ISSN 0280–5316
ISRN LUTFD2/TFRT--5656--SE

Identify a Surface with Robot Force Control

Anders Olsson
Sara Liljenborg

Department of Automatic Control
Lund Institute of Technology
November 2000

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

Document name
MASTER THESIS

Date of issue
November 2000

Document Number
ISRN LUTFD2/TFRT--5656--SE

Author(s)
Anders Olsson and Sara Liljenborg

Supervisor
Anders Robertsson and Rolf Johansson

Sponsoring organisation

Title and subtitle
Identify a Surface with Robot Force Control
(Identifiera en yta med en kraftreglerad robot)

Abstract

The topic of this thesis is to identify a surface with robot force control. To achieve this a robot performing contact force control on a surface while it also follows a trajectory is needed. All experiments were performed in the robot laboratory at the Department of Automatic Control at Lund Institute of Technology. The robot used was an ABB Irb-2000 robot, equipped with a wrist mounted force and torque sensor of type JR3.

In the master thesis the robot kinematics is treated. Kinematics describes the geometric relationship between the motion of the robot in joint space and the motion of the tool in the task space. Furthermore a compensation for the gravitational force acting on the end-effector was implemented.

Direct force control has been used throughout this thesis. Direct force control operates on a force error between the desired and the measured values and aims to have a constant value of the contact force. When only controlling on three joints a PI force controller with variable proportional part is to be preferred. This because it can be tuned to be very fast, when in contact and thereby it can apply a constant force on an object without large force errors. The simulated result did not completely agree with the result achieved when the experiment was done in the reality. The reason is probably the assumption in the simulation that the force signal was without noise.
The result from the three-dimensional experiments was applied to the six joint controller structures. All the simulations using six joints worked successfully, both reorienting and reorienting while following a trajectory. In the simulations it was possible to identify parts of simpler surfaces such as planes, cylinders, and spheres.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
47

Recipient's notes

Security classification

Contents

1. Introduction . . . 3
- 1.1 Problem formulation . . . 3
- 1.2 Method . . . 3
- 1.3 Experimental platform . . . 3
- 1.4 Simulation platform . . . 5

2. Literature study . . . 6
- 2.1 Background . . . 6
- 2.2 Sensor based force control . . . 6
- 2.3 Force sensors . . . 7
- 2.4 Industrial application . . . 9

3. Theory . . . 9
- 3.1 Forward kinematics . . . 10
- 3.2 Inverse kinematics . . . 12
- 3.3 The Jacobian . . . 12
- 3.4 Gravity compensation . . . 12

4. Communication between *Matlab* and *Envision* . . . 13

5. Simulations and Experiments with three joints . . . 14
- 5.1 Simulation . . . 14
- 5.2 Experiment with only position control . . . 15
- 5.3 Experiment with position and force control . . . 16
- 5.4 Experiment with force and position controller on joint one . . . 16
- 5.5 Variable K in the force controller . . . 17
- 5.6 Remark on the force error . . . 20
- 5.7 Verifying the results in *Envision* . . . 22
- 5.8 Conclusions . . . 22

6. Simulations and Experiments with six joints . . . 23
- 6.1 Reorienting the end-effector normal to a surface . . . 23
- 6.2 Reorienting while following a trajectory . . . 26
- 6.3 Conclusions . . . 31

7. Practical problems . . . 31

8. Conclusions and recommendations . . . 31

9. Future work . . . 33

10. References . . . 34

A. Program lists . . . 35
- A.1 Forward . . . 35
- A.2 Jacobian . . . 35
- A.3 Gravity compensation . . . 36
- A.4 Communication between Matlab and Envision . . . 36
- A.5 Six joint control . . . 40

B. Controller charts . . . 45

C. Robot drawings . . . 47

1. Introduction

1.1 Problem formulation

The topic of this thesis is to identify a surface with robot force control. To achieve this a robot performing contact force control on a surface while it also follows a trajectory is needed. All the time it should keep the tool mounted in the gripper perpendicular to the surface in order to determine the normal. To solve this problem it was split into smaller parts:

- An overview in form of a literature studie of force controlled robots in industrial automation. This was done as a project in the course Industrial Automation [Liljenborg and Olsson, 2000], Chapter 2.
- Establish the transformation matrices for forward and inverse kinematics, Chapter 3.
- Compensate for the gravitational force acting on the gripper on the measurement side of the force sensor, Chapter 3.
- Make a connection between different simulation programs to be able to make both a numerical and a visual simulation, Chapter 4.
- To perform a controlled contact with a non parallel surface, follow a trajectory and to visualize the non parallellity in a simulation program, Chapter 5.
- To perform a controlled contact with a surface, follow a trajectory, reorient the tool in the gripper to be perpendicular to the surface and to visualize the surface in a simulation program, Chapter 6.

Some practical problems that came up during the work are discussed in Chapter 7. The results and conclusions are presented in Chapter 8. Some recommendations for extensions and future work is presented in Chapter 9.

1.2 Method

To learn *Envision* some of the laboration material from the robot technology course from the Division of Robotics, LTH was used, [Bolmsjö and Olsson, 1999]. All our programs and models have first been simulated in *Matlab/Simulink* and in *Envision*. Finally experiments have been performed on the open robot control system at the Department of Automatic Control [Nilsson, 1996].

In the first experiments the nominal experimental setup consists of one-dimensional force control perpendicular to a planar surface, where the surface is parallel to the robot links two and three, see Figure 2. Position control for joint two and three has been used to follow a trajectory along the surface and joint one to do the force control and to compensate for the load disturbance which occurs when the surface is not located parallel to the robot. To compare the result between different controllers the maximum movement of joint one while still keeping the contact with the environment is used.

1.3 Experimental platform

All experiments were performed in the robot laboratory at the Department of Automatic Control at Lund Institute of Technology. The robot used was

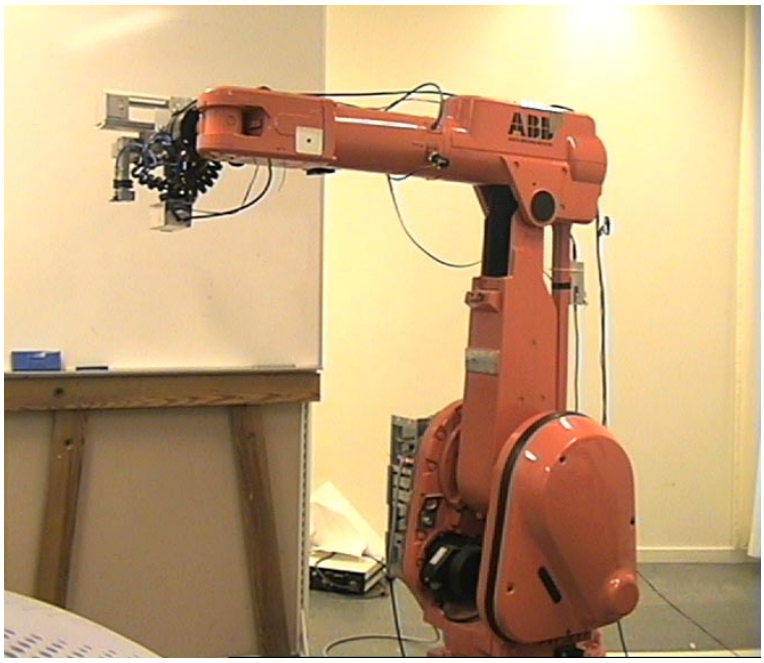

Figure 1 The ABB Irb-2000 robot.

an ABB Irb-2000 robot, equipped with a wrist mounted force and torque sensor of type JR3.

ABB Irb-2000 robot The Irb-2000 robot is an industrial robot which has six degrees of freedom (DOF), see Figure 1. Joint one, four, and six are of cylindrical type and joint two, three, and five are of revolute type, see Figure 2. To be able to go to any position with a arbitrary orientation the six DOF:s are needed. Mounted outside and attached to joint six is a force sensor. The force sensor's coordinate system is rotated 45 degrees in the positive direction of joint six. Outside the force sensor a gripper is mounted. Because of the rotation of the sensor the gripper is also rotated compared to joint six. In the gripper it is possible to attach a pen. The pen is used when drawing on a white board with the robot. The pen is attached to the gripper with a spring in order to make it more compliant. The spring also helps forces in the force sensor to build up slowly when the robot gets in contact with an object. The spring constant was measured to 0.35 N/mm.

Robot control system The robot control system was developed at the department and it allows a wide range of applications that the standard ABB control system will not permit [Nilsson, 1996]. The robot can be programmed in several different programming languages for example *Pålsjö* [Blomdell, 1997], [Eker, 1997], [Eker, 1999] *Matlab/Simulink* [Mathworks, Inc., 2000], *Modula 2*, or *C*-code. In this master thesis mostly *Pålsjö* and *Matlab/Simulink* had been used. *Pålsjö* is a real-time system which enables logging of joint angle values and forces from the sensor when running the robot. *Matlab/Simulink* with *Real-time workshop* [Mathworks, Inc., 2000] enables *Simulink* models to be translated to *C*-code and downloaded to the robot system. In the control system for the robot some *Matlab* functions and programs are available that can be used when control-

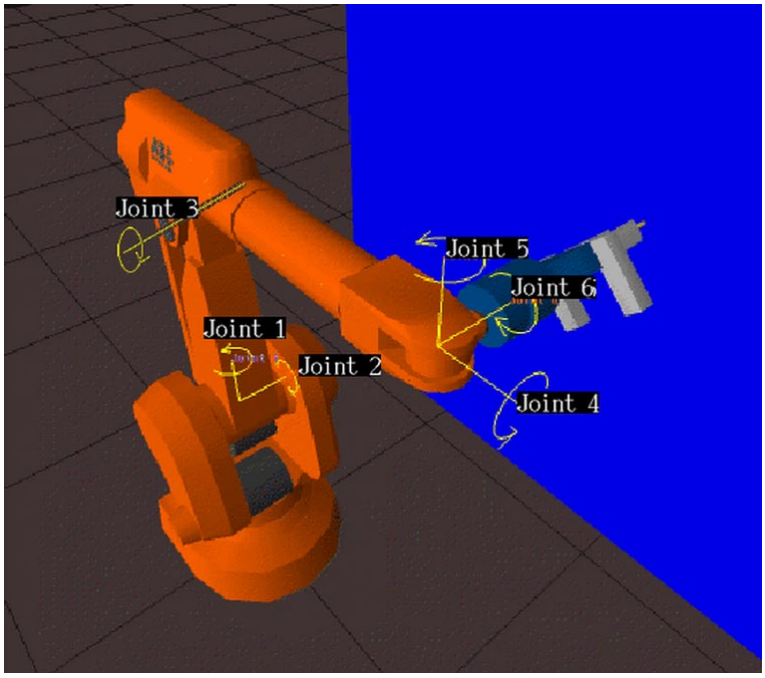

Figure 2 Joints on the ABB Irb-2000 robot.

ling the robot. Functions that will be mentioned later in this thesis is forward2400.m and invkin2400.m. A program much used was *Exc_handler*. This program was used when a trajectory was to be applied to the robot. The *Exc_handler* also allows logging of the current joint values, reference joint values, joint motor torques, joint velocities, and sensor forces. The sample rate for the position controller in the robot system is 5 ms.

Force sensor The force sensor used in the experiments was of type 100M40A manufactured by JR3 Inc., see Figure 4. The force sensor measures forces and torques in x-, y-, and z-direction. These forces and torques are read into the robot system via a DSP-board and can be reached from all the above mentioned software. The sample rate for the force sensor is 8 kHz.
To protect the sensor the gripper is mounted with a pneumatic lock that will come loose if the gripper is exposed for a large load. The sensor used in the experiments works in the range $\pm 400N$. Since the forces used in the experiments are very small compared to the measurement range, the measurement errors and the noise become very big.

1.4 Simulation platform

In the project, simulations of the developed controllers were an important part. Several different softwares were used. Due to that all controllers were implemented in *Matlab*/*Simulink* a first numerical simulation was performed there. When simulating in *Simulink* there was no opportunity to add the robot graphics, therefore a *Matlab* program called *Robotpos* was implemented, see Figure 5. This program shows the position and orientation of the robot for given joint values. When this was not sufficient a robot simulation software called *Envision* [Deneb Robotics, Inc., 2000]

was used. *Envision* has a robot library which makes it easy to import the wanted robot, it allows also in a convenient way to import 3D-CAM models. This makes it possible to build good simulation environment and to specify nominal trajectories. In order to use *Envision* with a *Matlab*/*Simulink* controller a network connection was needed between *Matlab* and *Envision*, which is further discussed in Chapter 4.

2. Literature study

2.1 Background

The research about force controlled robots has been going on for about 20 years. However, it has been hard to apply the research results to the product lines. There are only a few examples where force control has been a success. In some applications it has been easier to implement visual control instead. However, in the last few years the industry has been requesting sensor based force control again. In some occasions they can benefit from the use of force control to compliment the usual position control when programming a robot. Some examples:

1. Applying a limited force needed for a manufacturing process. When drilling a certain force is needed but the force is not allowed to exceed a certain limit, which will cause damage to the tool.
2. Pushing an object using a controlled force.
3. Dealing with geometric uncertainty by establishing controlled contacts. When polishing a surface a certain contact pressure is needed.

The last area is the most used. Experiments have been carried out trying to grind with a hybrid force position controlled robot. In the case where the tool was compliant it was successful. When hard tools are used they often break or the force sensor breaks. The main reason is that the time delay between detection of the force and response of the controller is too large.

2.2 Sensor based force control

Direct force control Direct force control operates on a force error between the desired and the measured values and aims to have a constant value of the contact force. Motion control capabilities along the unconstrained task directions are recovered using a parallel composition of the force and motion control actions. This is desirable in order to realize a compliant behavior only along those task directions that are actually constrained by the presence of the environment.

Hybrid control Hybrid control is a combination of control with force and torque sensors, and ordinary position control [Siciliano and Villani, 1999], [Shutter *et al.*, 1997]. The workspace is divided into purely motion-controlled directions and purely force-controlled directions. The position controlled part and the force controlled part results in two torques. The output is the sum of the torques. Force control and position control are decoupled with leads so that the control laws for each can be designed

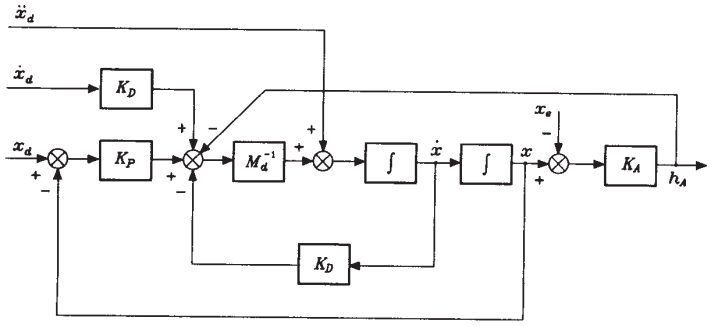

Figure 3 Block scheme for an impedance controller. M_d is the mass matrix, K_D the damping matrix, and K_P the stiffness matrix. h_A is the equivalent force. K_A is the environment stiffness matrix and x_e the undeformed environment rest position.

independently. Normally, the force control part consists of a PI-controller and the position part of a PD-controller. The reason for this is, in the position part, it is more desirable to have a quick response, while in the force part a smaller error is preferred. When a conflict between force and position errors occurs, the force error dominates over position error.

