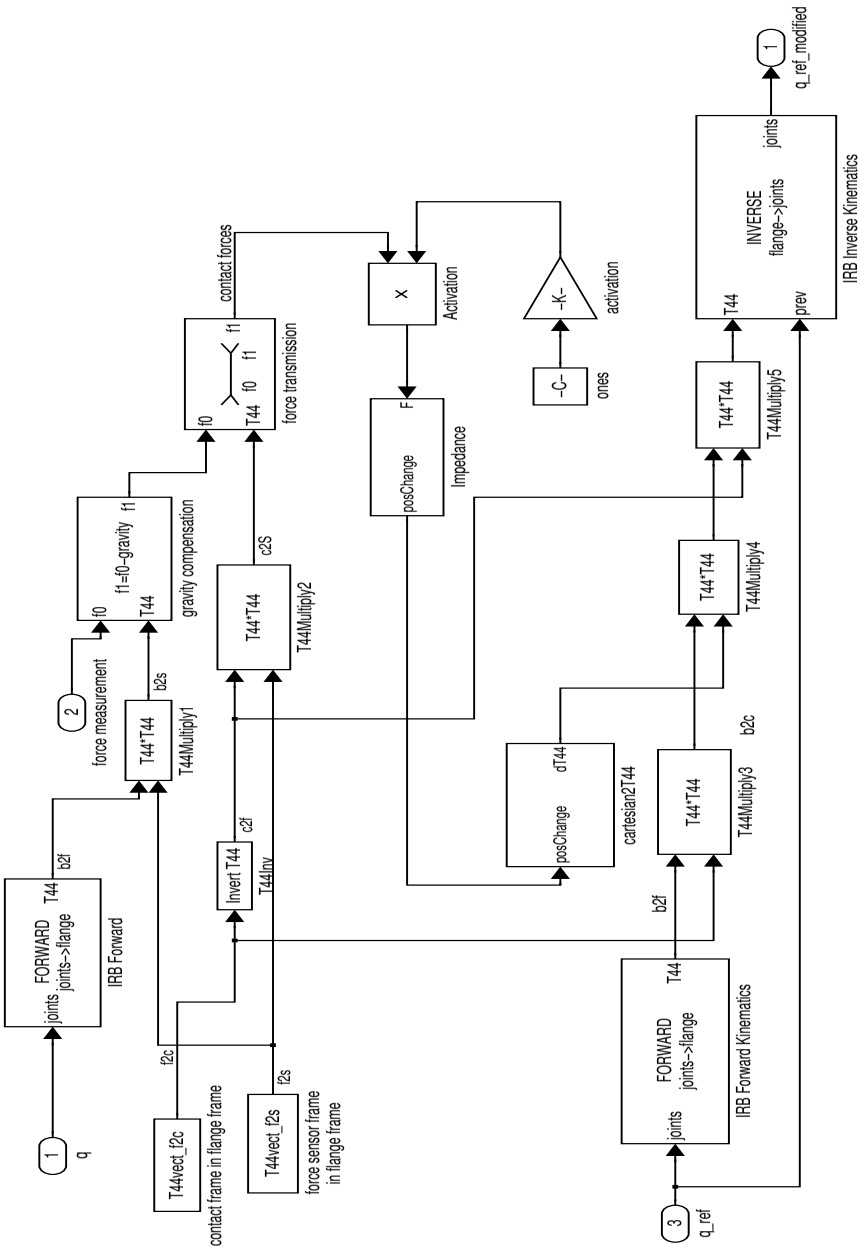


Figure C.6: Simulink model for Impedance Control.