Impedance control In impedance control, the ratio of force to motion is regulated [Hogan, 1985], [Johansson and Spong, 1994], [Zeng and Hemami, 1997], [Whitney, 1987]. It can be implemented in many ways depending on how the measured signals are used. The velocity is used to modify the damping constant of the manipulator. The manipulator control system does not only follow a trajectory, but also changes the impedance of the manipulator. Impedance control is often used when the robot needs to adapt to the damping of its environment. The block scheme of a manipulator in contact with an elastic environment under impedance control is in Figure 3. It can also be discriebed with Equation 1 where $\tilde{x} = x_d - x$.

$$M_d\ddot{\tilde{x}} + K_D\dot{\tilde{x}} + K_P\tilde{x} = h_A \tag{1}$$

x_d, $\dot{x}_d$, and $\ddot{x}_d$ represent the position, velocity and acceleration references. M_d is the mass matrix, K_D the damping matrix, and K_P the stiffness matrix. h_A is the equivalent force.

Parallel control Both hybrid control and impedance control have to cope with imperfections, such as unknown robot dynamics, measurement noise, and other external disturbances. To overcome these problems experiments have been carried out with combined hybrid and impedance control called parallel control. A combination between hybrid and impedance control makes it possible to distinct impedance for the position controlled and for the force controlled part. This makes it possible for the controller to maintain position or velocity and follow a force trajectory. Two impedances can be chosen by the user, one for the velocity part and one for the force part of the controller.

2.3 Force sensors

Force measurements are mainly based on determination of an equilibrium condition between two forces. The one which is better known is taken as

Figure 4 The sensor used in the experiments.

a reference in order to determine the value of the other. Force sensors are built according to the principal strain gauges on an elastic component or the piezoelectric method.

The elastic component on the strain gauge deforms elastically when it is loaded. This deformation causes the gauges to deform which causes its resistance to change. The voltage over the resistance can be amplified and measured. The measured voltage is proportional to the load, which makes it possible to estimate the force. In a strain gauge based transducer, the conversion chain is force-stress-strain-resistance-output voltage. At every conversion step in this chain parasitic influences can interfere with the result and may cause a loss in accuracy. The placement of the strain gauges on the elastically component depends on the measurement direction of the force or torque respectively. It also depends on if the sensor should be sensitive to temperature or not. The gauges are connected to each other in bridges. There are several kinds of bridges depending on if the sensor is used to measure only force or both force and torque. Some of these bridge couplings are not sensitive to the temperature component of the strain [Ekdahl, 1999]. Force sensors should be placed at locations close to application point of external forces such as end-effector or the tool itself.

The piezoelectric method uses a piezoelectric material that yields an electrical charge when it is mechanically loaded. In contrast to sensors using strain gauges, no flexing spring element is required. The force to be measured induces a mechanical stress that loads the piezoelectric material and produce an output signal. As piezoelectric material quartz can be used.

A force sensor is designed after the following criteria [Nasri, 1999]:
Application related requirements:

- accuracy,

- sensitivity,
- rigidity,
- dimension,
- force and moment range,
- weight,
- compact shape,
- robust construction.

Manufacturing related aspects:

- simple design of elastic component,
- easy gauge installation.

Transducers for forces from 0.1 to 10 MN are commercially available and used for industrial as well as research purposes. A typical force sensor is shown in Figure 4.

2.4 Industrial application

Industrial manufacturing is the largest area in which robot force sensors are used. Examples of applications were robot force sensors can be used [CRS, 2000]:

- product life cycle testing,
- force monitoring and control,
- dispensing of sealants and adhesives,
- deburring of metals and plastics,
- assembly machines and work cells,
- touch control and effort sensing,
- material handling.

3. Theory

The modeling of industrial robots is usually divided into kinematic and dynamic modeling. Kinematics of a robot refers to the geometric relationship between the motion of the robot in joint space and the motion of the tool in the task space. In Figure 5 the two frames are represented by the coordinate systems $o_0x_0y_0z_0$ and $o_Sx_Sy_Sz_S$ respectively. Robot dynamics are not treated in this thesis.

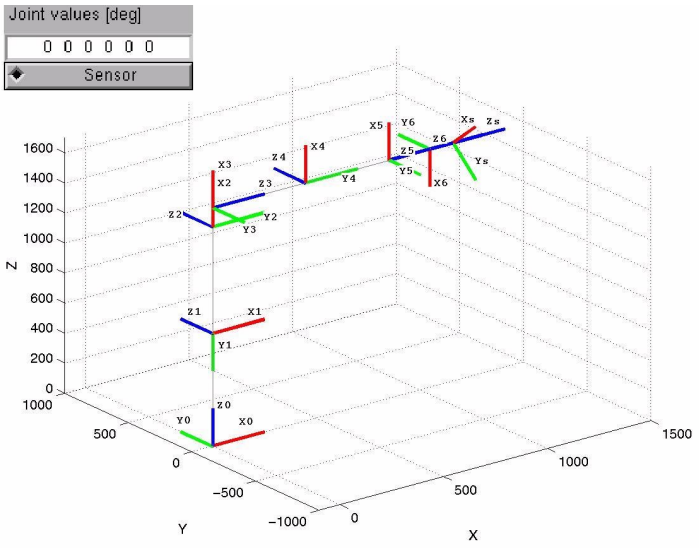

Figure 5 The coordinate systems for the robot.

link	a_i $[mm]$	α_i	d_i $[mm]$	θ_i
1	0	-$\pi/2$	750	θ_1
2	710	0	0	θ_2-$\pi/2$
3	125	-$\pi/2$	0	θ_3-θ_2
4	0	$\pi/2$	850	θ_4
5	0	-$\pi/2$	0	θ_5
6	0	0	100	θ_6
S	0	0	Tool length	$\pi/4$

Figure 6 The a_i, α_i, d_i, and θ_i parameters for the ABB Irb-2000 robot.

3.1 Forward kinematics

When the joint angles for all joints are known, the position and orientation for the end-effector can be calculated using forward kinematics. The orientation of the coordinate systems for the joints of the robot was investigated and the axes not defined by ABB robot standard were defined as shown in Figure 5. The standard Denavit-Hartenberg convention has been used, see Figure 7 [Denavit and Hartenberg, 1955], and the following definitions. With the coordinate system and the dimensional drawings of the robot, Appendix C, the parameters a_i, α_i, d_i, and θ_i could be obtained, Figure 6.

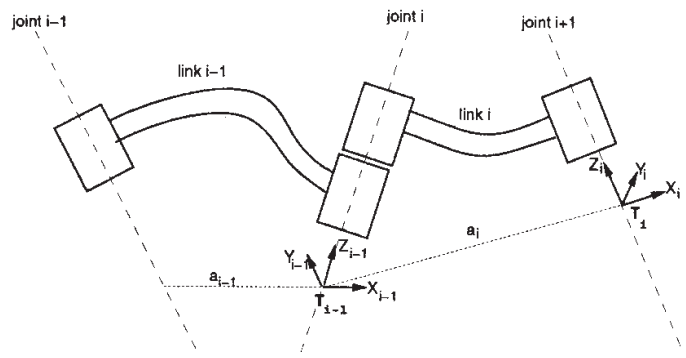

Figure 7 Standard Denavit-Hartenberg convention

a_i, link length	the offset distance between the z_{i-1} and z_i axes along the x_i axis.
α_i, link twist	the angle from the z_{i-1} axis to the z_i axis about the x_i axis.
d_i, link offset	the distance from the origin of frame i-1 to the x_i axis along the z_{i-1} axis.
θ_i, joint angle	the angle between the x_{i-1} and x_i axes about the z_{i-1} axis.

The transform, ${}^{i-1}_{i}T$, relation between coordinate system i with respect to coordinate system $i-1$ could be calculated as Equation 2 [Spong and Vidyasagar, 1989]

$$ {}^{i-1}_{i}T = Rot_{z,\theta_i} \cdot Trans_{z,d_i} \cdot Trans_{x,a_i} \cdot Rot_{x,\alpha_i} \tag{2} $$

The general form of the ${}^{i-1}_{i}T$ matrix then becomes:

$$ {}^{i-1}_{i}T = \begin{pmatrix} cos(\theta_i) & -sin(\theta_i) \cdot cos(\alpha_i) & sin(\theta_i) \cdot sin(\alpha_i) & a_i \cdot cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \cdot cos(\alpha_i) & -cos(\theta_i) \cdot sin(\alpha_i) & a_i \cdot sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

The last row in Equation 3.1 is just filled out to get a square matrix that is easy to use in calculations. The elements in the T matrix could be explained in the following matrix:

$$ {}^{i-1}_{i}T = \begin{pmatrix} {}^{1}_{1}R & {}^{1}_{2}R & {}^{1}_{3}R & p_x \\ {}^{2}_{1}R & {}^{2}_{2}R & {}^{2}_{3}R & p_y \\ {}^{3}_{1}R & {}^{3}_{2}R & {}^{3}_{3}R & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3} $$

were the elements p represent the position for the ith coordinate system relative to the $(i-1)$th coordinate system and the R matrix represents the orientation.

To change between coordinate systems, the T-matrices between the systems has to be multiplied. For example, to get the transformation matrix

from the base coordinate to the sensor (frame S) all the T matrices has to be multiplied.

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

To calculate forward kinematics forward2400.m was used when transforming between coordinate system 0 to 6 otherwise the *Matlab* code in Appendix A.1 was used.

3.2 Inverse kinematics

If the end-effector position and orientation are known the joint angles can be calculated using inverse kinematics. To simplify it a bit the problem is separated into two subproblems. This separation is possible because the last three joint axes intersect in one point. First calculate the wrist position were the last three joint axes intersect from the known TCP, tool length, and orientation. From the wrist position the first three joint angles are then calculated. After this the last three joint angles can be found. Unfortunately there usually exists more than one solution. The function used was invkin2400.m.

3.3 The Jacobian

The Manipulator Jacobian can be used as a transformer of forces and torques between different coordinate systems. The Jacobian can also transform linear and angular velocity between different coordinate systems. The analytical Jacobian is a different matrix calculated from the derivatives of the position and orientation. The analytical Jacobian is not treated in this thesis.
The matrix operator

$$ P\times = \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix} $$

corresponds to taking the cross product with the vector $p = [p_x\, p_y\, p_z]^T$ and is used to calculate the Jacobian [Craig, 1989]. The Jacobian is calculated with the R matrix, Equation 3, and the matrix operator $P\times$. Equation 4 shows the force and torque Jacobian.

$$ J_{force} = \begin{pmatrix} {}^{0}_{S}R & 0 \\ {}^{0}_{S}P \times {}^{0}_{S}R & {}^{0}_{S}R \end{pmatrix} \tag{4} $$

The Jacobian for transforming linear and angular velocity is the transpose of Equation 4. Several m-files were written to support the calculations of which some are presented in Appendix A.2.

3.4 Gravity compensation

Because of the gravitational force acting on the end-effector outside the sensor, the values read from the sensor varies a lot depending on the robot configuration and in which position the force sensor was reset. This problem was solved using the Jacobian and an assumption that the force sensor was reseted when the end-effectors gravitational force only was acting in the sensors z-direction. The end-effector weight could be determined when the

robot was in its home position, all joints equal to zero, by reading the sensor's force in the z-direction or the absolute value of the x and y forces. To get a better accuracy of the end-effector weight several readings were made when the robot was in different positions. This readings were made when joint one to four and six were set to zero and joint five was set to values between -90 and 90 with 10 degrees step between the readings. With simple trigonometry Equation 6 could be found and the mass computed.

$$\sqrt{F_x^2 + F_y^2} = m \cdot g \cdot \cos\alpha \quad (5)$$

$$F_z + m \cdot g = m \cdot g \cdot \sin\alpha \quad (6)$$

A mean value of the computed mass from all the measurements was calculated and found to be m = 7.0 kg. When weighing the end-effector on a scale the mass was found to be 6.7 kg. In all further calculations m = 7.0 kg is used, this because all other collected force data is read from the sensor.

The center of gravity (COG) of the end-effector could also be determined using the collected data. By dividing the torques in the x direction with the forces in the y direction or the other way around the lever from the sensor to the COG of the end-effector could be computed. The lever was computed to 139 mm. With a known end-effector COG, a Jacobian from the COG to the sensor frame was computed. To eliminate the gravitation force a force was applied at the COG in the opposite gravitational force direction. This force was mapped to the sensor frame using the Jacobian. When adding the mapped force from the COG and the read forces from the sensor a gravitational compensation was made which eliminates the gravitational force independent of the robot configuration. The *Matlab* source code for this operation is presented in Appendix A.3.

4. Communication between *Matlab* and *Envision*

To visualize the results from a robot simulation were the controller is implemented in *Matlab* a link between *Matlab* and *Envision* is needed [Olsson, 2000]. In *Envision* there is an environment called LLTI (Low Level Telerobotic Interface). This environment is described in [Olsson, 2000] and in the *Envision*-help. The main stucture of *Envision* is shown in Figure 8. LLTI provides a possibility to either control a real robot from the simulation program or to control the modeled robot in *Envision*. To be able to use the LLTI a C-program which both handles the interface with *Envision* and acts like a network client had to be implemented. The server which communicates with the client was developed in *Matlab* as a m-file. The communication between the client and the server uses the local network and a communication software called *Matcomm* which is developed at the department [Blomdell, 1997]. The use of these programs makes it possible to execute *Envision* commands from *Matlab*/*Simulink*. There are two types of commands that can be executed, either CLI-commands (Command Line Interpreter) or LLTI-commands. When a CLI-command is to be executed a text string is sent from the *Matlab* server using the function runCLI.m. The client receives the string and adds a code which tells *Envision* that

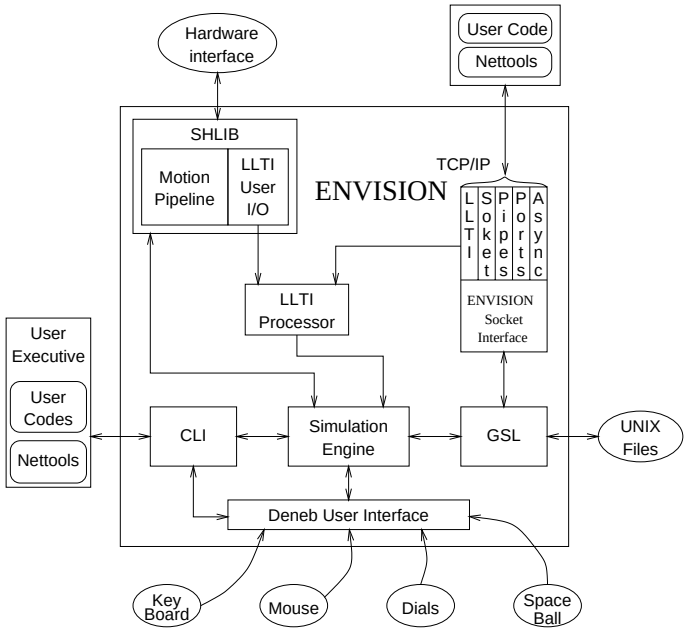

Figure 8 The ommunication used between *Matlab* and *Envision* was the LLTI (Low Level Telerobotic Interface).

the received string is a CLI-command. Both the code and the string is returned to *Envision*. If a LLTI-command is to be executed then an array of type double is sent from the *Matlab* server using the m-file runLLTI.m. The client will return the array to *Envision* without altering it. How these command arrays are used is described in the *Envision* online help under the section LLTI [ENVISION, 2000]. All code for the communication is to be found in Appendix A.4.

5. Simulations and Experiments with three joints

The purpose of the experiment was to examine if the robot could follow a trajectory on a surface not parallel to the robot and to visualize the nonparallellity in *Envision*. The trajectory was chosen to be a circle with a diameter of 500 mm, see Figure 9. To follow this trajectory only three joints are needed. Joint 2 and 3 for the circle trajectory in the plane and joint 1 for compensationg for the deviation perpendicular to the circle.

5.1 Simulation

The circle trajectory was generated in the robot simulation program *Envision*. The trajectory made only uses joint two and three on the robot. In order to verify that the trajectory was correct a simulation in *Envision* was performed, this to minimize the risk of failure when the trajectory was to be applied to the real robot. During a simulation in *Envision* it is also possible to see if the robot joints stay within their mechanicle boundaries. In order to make a good simulation a model of the experiment platform was created in *Envision*, see Figure 9. When the simulation performed well the trajectory was saved to a ASCII text file. This file contained joint values

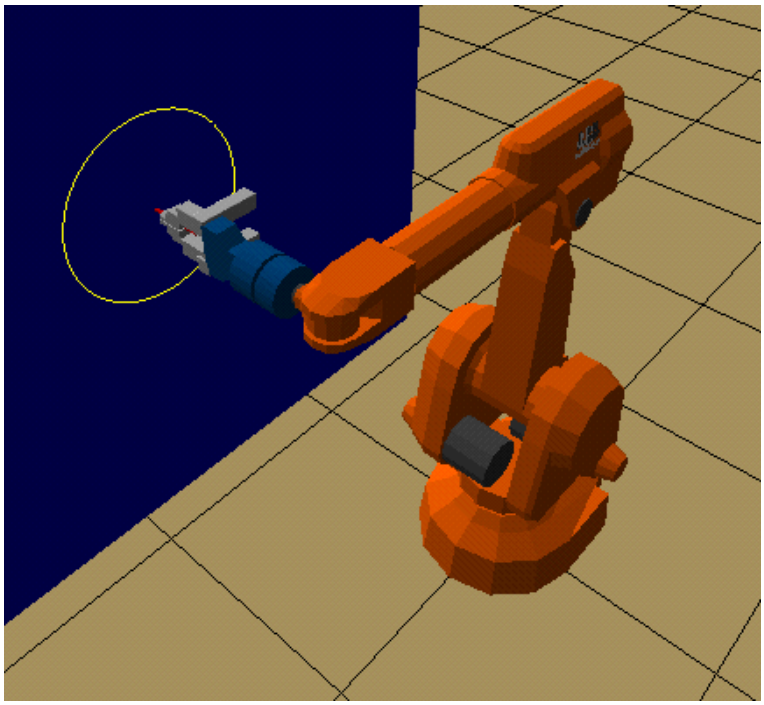

Figure 9 Simulation setup

for joint two and three and the corresponding time for these values.

5.2 Experiment with only position control

In order to apply the *Envision* generated trajectory to the robot system from *Matlab* some corrections had to be made. The trajectory was sampled with a 200 ms interval, however the robot system uses a sample time of 5 ms. To solve this a simple interpolation of the trajectory's joint values were made. The robot system also requires all joint values to be in motor radians. The conversion for joint six is calculated differently due to a mechanical coupling between joint five and six.

For joint 1..5.

```
ValueInJointDEG(i)=180/pi/N(i)*ValueInMotorRAD(i);
```

For joint 6.

```
ValueInJointDEG(6)=180/pi/N(6)*(ValueInMotorRAD(6)
                                +ValueInMotorRAD(5));
```

where N_i is the gear ratio for joint i.
After these conversions the trajectory could be downloaded as position reference for the internal position controller of the robot using the *Exc_handler*. The trajectory was applied to joint two and three, while the other joints were given zero reference value and should stay in their initial positions. When running the trajectory the non parallellity of the white board towards the robot showed. It resulted in that the robot only had been in contact with the white board during some of the time.

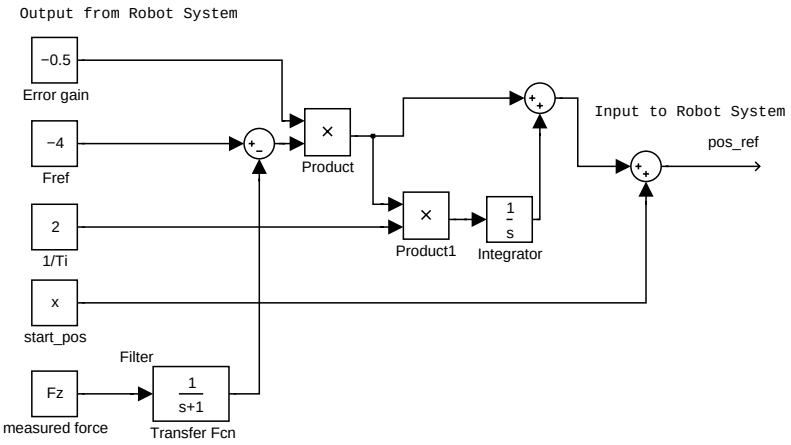

Figure 10 This is the force controller model that was downloaded to the robot system.

5.3 Experiment with position and force control

To compensate for the uncertain angle of the white board a force controller was implemented in *Matlab*/*Simulink*. The controller type used was a direct force PI-controller, see Figure 10, which gives an input to the robot systems position controller. This controller should keep the force measured by the sensor at $2N$, i.e. a $2N$ force acting on the white board. Because of a great deal of noise when reading from the force sensor, the signal is filtered through a low pass filter. When applying this controller to joint one and then running the trajectory the robot stayed in contact with the white board during the entire run but the maximum movement of joint one could only be about 0.9 degrees. The force and the filtered force is plotted in Figure 11. The parameters used was $F_{ref} = -2$, $K_f = -0.5$ and, $T_i = 0.5$.

In order to reduce the force sensor noise effect on the controller the force reference was raised from $2N$ to $4N$ and the speed of the circle trajectory was cut in half. After these changes the maximum movement of joint one for still beeing in contact was about 6.2 degrees. The plot of the force can be seen in Figure 12. The plot reveals that the filtered force is in the interval of $4N \pm 0.7N$ which means that the controller fails to keep the force constant at $4N$. In the plot it also can be seen that the measured force suffered from noise. The controller parameters used in this experiment was $F_{ref} = -4$, $K_f = -0.5$ and, $T_i = 0.4$.

5.4 Experiment with force and position controller on joint one

Another control structure was implemented with a proportional position controller. Instead of giving an position reference to the system's position controller a torque reference direct to the robot motor for joint one was the output of this controller. This controller is shown i Figure 13. When running the slow circle with a force reference of $4N$ about the same result was reached as in Section 5.3, see Figure 14. The values for the plot is taken at the same angle as when using force control. The control parameters used is $K_{Force} = -0.8$, $T_{iForce} = 0.5$, and $K_{Position} = 0.5$.

Some experiments with a PI-force controller and a PD-position controller was also accomplished but it proved to be very difficult to tune

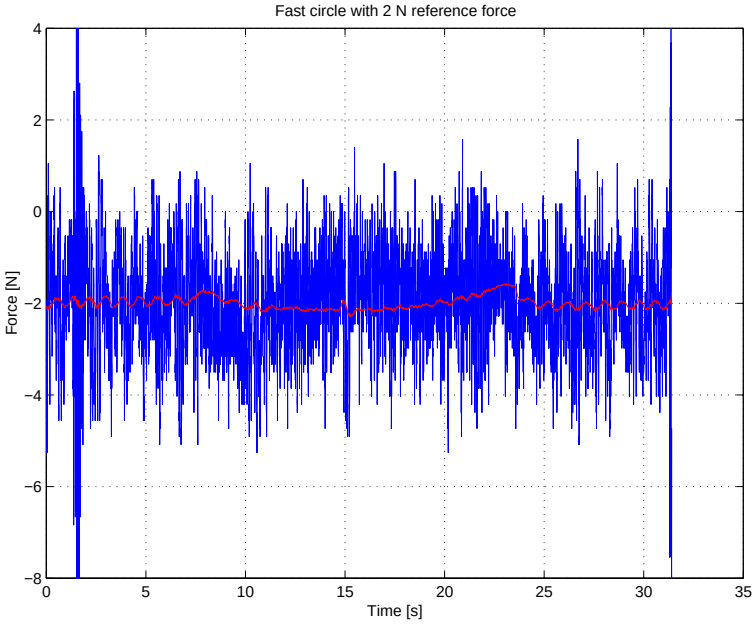

Figure 11 The force signal and the filtered force signal when drawing a circle at the higher speed. The force controller in Figure 10 is used.

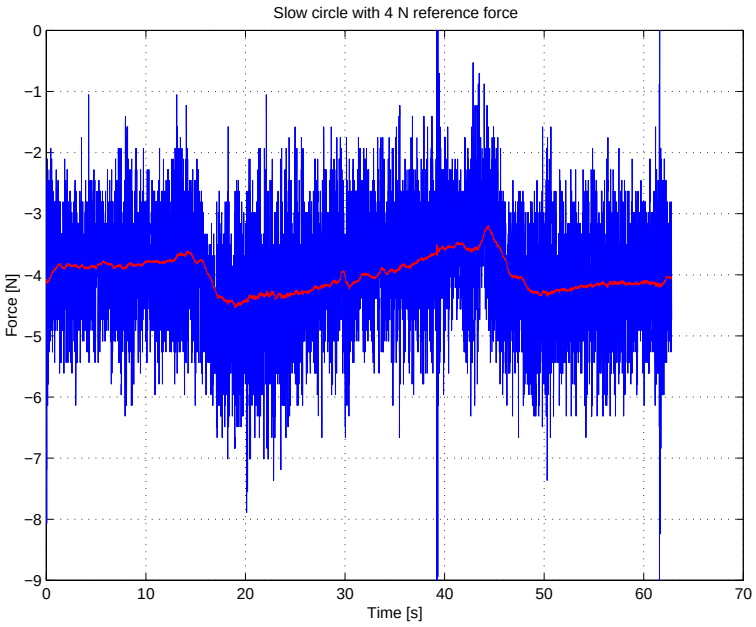

Figure 12 The force signal and the filtered force signal when drawing a circle at the lower speed. The force controller in Figure 10 is used.

the derivative part. This also probably because of the noise from the force sensor.

5.5 Variable K in the force controller

One problem during the experiments was the proportional K parameter in the force controller. When the robot is approaching the surface a small

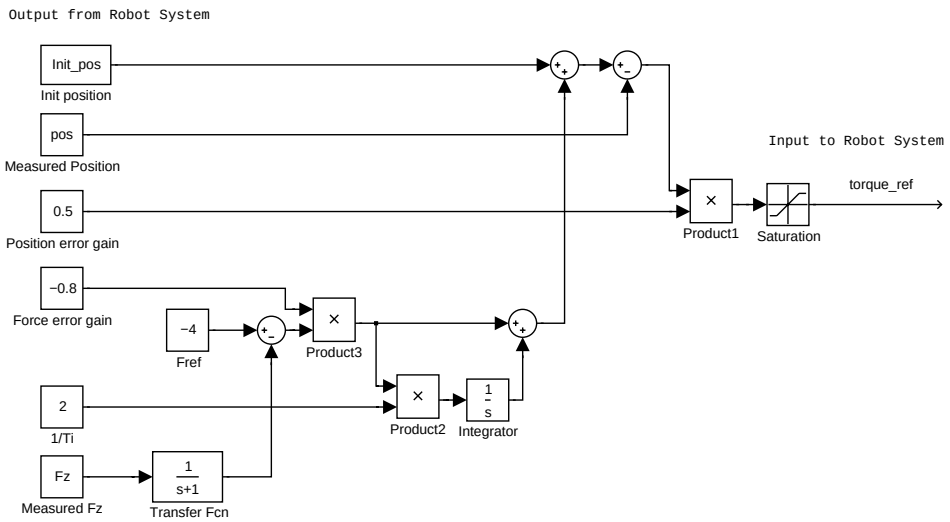

Figure 13 The force position controller used in the experiments.

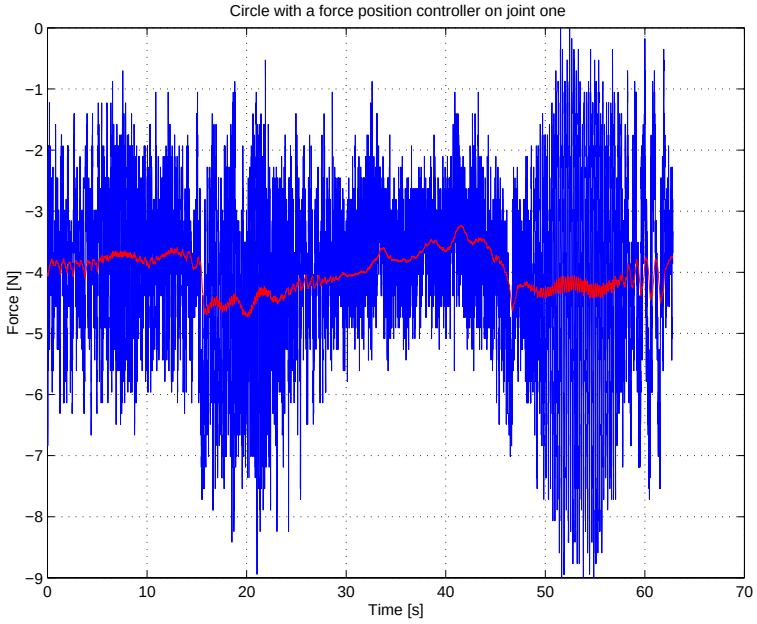

Figure 14 The force signal and the filtered force signal when drawing a circle at the lower speed. The force position controller in Figure 13 is used.