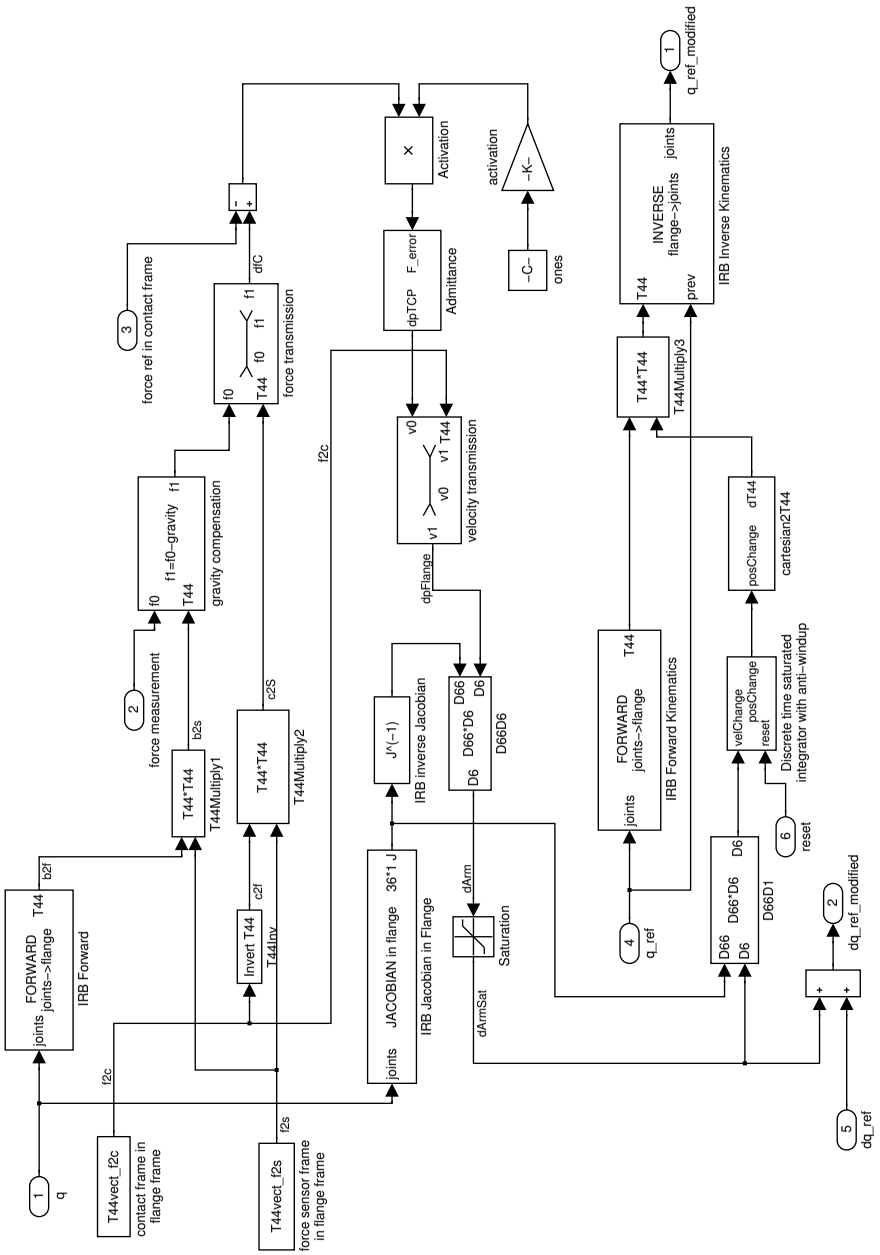


Figure C.7: Simulink model for Admittance Control.

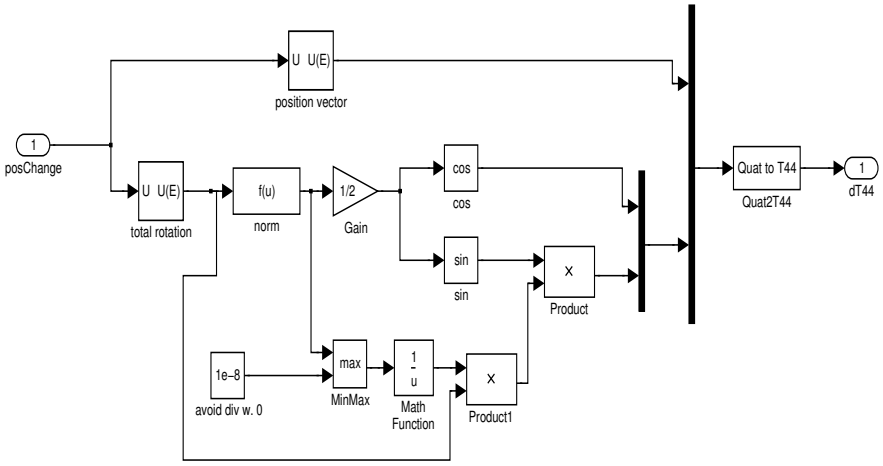


Figure C.8: Simulink model for conversion from Cartesian position change to transformation matrix from [15].