K is needed otherwise it will smash too hard into the surface. But when the robot is in contact a big K is desirable so it will not loose the contact. To handle this gain scheduling have been tried with respect to the measured contact force. A simulation in *Simulink* was made to try out different control parameters, see Figure 15. K varies as the following equation:

$$varK = K \cdot (-atan(Fz)/\pi + 1/2) \tag{7}$$

The used parameters was $K = -1.3$, $Ti = 0.25$ and, $Fref = -4N$. The simulation setup is described closer in Chapter 6.1. The model was changed to run at the robot system, see Figure 16. The same experiments as in the

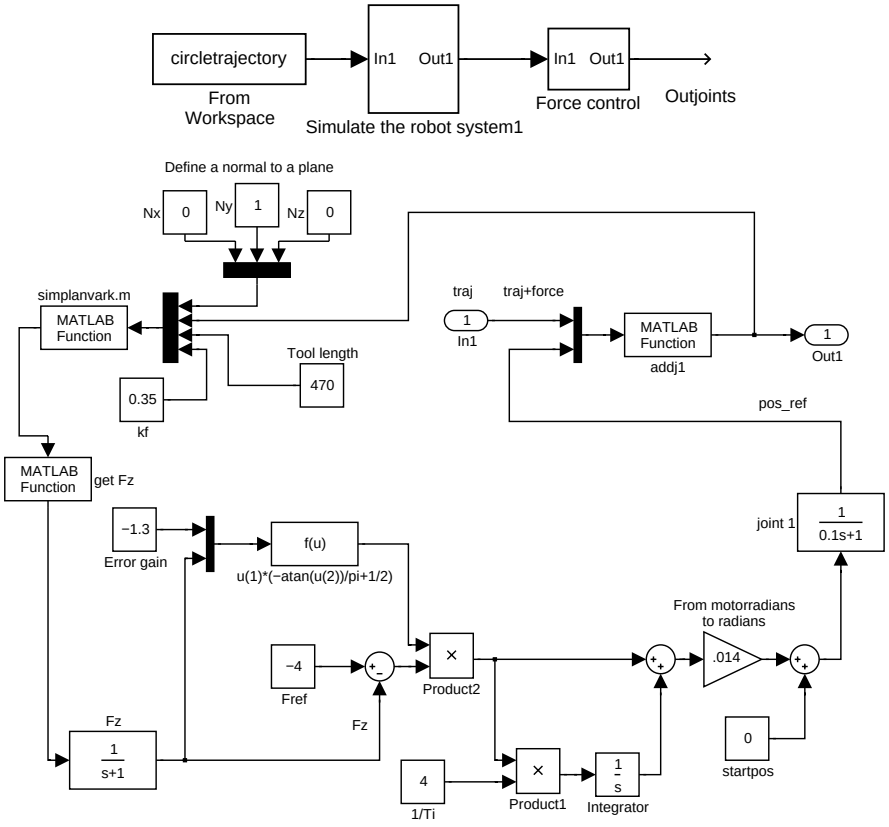

Figure 15 The *Simulink* models when experimenting with variable K. The system below is the subsystem in the box *Force Control* to the right in the upper block diagram. The box in the middle simulates the robot and is explained in Chapter 6.1.

sections above was performed but a much better result was achieved. The K and Ti was changed to give a slower system than in the simulation and then with small steps the system was made quicker. The best result was estimated when $K = -1$ and $Ti = 0.5$. During the run joint one has moved 16.0 degrees. Movement up to about 24.0 deg was possible but then the force varies a lot. Even bigger movements were tried but then the robot lost the contact with the white board and once the contact was lost the variable K parameter becomes much smaller and it takes a while to establish contact again.

When drawing the circle on a non parallel surface the projection will be ellipsoidal or egg-shaped. The drawn curve can be seen in the sequence of pictures in Figure 18.

To achieve even better results a pole placement were tried. To do a proper pole placement design more variables than K and T_i was needed so also the filter used to filter the measured force was used in the design. With the pole placement the following values were estimated $K = -0.2$, $T_i = 0.15$, and the time constant in the filter $M = 0.1$. The simulation became much better with this values but in the reality they did not work as well. The time constant for the filter could not be less than $M = 0.25$ otherwise the system became unstable and then the parameters used before were faster.

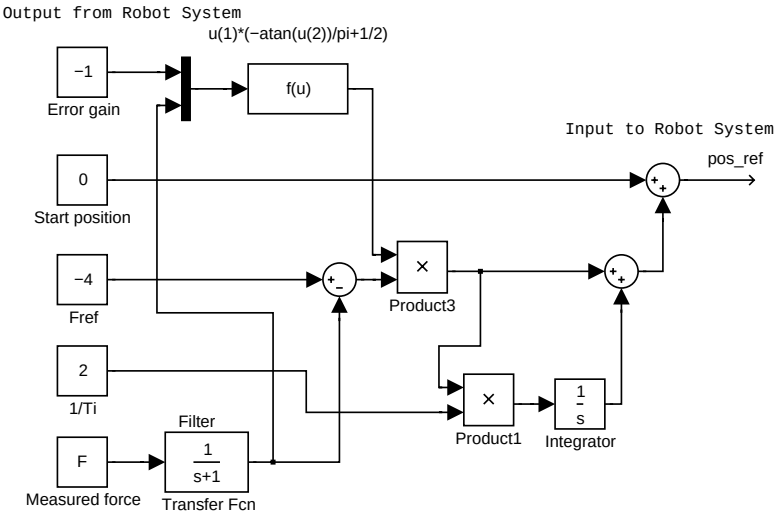

Figure 16 The variable K model that was tried on the robot.

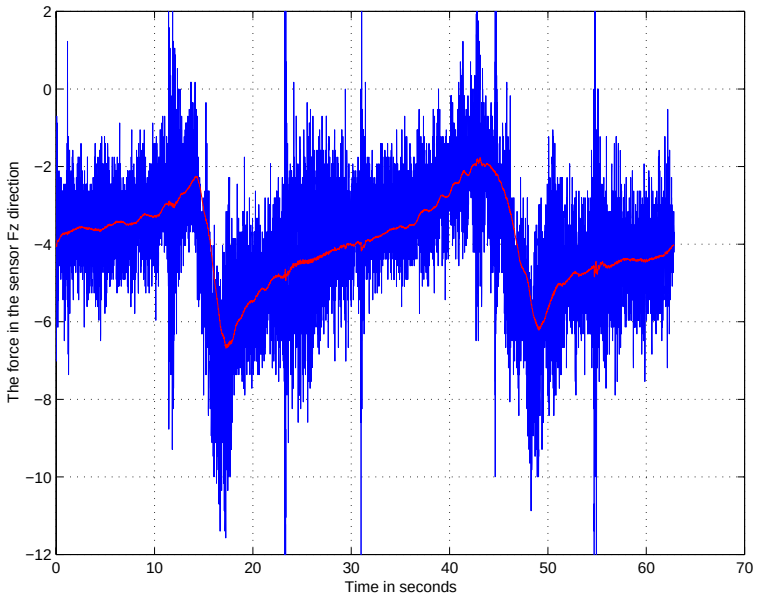

Figure 17 The contact force between the robot and the white board when using the variable K controller in Figure 16.

5.6 Remark on the force error

As can be seen especially in Figure 17 the force varies. From the block scheme in Figure 19 the transfer functions from the reference force F_{ref} and from the load disturbance D to the control error e, could be calculated:

$$E = \frac{1}{1 + G_f \cdot G_p \cdot G_c} \cdot R - \frac{G_f \cdot G_p}{1 + G_f \cdot G_p \cdot G_c} \cdot D \tag{8}$$

The integral part in the controller takes care of constant load disturbances. However, following a trajectory along a non parallel surface will give a load disturbance which increases/decreases with respect to the horizontal pro-

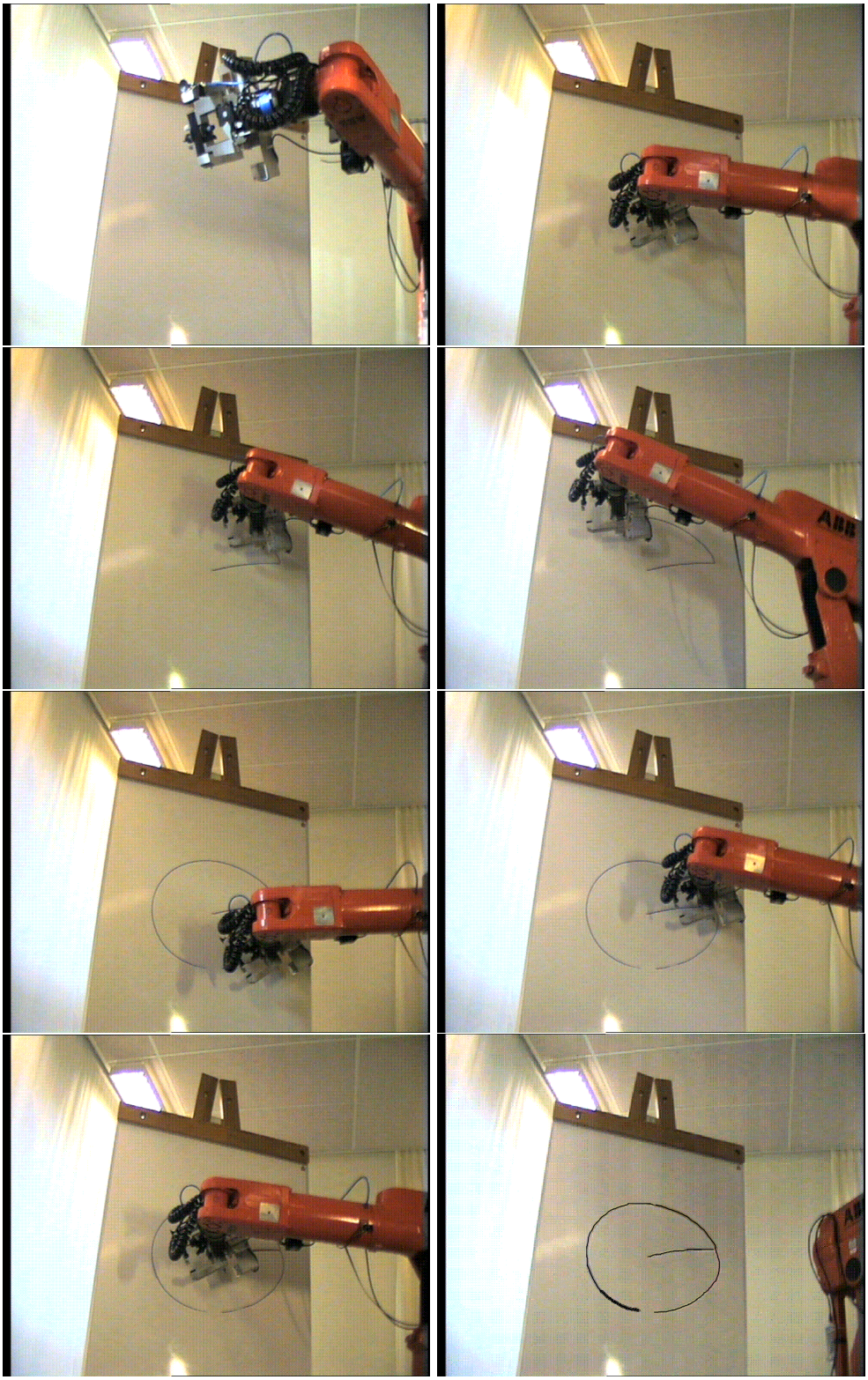

Figure 18 The robot draws after a circle trajectory using variable K. The ellipsoidal form can be seen.

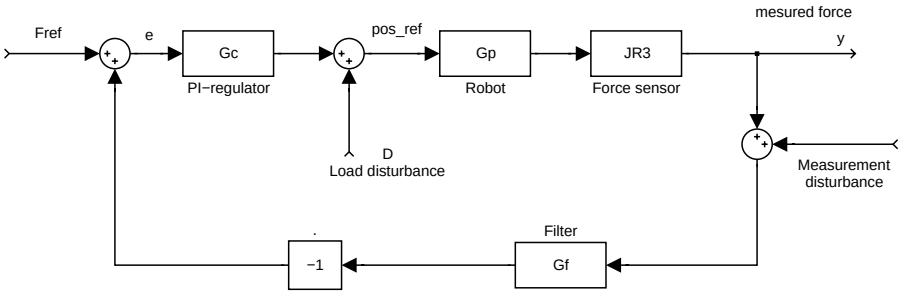

Figure 19 The schematic figure over the variable K.

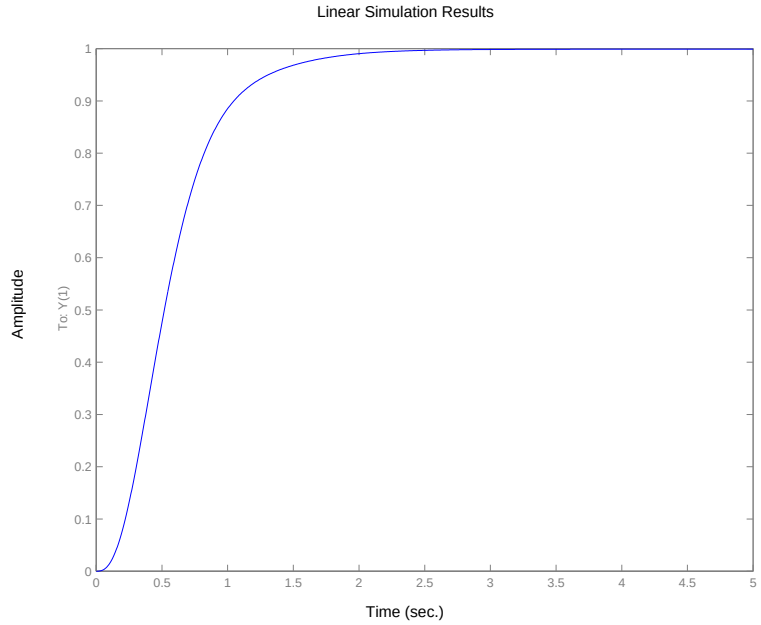

Figure 20 The theoretically calculated force error for a disturbance ramp.

jection of the trajectory. When plotting the response of a step disturbance for the transfer function in Equation 8 the error will go to zero. When plotting the transfer function against a disturbance ramp the error will go to one, see Figure 20, which is in the same area as the error in Figure 17.

5.7 Verifying the results in *Envision*

During the experiments joint values for joint one, two, and three were recorded and stored. After some conversions *Envision* could read the values. When putting a trace in *Envision* on the robot's TCP the non parallellity of the white board could be seen and measured when running through the collected joint values, see Figure 21.

5.8 Conclusions

The best result were achieved when the variable K model was used. The simulated result did not completely agree with the result achieved by experiments. The reason is probably the assumption in the simulation that the force signal was without noise. In *Envision* the orientation of the white

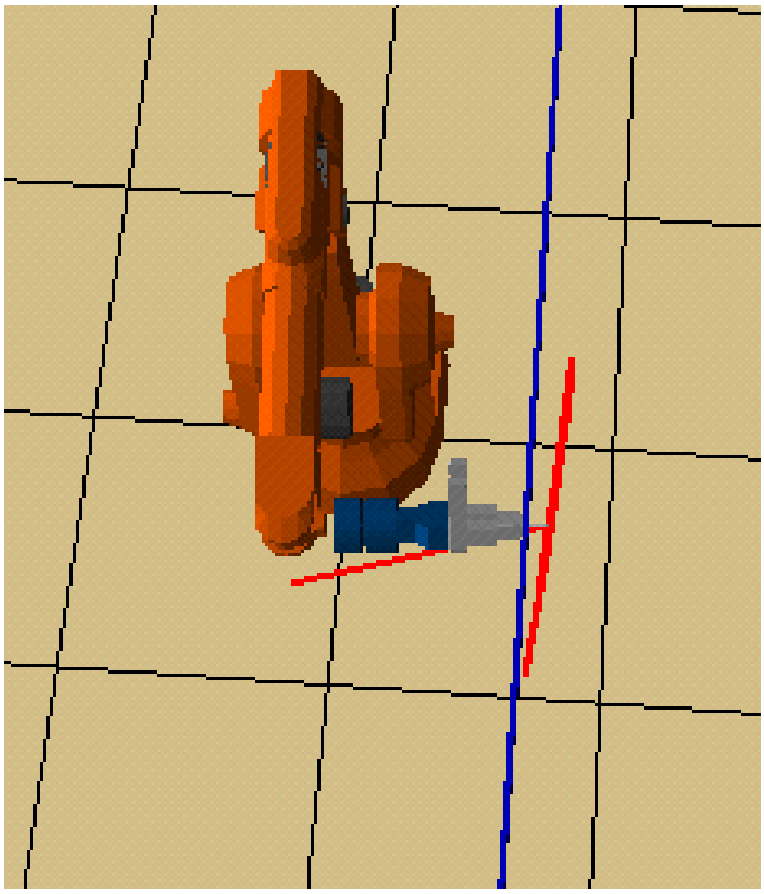

Figure 21 The non parallellity of the real white board compared to the virtual can be seen.

board compared to the robot could be seen very clearly.

6. Simulations and Experiments with six joints

To be able to both follow a trajectory and reorient along a surface all six joints are needed.

6.1 Reorienting the end-effector normal to a surface

The purpose of the experiment was to make the robot perform controlled contact with a surface and then reorient its end-effector to be perpendicular to that surface. This is accomplished when there is only a force in the force sensor's z-direction, i.e. no forces in the sensor's x- and y-direction. In order to reorient the end-effector all six joints had to be controlled. When the end-effector is not in contact with a surface it should move in the end-effector's z-direction until it comes in contact with a surface.

Simulating a plane In order to make a realistic simulation a virtual surface had to be implemented. The virtual plane was chosen by defining a fix point on the plane and the normal vector to it. The function simulating

link	samples	$T[s]$
1	22.5	0.1125
2	19.5	0.0975
3	14	0.07
4	20	0.1
5	16.5	0.0825
6	22	0.11

Figure 22 The estimated time constants for the joints.

the plane returned forces in x-,y- and z-direction to the controller depending on the position of the robot relative to the defined plane. Forces in z-direction was set to zero if the Tool Center Point (TCP) of the robot was not in contact with the plane. Else if the TCP had past through the plane the force in z-direction was calculated to be proportional to the distance to the plane. Forces in x- and y-direction was estimated by comparing the end-effector normal with the normal to the plane. Using the Jacobian a comparation of the two frames were possible by mapping the plane's coordinates and orientation in world coordinates to the end-effector frame. The force in x-direction was chosen to the x-component of the plane's normal in end-effector frame. The force in y-direction was chosen to the y-component of the plane's normal in end-effector frame. The *Matlab* code for the simulated plane is presented in Appendix A.5.

Limitation of joint angles and joint velocities In order to be sure that no corrupted joint angles were sent to the robot a *Matlab*-function was implemented, see limitedjoints.m in Appendix A.5. This function checks all joint angles to be reasonable and within the robot's reach. If the input joint angles are corrupt then the last setup of correct joint angles are sent to the robot. The function also checks that all joint velocities are within a certain limit. A speed limit of 50 deg/s seemed reasonable.

Simulating a robot Because of the fact that the virtual robot in *Envision* moved instantly to the joint values received from the controller a closed loop with *Envision* was impossible. Instead of full robot dynamics, the PID-controller in the robot system, was modeled as six decoupled first order systems in *Matlab*. The time constants for the joints were identified by sending a step to each of the robot's joints and then logging the movement of the robot. The estimated time constants for the joints are shown in Figure 22. The samples column shows how many samples the robot system needed before the robot had reached 63% of the step height. With a known sample time, 5 ms, the time constant for the different joints were computed.

The simulation To communicate with *Envision* from the *Matlab* controller a network connection was established with a communication package called *Matcomm*. The connection is described in Chapter 4. This connection enables *Envision* commands to be executed from within *Matlab*.
A *Matlab*/*Simulink* controller chart was developed which is designed to

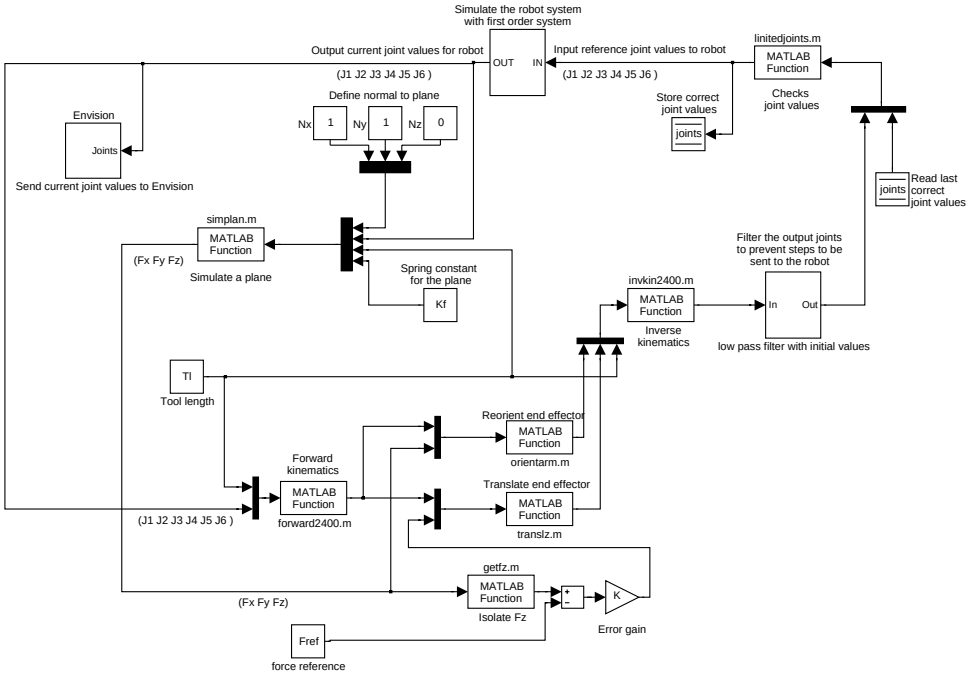

Figure 23 Sixaxes PI force controller for use in simulations

use the real robot system's position controllers, Figure 23. The force controller is a PI-controller. The output from the controller i.e., the input position reference to the robot, should be in joint angles. This simulation version of the controller should send the joint angles to *Envision* after each sample. In order to translate or rotate the end-effector a conversion from joint angles to Cartesian coordinates were needed. The controller uses the forward kinematics described in Chapter 3.1, for this. After this conversion the controller makes the necessary translations and rotations for the end effector depending on the read forces from the simulated plane. The controller translates the end-effector in its z-direction until it intersects with a surface. When a surface is found the controller strives to hold a constant reference force, F_{ref}, acting on the plane. When the end effector is in contact with a surface the controller tries to reorient the end-effector to be normal to the surface. This is done by rotating the end-effector to a orientation where the forces in x- and y- direction are zero. The reorientation of the end-effector only happens if the end-effector is in contact with a plane. Then the TCP-coordinates is converted back into joint values by using the inverse kinematics, Chapter 3.2. These joint values are sent through a low pass filter before they are returned to the robot system as reference joint values. This to prevent large steps in joint values to be sent to the robot.

When running the simulation all robot joints were set to zero except joint four and five which were set to 90 deg. Furthermore a plane was chosen with a plane normal to (1 1 0) in world coordinates and fixed at a position 100 mm in front of the robot in the sensor's z-direction. Moreover the reference force, F_{ref}, was chosen to $-4N$. The simulation result is shown in Figure 26. The simulated forces in x- and z-direction during the simulation are presented in Figure 24. There are no forces in the y-direction

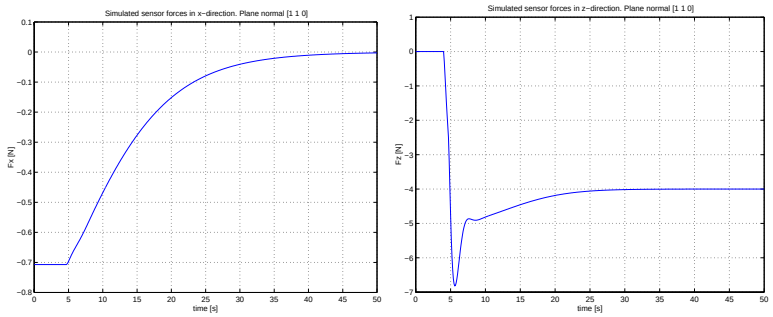

Figure 24 Simulated sensor forces in x- and z-direction.

due to the chosen plane normal.

Experiment Unfortunately some problems occurred when trying to download the controller to the robot system. This problem made it impossible to see if the controller worked in reality. Details about the downloading problem could be found in Chapter 7. A controller based on the successfully performed simulations was developed anyway and it hopefully works in reality. This controller is shown in Appendix B, Figure 34. The controller is based on a force PI-controller with variable proportional part which gives an input to the robot system's built in PID position controllers. Before the joint values are sent to the robot system's PID-controllers they are checked in a *Matlab*-function to be reasonable. This is described in Chapter 6.1. Furthermore the controller is provided with a *Matlab*-function to compensate for the influence of the gravitational force on the end-effector for different joint values. This is described in Chapter 3.4.

6.2 Reorienting while following a trajectory

The next step was to reorient the end-effector normal to a surface while simultaneously following a trajectory. The robot should if not in contact to a surface translate in the end-effector's z-direction during the trajectory until it reached a surface. When overloading with a trajectory a new set of joint values is taken each sample from the trajectory. This change in trajectory joint values acts like a feed forward to the force controller.
This kind of controller would be useful when scanning an object with an unknown or inexact position. By logging the joint values of the robot when in contact with the object, the exact position and the objects surface normal could be obtained. With this kind of information a computer image of the object could be created. Furthermore the information could be used to calibrate a robot system with respect to a robot cell.

Controller By using the main structure of the controller developed in Chapter 6.1, a controller that worked relative to a trajectory was implemented. The controller principle is shown in Figure 27. This controller start out from trajectory joint values for the robot's all six joints, and then adds the output reference joint values from the force PI-controller. This forms a position reference to the robot system's PID-controller. The controller used in the simulations, see Figure 28, simulates the robot system and its PID-controllers with a first order system. All *Matlab* m-files in the

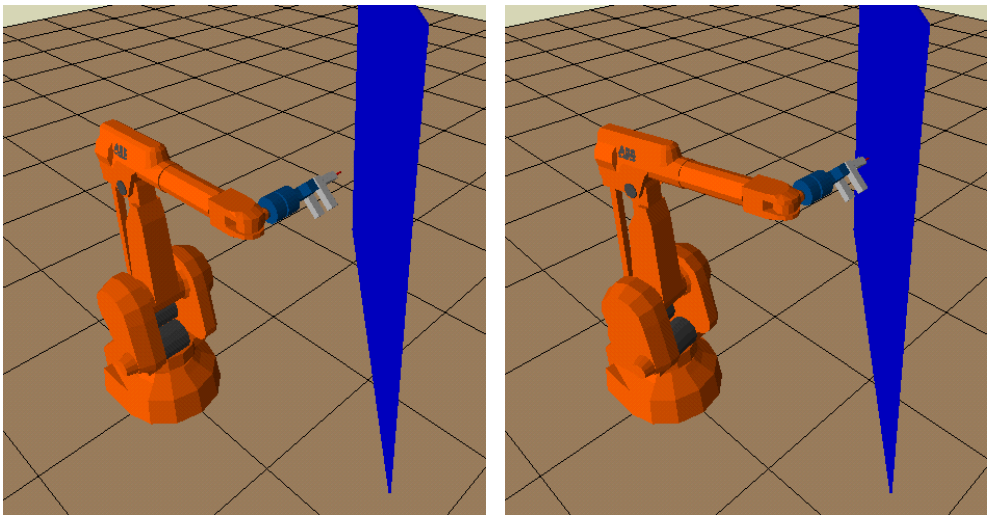

Figure 25 To the left is the robot's initial position. The initial position can be changed to a direction that makes it possible to find the plane. To the right the robot is searching for a plane in the z-direction.

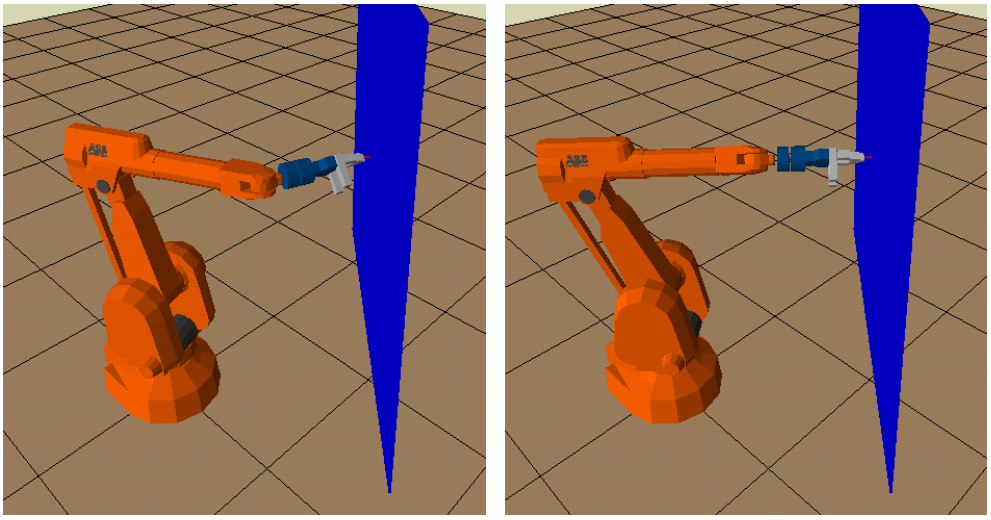

Figure 26 When a force greater than 3 N is detected the robot starts reorientating, left picture. In the picture to the right the robot has found the normal to the plane and stays there.

controller chart is presented in Appendix A.5. The force PI-controller uses an variable proportional part, in the sensors z-direction. This type of controller was developed in Chapter 5.5. In force x- and y-direction an ordinary PI-controller is used.