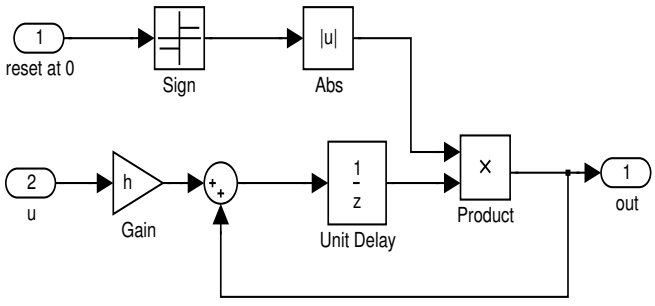


Figure C.9: Simulink model for discrete time integrator with reset.

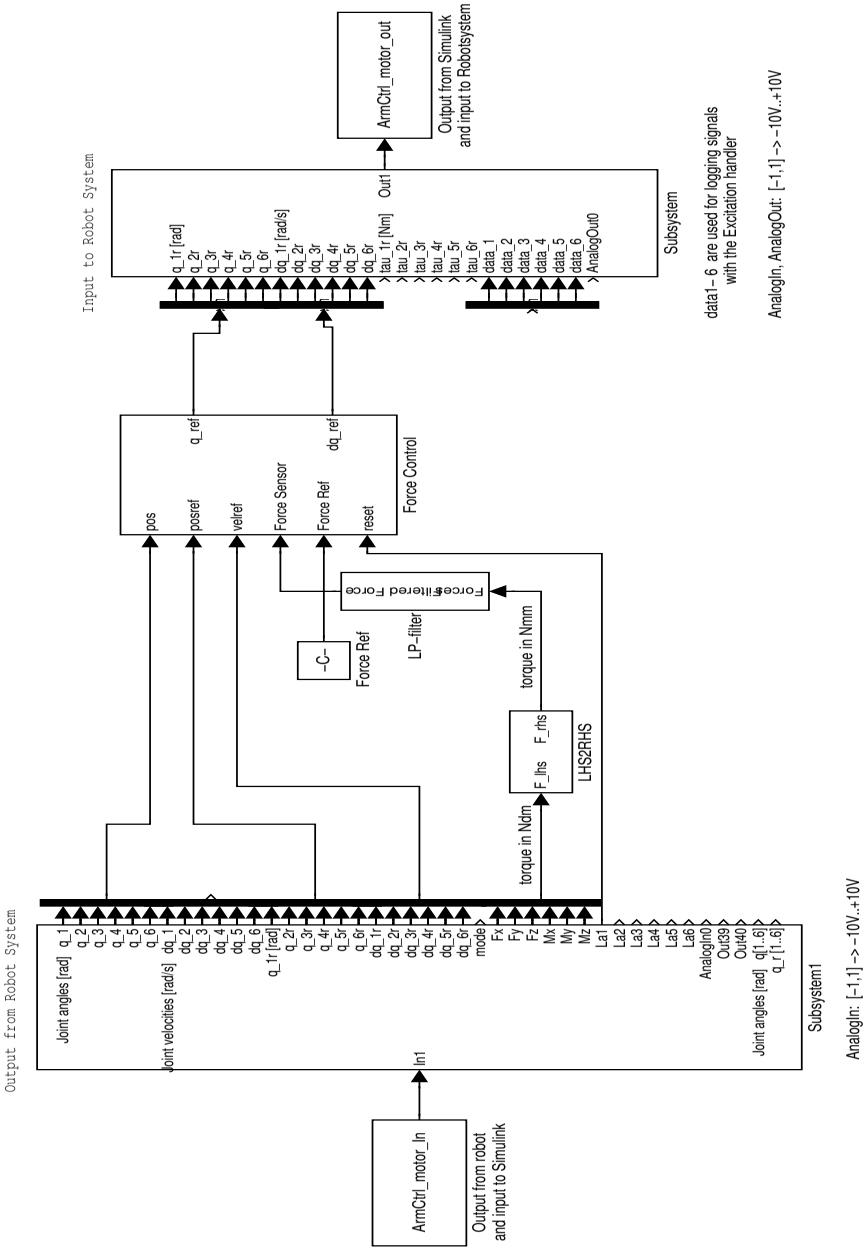


Figure C.10: Arm control template with force controller.