Simulating a cylinder Instead of the previous simulation with a plane a virtual cylinder was implemented in *Matlab*. The code is presented in Appendix A.5. When using a cylinder instead of a plane the normal in the contact point between the robot and the surface changes depending on where the robot gets in contact with the cylinder. For this reason the robot has to reorient the end-effector more during the trajectory than when using

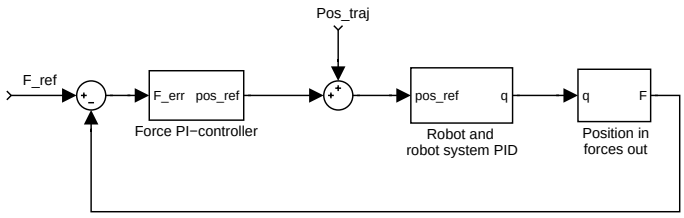

Figure 27 Controller principle

a plane, and thereby it will be easier to see if the controller works properly. The implementation of the cylinder returns forces in x-, y-, and, z-direction which depends on the robot's end-effector orientation and position. Fz is set to zero if the robot is not in contact with the cylinder, and when in contact it is set to be proportial to the distance to the cylinder surface. Fx and Fy are determined by looking at the difference of the cylinder surface normal in the intersection point and the end-effector's z-direction. If these two directions are the same then the end-effector is normal to the surface, which leads to that the forces in x- and y-direction is set to zero.

The simulation To be able to test the controller properly a trajectory had to be created which was chosen to be a circle with a radius of 150 mm. The trajectory should initiate in the robot's TCP when its joint values are [0 0 0 90 90 0] degrees. This position is thereby chosen to be the starting point of the simulation.
In the simulation the cylinder radius was set to 400 mm, and its center was positioned 500 mm in front of the end-effector. Furthermore the cylinder was chosen to be horizontal. With this placement of the cylinder the robot has to translate 100 mm to intersect with the cylinder. The simulation environment can be seen in Figure 29. In addition the force reference, F_{ref} was set to $-4N$. The simulation resulted in plots of the contact forces between the end effector and the cylinder and forces in x- and y-direction. These plots can be seen in Figure 30. The controller strives to keep the forces in x- and y- direction to zero which it accomplished quite well. Forces in z-direction should be zero when not in contact and F_{ref} when in contact. The plot shows that the contact force varies some around F_{ref}, but the error is small. The total time for completing the trajectory is 80 s. When cutting the total trajectory time in half it resulted in more variations around F_{ref} as can be seen in Figure 31. The force error in this plot is bigger than when running the slower trajectory. This concludes that the force error depends on the speed of the trajectory. If a bigger error can be tolerated then a faster trajectory can be overloaded. The graphical part of the simulation made in *Envision* is shown in the Figures 32 and 33 as a sequence. The sequence shows how the robot starts 100 mm from the cylinder surface and starts translating towards it, furthermore the robot follows the predefined trajectory. As soon as the end-effector of the robot reaches the cylinder surface the reorientation begins to keep the end effector normal to the plane. When the robot reaches the last set of trajectory joint values it will only strive to keep a constant force acting on the surface.

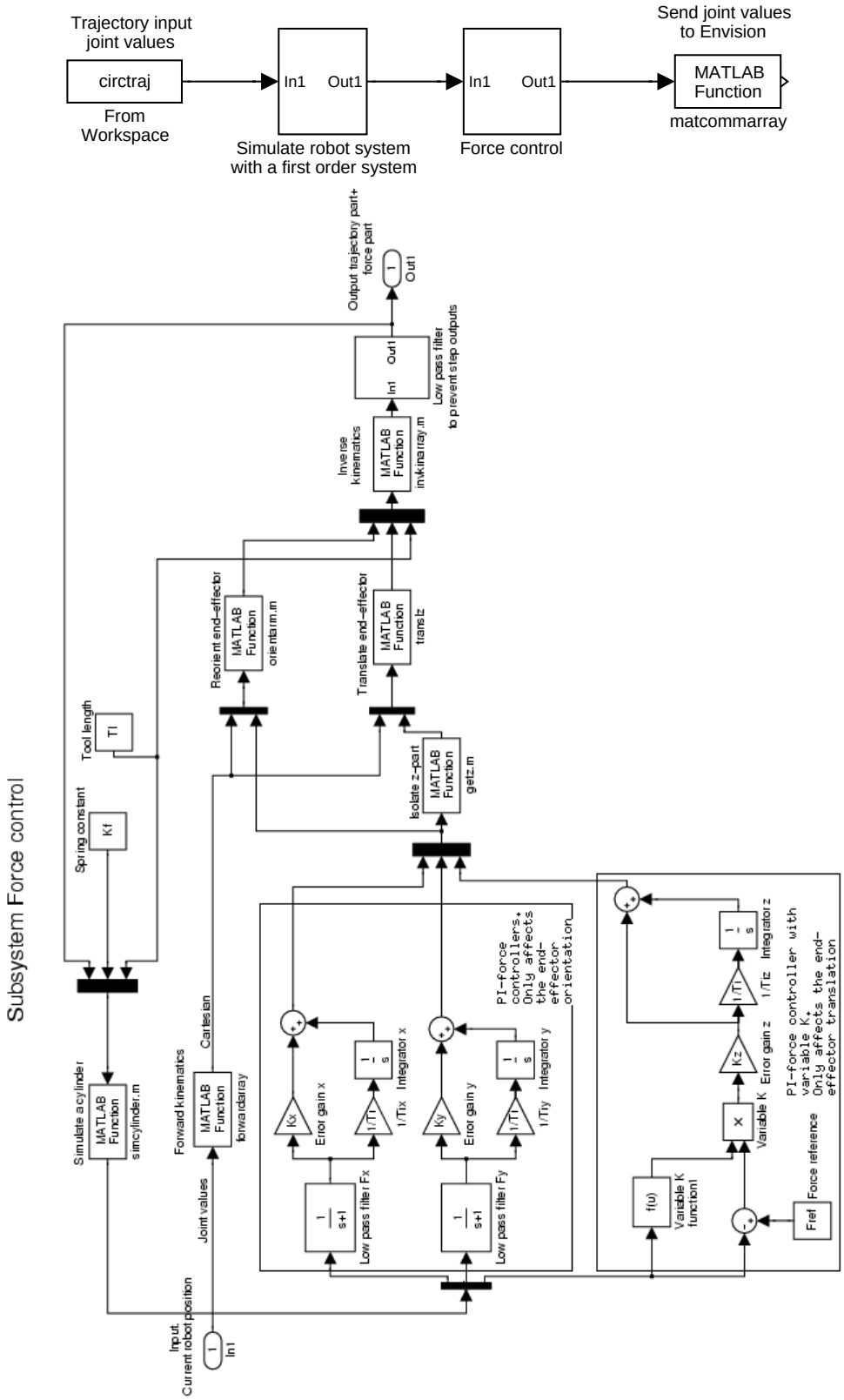

Figure 28 Controller used in the simulations in Section 6.2.

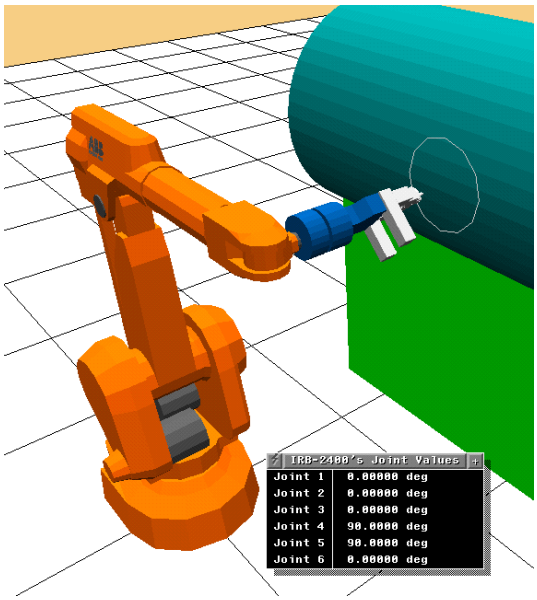

Figure 29 Simulation environment in *Envision*. The robot is in its initial position.

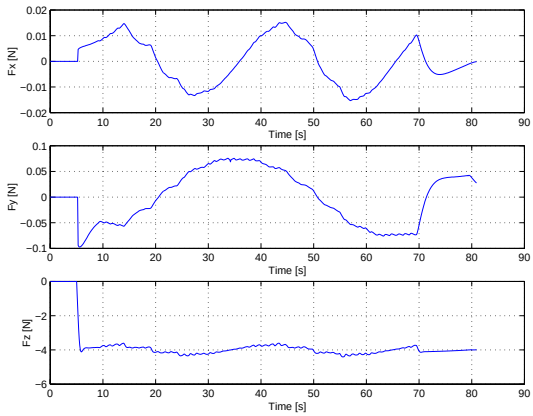

Figure 30 Forces in x-, y-, and, z-direction during the simulation of following a circle on the cylinder.

Experiment The downloading problem described in Chapter 7 resulted in that it was impossible to verify the simulated controller in reality. A controller based on the successfully performed simulations was developed anyway to be used when the code generation problem has been solved. This controller is shown in Appendix B, Figure 35. The trajectory should be downloaded via the *Exc_handler* and is then accessible in the model by *Traj j1* to *Traj j6*, which corresponds to the joint values in the trajectory. Furthermore the controller is equipped with force filters in order to minimize the effect of the noise. The forces are also compensated for the gravitational force acting on the end-effector before they are used in the controller. Before the controller sends a new set of reference values to the robot system a final check is performed to see if the values are reasonable.

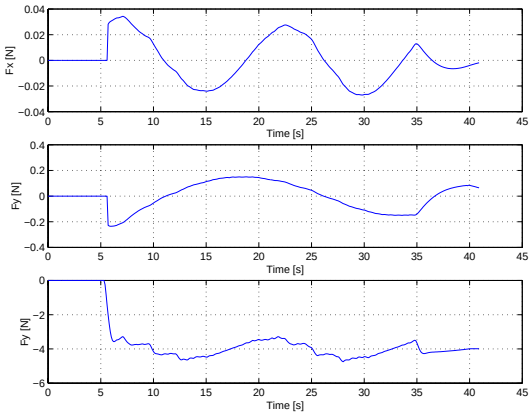

Figure 31 Forces in x-, y-, and, z-direction during a simulation when the speed is the dubble compared with in Figure 30.

6.3 Conclusions

All the simulations worked successfully, both reorienting, Chapter 6.1, and reorienting while following a trajectory, Chapter 6.2.

7. Practical problems

During the work with the master thesis some unexpected problems occurred. The most significant problem was the compiling and downloading of the controller charts developed. All controllers were implemented in *Matlab*/*Simulink*. The interfacing with the robot system was handled by a *Matlab*–toolbox called *Real-Time Workshop*. This toolbox translates the *Matlab*/*Simulink*–charts to *C*–code which then can be compiled and downloaded to the robot system. The most of the source code written was implemented as *Matlab*–functions which was not supported by *Real-Time Workshop*. This should not be a problem to evade by using the *MCC, Matlab to C*/*C++ Compiler*. This compiler compiles *Matlab*–functions to *C*/*C++*–code which should be supported by *Real-Time Workshop* but unfortunatly was not. After extensive research it became clear that this problem was due to an error in *MCC*. According to *Matlab* this error will be corrected in future versions of the compiler. The *MCC* used was Version 2.0.1.

8. Conclusions and recommendations

When only controlling on three joints a PI force controller with variable proportional part is to be preferred. This because it can be tuned to be very fast when in contact and thereby it can apply a constant force on an object without large force errors. The simulated result did not completely agree with the result achieved when the experiment was done in the reality. The reason is probably the assumption in the simulation that the force signal was without noise. From the measurements during the force control, it is possible to update and calibrate the simulation model with respect to the real work-cell, as for instance measuring the angle or the curvature

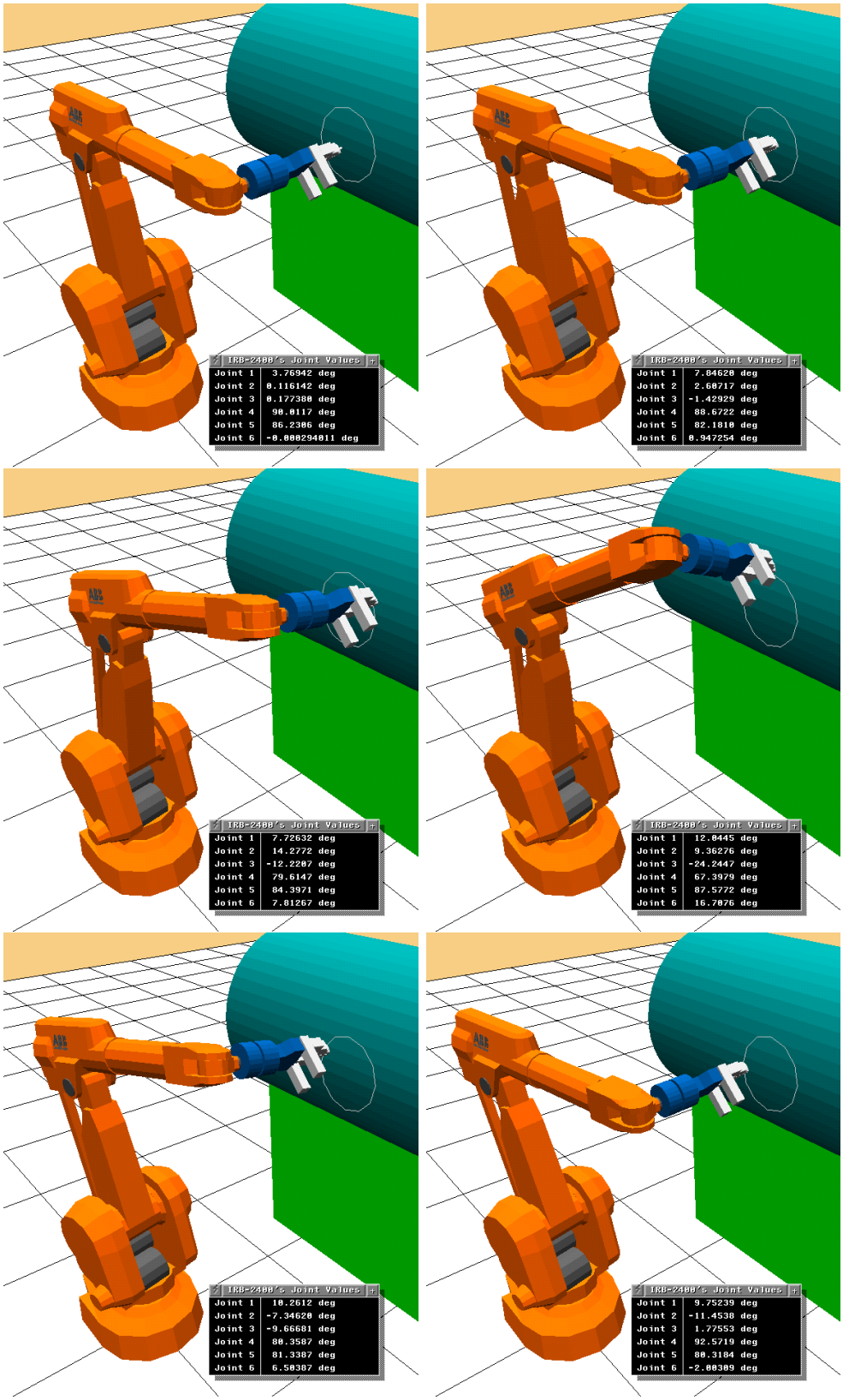

Figure 32 Simulation sequence of force control on a cylinder during a trajectory.

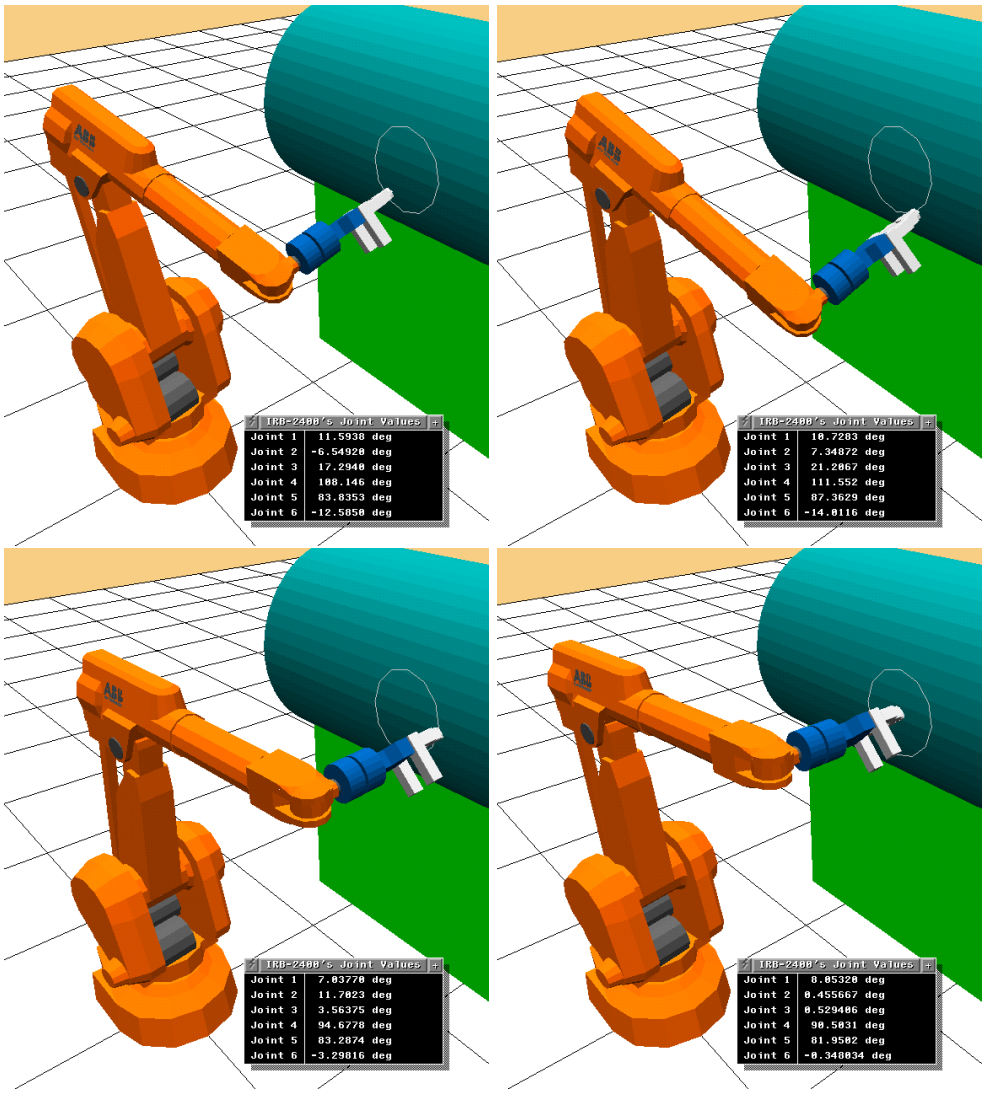

Figure 33 Simulation sequence of force control on a cylinder during a trajectory.

of a surface. All the simulations using six joints worked successfully. It was possible to reorient the gripper to be perpendicular to the surface meanwhile following a trajectory. In the simulations it was possible to identify parts of simpler surfaces such as planes, cylinders, spheres.

9. Future work

The main future expansion is to get the controllers that control all six axes to work on the real robot. This can not be done until *Mathworks* releases a new version of the *MCC, Matlab to C/C++ Compiler*, that is a newer version than 2.0.1, or until all *Matlab*–code is translated to *C*–code manually. However, since *Matlab*-toolboxes and *Matlab* matrix manipulation commands which are not present in the *C*–language is frequently used in the developed software, a manual translation to *C* is not reasonable.
The controller charts developed with a direct force controller can also be

developed further by controlling both the joint positions and joint velocities.

10. References

ABB Robotics (1991): *Product Manual IRB 2000*.

Blomdell, A. (1997): "A real time control language for the Pålsjö environment.". Master's thesis ISRN LUTFD2/TFRT--5578--SE, Department of Automatic Control, Lund Institute of Technology.

Bolmsjö, G. and M. Olsson (1999): *Robot Technology, Exercise 1-3, UARC*. Division of Robotics, Dept. of Mechanical Engineering, Lund University.

Craig, J. J. (1989): *Introduction to Robotics*. Addison-Wesley Publishing Company.

CRS (2000): "CRS robotics." http://www.crsrobotics.com/markets/mkt_intro.html.

Denavit, J. and R. S. Hartenberg (1955): "A kinematic notation for lower-pair mechanisms based on matrices." *Journal of Applied Mechanics*, pp. 215–221.

Deneb Robotics, Inc. (2000): "Deneb Robotics, Inc homepage." http://www.deneb.com/.

Ekdahl, I. (1999): *Mätning av icke elektriska storheter*. Department of Industrial Electrical Engineering and Automation, Lund Institute of Technology, Lund, Sweden.

Eker, J. (1997): *A Framework for Dynamically Configurable Embedded Controllers*. Licenciate Thesis ISRN LUTFD2/TFRT--3218--SE, Department of Automatic Control, Lund Institute of Technology.

Eker, J. (1999): *Flexible Embedded Control Systems. Design and Implementation*. PhD thesis ISRN LUTFD2/TFRT--1055--SE, Department of Automatic Control, Lund Institute of Technology.

ENVISION (2000): "ENVISION Online Documentation." file:/usr/deneb/vmap/docs/envision_HOME/HOMEPAGE.html.

Hogan, N. (1985): "Impedance control: An approach to manipulation, Parts I-III." *Journal of Dynamic Systems, Measurement, and Control ASME*, **107**, pp. 1–24.

Johansson, R. and M. Spong (1994): "Quadratic optimization of impedance control." *In Proc. 1994 IEEE Int. Conf. Robotics and Automation*, pp. 616–621.

Liljenborg, S. and A. Olsson (2000): "Force controlled robots in industrial automation." Technical Report. Department of Industrial Electrical Engineering and Automation (IEA).

Mathworks, Inc. (2000): "Mathworks, Inc. homepage." http://www.mathworks.com/.

Nasri, H. (1999): *Modelling and requirements of the automated deburring process*. PhD thesis, Division of Robotics, Department of Mechanical Engineering, Lund University, Lund.

Nilsson, K. (1996): *Industrial Robot Programming*. PhD thesis ISRN LUTFD2/TFRT--1046--SE, Department of Automatic Control Lund Institute of Technology.

Olsson, M. (2000): "How to extend Vmap functionality." Technical Report. Division of Robotics, Dept. of Mechanical Engineering, Lund University.

Shutter, J. D., H. Bruyninckx, W.-H. Zhu, and M. W. Spong (1997): "Force Control: A Bird's Eye View." In Siciliano and Valavanis, Eds., *Control Problems in Robotics and Automation*, number 230 in Lecture Notes in Control and Information Sciences. Spinger-Verlag, London, UK.

Siciliano, B. and L. Villani (1999): *Robot Force Control*. Kluwer Academic Publishers.

Spong, M. W. and M. Vidyasagar (1989): *Robot Dynamics and Control*. John Wiley & Sons, Inc.

Whitney, D. E. (1987): "Historical Perspective and State of the Art in Robot force Control." *The International Journal of Robotics Research*, **6**. Massachusetts Institute of Technology.

Zeng, G. and A. Hemami (1997): "An overview of robot force control." *Robotica*, **15**. Cambridge University Press, Cambridge.

A. Program lists

A.1 Forward

```
function  result=T01(j1)
% function  result=T01(j1);
% Calculates the transformation matrix from frame 0 to 1 for
%joint angle j1;
result=[cos(j1)  0  -sin(j1)  0 ;sin(j1)  0  cos(j1)  0 ;
0  -1  0 750 ; 0  0  0  1];
```

A.2 Jacobian

```
function  result=jacobian0S(joints)
% function  result=jacobian0S(joints);
% Calculates the force jacobian from frame 0 to frame S(ensor);
t0s=T0S(joints);
px=t0s(1,4);
py=t0s(2,4);
pz=t0s(3,4);
p=[0 -pz py;pz 0 -px;-py px 0];
r0s=t0s(1:3,1:3);
J11=r0s;
J21=p*r0s;
J12=zeros(3);
J22=r0s;
Jupper=cat(2,J11,J12);
Jlower=cat(2,J21,J22);
result=cat(1,Jupper,Jlower);
```

A.3 Gravity compensation

```
function result=gravcomp(joints,FM,mg);
% function result=gravcomp(joints,FM,mg);
% function result=gravcomp(joints,FM);
% This function requires that the force sensor is reset when
% fz=mg;
% FM=[fx fy fz mx my mz] in sensor frame;
% mg is optional (default=70) = sensor weight*g (ca 70);

if nargin ==2
   mg=70;
end;

% Make force coord. syst. right oriented (left oriented from
% the start);
FM(2)=-FM(2);
FM(5)=-FM(5);
%add mg to fz (because the sensor is zeroed in this direction);
FM(3)=FM(3)+mg;
% Convert from Ndm to Nmm;
FM(4)=FM(4)*100;
FM(5)=FM(5)*100;
FM(6)=FM(6)*100;

% Force to prevent grav. force;
Fgrav=[0 0 mg 0 0 0];
Ftp=inv(jacobianTP0TP(joints))*Fgrav';
Fcomp=jacobian0TP(joints)*Ftp;
FS=inv(jacobian0S(joints))*Fcomp;
FM=FM+FS';

% Make force coord. syst. left oriented again;
FM(2)=-FM(2);
FM(5)=-FM(5);
% Convert from Nmm to Ndm;
FM(4)=FM(4)/100;
FM(5)=FM(5)/100;
FM(6)=FM(6)/100;
result=FM;
```

A.4 Communication between Matlab and Envision

runCLI

```
function runCli(clistring,id);
% function runCli(clistring,id);
%
% id is the communication line obtaind when starting the
% communication with Envision.
% id=matcomm('server','matlabc');
% Example:
% runCli('PAN CAMERA RIGHT 20 IN 20',id)
```

```
%
matcomm(id,clistring);
```

runLLTI

```
function runLLTI(LLTIarray,id);
% function runLLTI(LLTIarray,id);
%
% id is the communication line obtaind when starting the
% communication with Envision.
% id=matcomm('server','matlabc');
% Example
% runLLTI([10.0 6 0 0 0 0 0 0],id)
% This cmd sets all the six joints on the robot to 0.
matcomm(id,LLTIarray);
```

matenv

```
/*Functions which makes it possible to communicate with Envision
           from Matlab via LLTI (Low Level Tele Robotic)*/

#include <stdio.h>
#include <string.h>
#include <shlibdefs.h>
#include "libmatcomm.h"
static MatCommLine *mcLine;
static int i=100000;
static char  cliCommand[144];
static float packet[144];
/*static char charpacket[144];*/

/*-------------------------IRIX-----------------------------*/
/*-------------Error handling functions---------------------*/
void error(char *message)
{
  fprintf(stderr, ": ERROR: %s\n", message);
  exit(1);
}

void warning(char *message)
{
  fprintf(stderr, ": WARNING: %s\n",  message);
}

/*--------------------Setup connection----------------------*/
void
llti_init_1()
{
  int errcode;
  /* char n="rune"; */
  char *name="rune";
  printf("Initializing\n");
  /*satt treje parametern till tio med nyare matcomm*/
```

```
  mcLine=MatCommOpen("bellman.control.lth.se","matlabc", 1000);
  if(!mcLine)
     printf("llti_init_1: Can't initialise client socket\n");
  else
    printf("Connection established\n");

 /*  name=n; */
  errcode=msg_pop_create( name );
  return;
}

int readfloat(int nrows,int ncols){
  int i;
  int j;
  float floatdata[nrows][ncols];
  MatCommReadFloat(mcLine, floatdata, nrows, ncols);
  for (i = 0 ; i < nrows ; i++) {
    for (j = 0 ; j < ncols ; j++) {
      printf("%f ", floatdata[i][j]);
      packet[j]=floatdata[i][j];
    };
    printf("\n");
  };
  free(floatdata);
  printf("Recevied data in readFloat\n");

  return( 1 );
}

int readchar(int nrows,int ncols){
  int j=0;
  char *chardata = (char*) malloc(sizeof(char)*nrows*ncols+1);
  if(MatCommReadChar(mcLine, chardata,nrows,ncols)==0){
    printf("Failed to receive string\n");
  }
  else{
    printf("Recevied string= ");
    chardata[sizeof(char)*nrows*ncols]='\0';
    printf(chardata);

    j=0;
   /*Save the text string direct int the static var. cliCommand*/
    while(chardata[j]!=0){
      cliCommand[j]=chardata[j];
      j++;
    }
    cliCommand[j]='\0';
  }
  chardata="\0";
  free(chardata);
  return (1);
}
```

```
void sendCliCommand(){
  int errcode= cli ( cliCommand );
}
/*----------------Read from communication line---------------*/

float *llti_read_1(){
  int nrows;
  int ncols;
  int datarec=0;
  int j=0;
  int ecode;
  int yy;
  int slength;
 /*  float *packetPointer; */
/*   char *cliCommandPointer; */
  i++;
  if(i>100000){
    printf("The program has now scanned the communication line
             for data 100 000 times since last message\n ");
    i=0;
  }

  if(MatCommDataAvailable(mcLine)){
    if (MatCommIsReal(mcLine, &nrows, &ncols)){
      ecode=readfloat(nrows,ncols);
      j=0;
      /*The next statement will only write none zero floats*/
      while(packet[j]!=0){
        printf("Packet to return to Envision= %f ", packet[j]);
        printf("\n");
        j++;
      }
      return packet;
    }
    else{
      if(MatCommIsChar(mcLine, &nrows, &ncols)){
        ecode=readchar(nrows,ncols);

        j=0;
        slength = strlen(cliCommand);
        printf("%s length: %i\n",cliCommand,slength);
        packet[0]= 130.0;
        packet[1]= slength;
        memcpy(&packet[2],cliCommand,slength);
        /*Return packet to Envsion*/
        return packet;
      }
    }
  }
  else
    return(0);
```

```
}


/*--------------------Close connection function---------------*/
void llti_close_1()
{
  printf("Closing down connection\n");
  MatCommClose(mcLine);
  printf("Connection closed\n");
  return;
}

/*--------------------Test main program-----------------------*/

/* int main(){ */
/*   float* mainpacket; */
/*   int y; */
/*   int t; */
/*   printf("Nu i main\n"); */
/*   llti_init_1(); */

/*   while(1){ */
/*     mainpacket=llti_read_1(); */
/*     for (y = 0 ; y < 2 ; y++) { */
/*       printf("main %f ", mainpacket[y]); */
/*       printf("\n"); */
/*     };  */
/*     sleep(1); */
/*   } */


/* } */


/*------------------------IRIX-------------------------------*/
```

A.5 Six joint control

orientarm

```
function result = orientarm(par);
% function result = orientarm(par);
% par(1:12 )=[T(1,1:4) TT(2,1:4) T(3,1:4)]
% where T is the transformer matrix.
% par(13:18)=[Fx Fy Fz Mx My Mz]
% The function reorients the endeffector if the contact force
% (Fz) is 2N<Fz<8N.
T=[par(1:4)';par(5:8)';par(9:12)'; 0 0 0 1];
FM=par(13:18)';
%Make sensor coordintae system right oriented
FM(3)=-FM(3);
%Define rotation angles
```

```
if FM(3)>2 & FM(3)<8
        alfa=atan(FM(1)/FM(3));
        beta=atan(FM(2)/FM(3));
else
        alfa=0;
        beta=0;
end
%Rotate the transformer matrix
TT=T*roty(alfa)*rotx(-beta);
%Return the new reoriented transformer matrix as an array
result=[TT(1,1:3) TT(2,1:3) TT(3,1:3)];
```

translz

```
function result=translz(par);
% function result=translz(par);
% par(1:12 )=[T(1,1:4) TT(2,1:4) T(3,1:4)]
% where T is the transformer matrix.
% par(13)=The distance the endeffector should be translated in
%its z-direction.
T=[par(1:4)';par(5:8)';par(9:12)'; 0 0 0 1];
lz=par(13);
TT=T*transl([0 0 lz]);
%Return the new translated transformer matrix as an array
result=[TT(1,4) TT(2,4) TT(3,4)];
```

invkinarray

```
function result=invkinarray(par);
T=[par(1:3)' par(10); par(4:6)' par(11);
   par(7:9)' par(12); 0 0 0 1];
toollength=par(13);
result=invkin2400(T,1,1,toollength);
```

forwardarray

```
function result=forwardarray(par);
joints=par(2:7);
tool_length=par(1);
T=forward2400(joints, tool_length);
result=[T(1,1:4) T(2,1:4) T(3,1:4)];
```

simplan

```
function result=simplan(par)
% function result=simplan(par)
% par(1:3)=[Nx Ny Nz]. The disired plane's normal array.
% par(4:9)=[j1 j2 j3 j4 j5 j6]. Robot joint angles.
% par(10)=The end effector tool length .
% par(11)=The spring constant for the plane.
%
% This function simulates a plane choosen by its normal array
% and a fix point. The function returns forces in  x-, y-, and
% z-direction depending the robot's position relative to the
```

```
% defined plane.
    par=par';
    Kf=par(11);
    joints=par(4:9);
    toollength=par(10);

%****************************************
% Define a plane in coord syst 0
    Nsurf0=unit(par(1:3));
% Define plane point as the startingpoition of the robot +100 in
% x- and y-direction
    tmppos=forward2400([0 0 0 pi/2 pi/2 0],toollength);
    xref=tmppos(1,4)+100;
    yref=tmppos(2,4)+100;
    zref=tmppos(3,4);
% Write plane on kx*X+ky*Y+kz*Z+d=0;
    kx=Nsurf0(1);
    ky=Nsurf0(2);
    kz=Nsurf0(3);
    d=-(kx*xref+ky*yref+kz*zref);
%****************************************
%******************FZ*********************
%End effector origin coords in world coord syst.
    T=forward2400(joints,toollength);
    x=T(1,4);
    y=T(2,4);
    z=T(3,4);
%End effector z-direction in world coord syst.
    N=[T(1,3) T(2,3) T(3,3)];
    t=(-d-kx*x-ky*y-kz*z)/(kx*N(1)+ky*N(2)+kz*N(3));
%The line (in zeffector direction) cuts the plane in (px,py,pz)
    px=x+N(1)*t;
    py=y+N(2)*t;
    pz=z+N(3)*t;
%distance from end effector origin to (px,py,pz)
% in the world coordinate systen
    dist0=[px-x py-y pz-z 0 0 0];
%distance from end effector origin to (px,py,pz) in end effector
%coord. syst
    dist6=inv(jacobian06(joints))*dist0';
    if dist6(3)<0
      Fz=Kf*dist6(3);
    else
      Fz=0;
    end

%****************************************

%******************FX AND FY***************
    Nsurf0=[Nsurf0 0 0 0];
    Nsurf6=inv(jacobian06(joints))*Nsurf0';%stämmer;
% Plane in cord syst 6
```

```
    Nsurf6=inv(jacobian06(joints))*Nsurf0';%stämmer;
    F=[Nsurf6(1) Nsurf6(2) Fz];
%****************************************
% Return the simulated forces [Fx Fy Fz]
result=[F(1:3)];
```

limitedjoints

```
function result=limitedjoints(par)
% If the joint values moves out of the specifed limit, the
% outvalues will be the last valueble jointvalues.
j1=par(1);
j2=par(2);
j3=par(3);
j4=par(4);
j5=par(5);
j6=par(6);
oldj1=par(7);
oldj2=par(8);
oldj3=par(9);
oldj4=par(10);
oldj5=par(11);
oldj6=par(12);
gr2rad=pi/180;
maxv=50*gr2rad; % [rad/sec];
h=0.005;   % change this value when you change sampletime;

if (j1 > -150*gr2rad)&(j1 < 150*gr2rad)&((j1-oldj1)/h < maxv)
  j1OK=1;
else
  j1OK=0;
end

if (j2 > -60*gr2rad)&(j2 < 60*gr2rad)&((j2-oldj2)/h < maxv)
  j2OK=1;
else
  j2OK=0;
end

if (j2+j3 >-60*gr2rad)&(j2+j3< 60*gr2rad)&((j3-oldj3)/h < maxv)
  j3OK=1;
else
  j3OK=0;
end

if (j4 > -180*gr2rad)&(j4 < 180*gr2rad)&((j4-oldj4)/h < maxv)
  j4OK=1;
else
  j4OK=0;
end

if (j5 > -100*gr2rad)&(j5 < 100*gr2rad)&((j5-oldj5)/h < maxv)
```

```
  j5OK=1;
else
  j5OK=0;
end

if (j6 > -200*gr2rad)&(j6< 200*gr2rad)&((j6-oldj6)/h< maxv)
  j6OK=1;
else
  j6OK=0;
end

if j1OK*j2OK*j3OK*j4OK*j5OK*j6OK > 0.5
  result=[j1 j2 j3 j4 j5 j6]';
else
  result=[oldj1 oldj2 oldj3 oldj4 oldj5 oldj6];

end
```

getfz

```
function result=getfz(FM);
% function result=getfz(FM);
% FM=[Fx Fy Fz Mx My Mz]
%
% This function returns the z-component of the force-torque array.
result=FM(3);
```

B. Controller charts

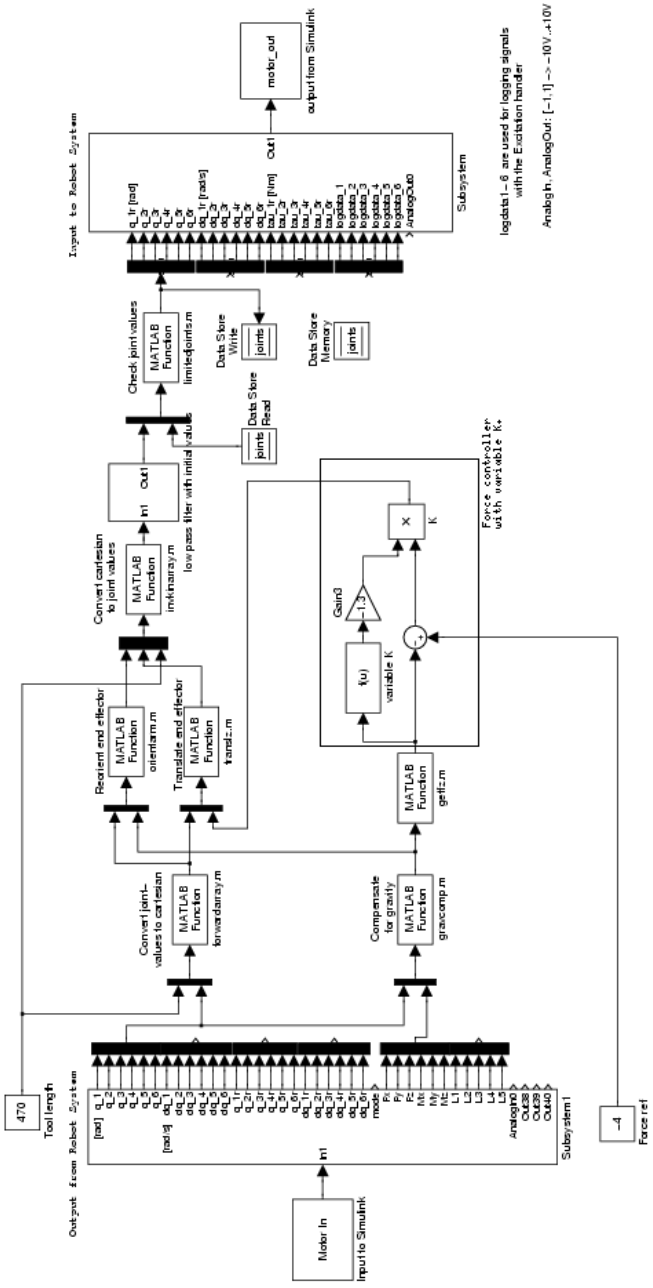

Figure 34 Controller developed for the robot system for reorienting to be normal to a surface.

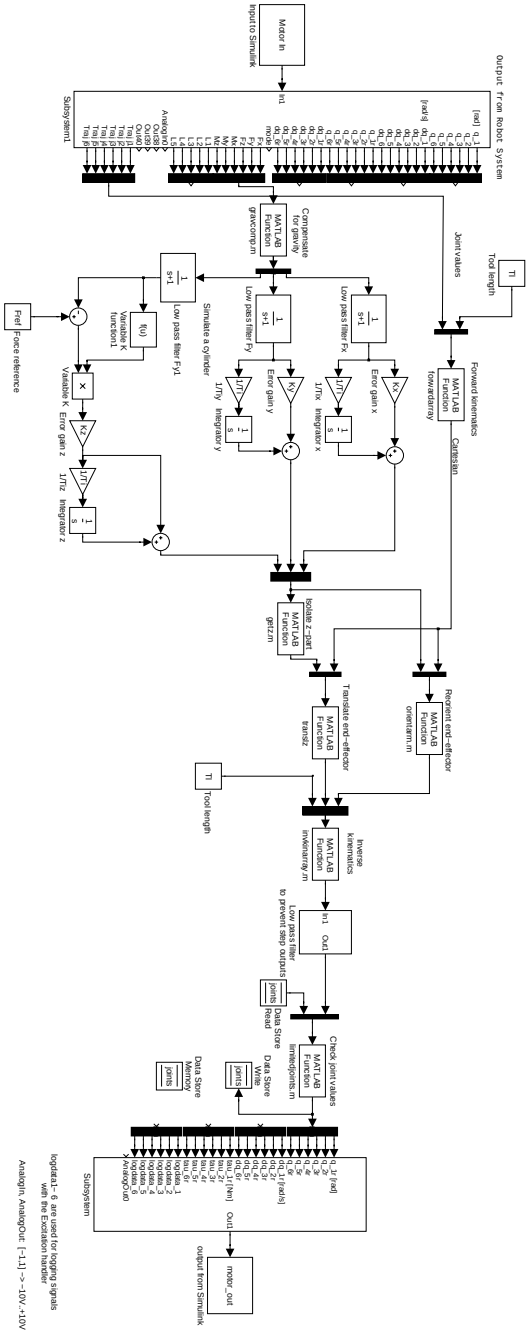

Figure 35 Controller developed for the robot system when reorienting and following a trajectory.

C. Robot drawings

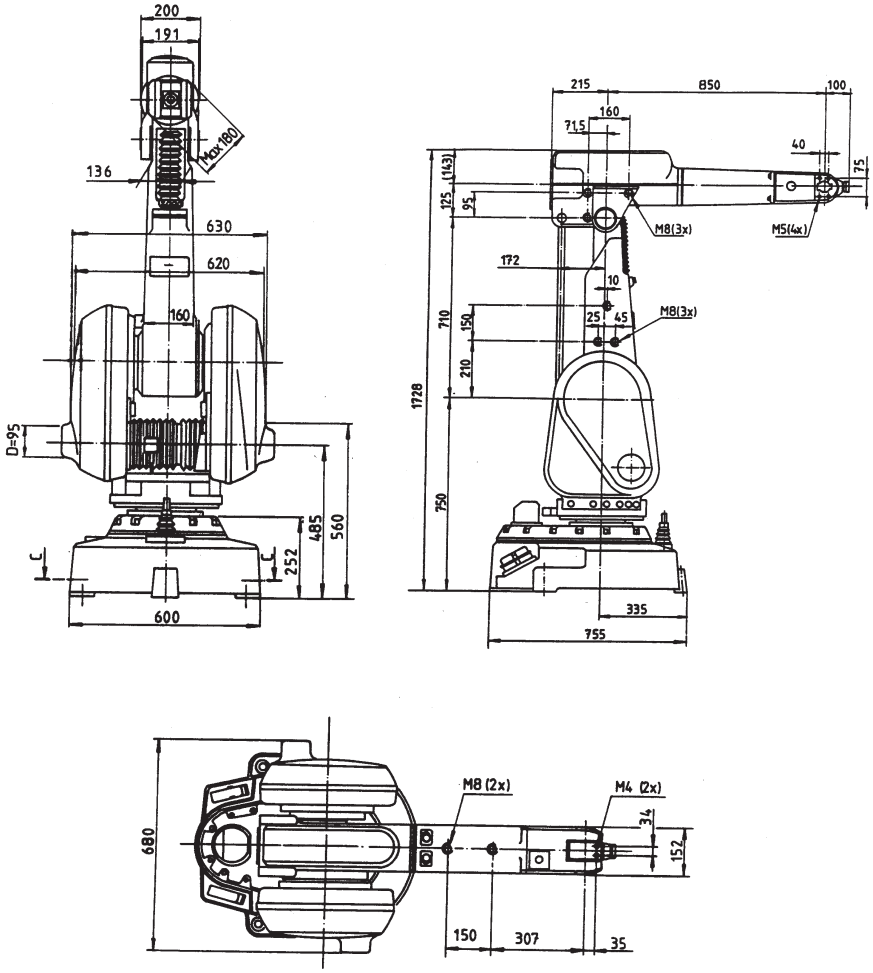

Figure 36 The dimensional drawings of the robot [ABB Robotics, 1991].