

ISSN 0280-5316
ISRN LUTFD2/TFRT--5863--SE

From CAD-design to force controlled robot manufacturing

Ana Grau Torres

Department of Automatic Control
Lund University
September 2010

Lund University Department of Automatic Control Box 118 SE-221 00 Lund Sweden		<i>Document name</i> MASTER THESIS	
		<i>Date of issue</i> September 2010	
		<i>Document Number</i> ISRN LUTFD2/TFRT--5863--SE	
<i>Author(s)</i> Ana Grau Torres		<i>Supervisor</i> Anders Robertsson Automatic Control, Lund Rolf Johansson Automatic Control, Lund (Examiner)	
		<i>Sponsoring organization</i>	
<i>Title and subtitle</i> From CAD-design to force controlled robot manufacturing (Från CAD-ritning till tillverkning med kraftreglerad robotstyrning)			
<i>Abstract</i> <p>This Master Thesis presents the necessary steps to manufacture a piece, starting from the draw in the software Pro/ENGINEER and then processing the generated G-code to create a RAPID program understandable for the robot. Some additions required to make the code work on the Teach Pendant of the IRB 2400-16 are explained in detail. Apart from the RAPID program, the robot is also controlled with the help of a Simulink controller created to avoid collisions and to assure a compliant behaviour with the environment. An extra application is included in order to obtain information about the shape of the workpiece that the robot will machine.</p>			
<i>Keywords</i>			
<i>Classification system and/or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i> 0280-5316			<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 76	<i>Recipient's notes</i>	
<i>Security classification</i>			

Acknowledgements

First of all, I would like to thank to my supervisor Anders Robertsson the great opportunity he gave to come to Lund University to do this Master Thesis. I really appreciate his help and prompt reply to all my questions despite his busy daily routine. Working in such an amazing place has been an incredible experience.

Special mention to my family for believing in me during all this time and never giving up. To my sister Alicia and my parents because they were able to see my potential and always kept encouraging me to get farther. They have been a huge support during these years even when I was discouraged. I know hearing my stories and complaints about faculty issues is not easy but they were there for me.

Of course, I am very grateful to all my friends who help me when the lack of ideas attacks or just when I need a break. Thanks to: Vicente, Rubén, José, Jorge, Jana, Pablo and all the crew from the ETSII.

It has been a long journey but now it is getting to an end. I have learnt a lot and I expect to learn much more in the future. The satisfaction of a well done work is irreplaceable.

Contents

1. Introduction.....	11
1.1 Background.....	11
1.2 Problem formulation.....	11
1.3 Outline.....	11
2. Tools and scenario.....	12
2.1 Hardware tools.....	12
2.2 Software tools.....	15
2.3 Interconnection between software and hardware elements.....	16
3. Theoretical base.....	18
3.1 Homogeneous transformations.....	18
3.2 Joint and frames description.....	19
3.3 Forward kinematics.....	20
3.4 Inverse kinematics.....	21
3.5 Jacobian (differential kinematics).....	22
3.6 Force control.....	23
4. Pro/ENGINEER.....	27
5. Postprocessor.....	28
5.1 Postprocessor output: Corrections and notes.....	28
6. RobotStudio.....	30
6.1 Program structure.....	30
6.2 Variable details.....	31
6.3 Variable definition.....	32
7. Robot control: ideas and basis.....	34
7.1 Gravity compensation.....	34
7.2 Impedance controller.....	39
7.3 Contact algorithm.....	39
7.4 Machining algorithm.....	40
8. Robot control: practical tests.....	41
8.1 Contact program.....	41
8.2 Contact algorithm results.....	46
8.3 Machining task.....	48
8.4 Machining algorithm results.....	51
9. Conclusion.....	54
References.....	56

Appendix A: Pro/ENGINEER guide.....	57
A.1 Creating the file and adding the model.....	58
A.2 Creating the workpiece, milling volume or milling window.....	59
A.3 Operation setup.....	62
A.4 Machining sequence:.....	63
A.5 Tool path and NC code.....	66
Appendix B: State machine.....	68
Appendix C: Basic velocity.....	70
Appendix D: Surf.m.....	71
Appendix E: M_adapt.....	73
Appendix F: Integral draining.....	74

1. Introduction

1.1 Background

In today's industry there are many important factors that make a difference between companies. Quality, costs and production time have to be taken into account when it comes to plant scheduling and all of them should be balanced to obtain the best results.

One aspect that can help to improve the previous parameters is automation. By introducing automation elements in the factory time is drastically reduced, costs come down because of the production increase and quality reaches more homogeneous and controllable values. On the other hand, in many cases, replacing human operators with automatic devices avoids accidents and tedious jobs for the worker.

Robotic arms are more and more used in order to achieve these targets. They can carry out many different tasks simply by changing the tool attached. This Master Thesis deals with all these ideas. Machining pieces with the help of robots as an alternative to CNC machines (Computed Numerically Controlled machines) could lead to cost reductions and more flexible processes.

1.2 Problem formulation

The main objective in this Thesis is to develop a protocol that allows to integrate the whole chain of production, from design to final manufacture so the robot can create pieces or components starting from a drawing generated using Pro/ENGINEER. Then it would be possible to obtain the code for a CNC machine and postprocess it with the software Winpost to get a RAPID program.

In order to make the process more complete the robot must be able to find the workspace where the raw piece or workpiece is located and then, carry out the necessary movements as specified in the RAPID code.

1.3 Outline

In the first chapters some theoretical aspects about the robot and its dynamics are presented. A brief explanation to force controllers is also included due to its importance in this Thesis.

A short chapter about Pro/ENGINEER is introduced to show its relevance in the project, and a more detailed manual about this program is included in the first Appendix with the necessary steps to begin the design process.

After that a couple of chapters explain how to convert the code coming from Pro/ENGINEER into something understandable for the robot.

Finally, the main controllers for the practical tasks are presented and the conclusions from the experiments are shown.

2. Tools and scenario

All the work developed during this Thesis took place in the Robotic Lab at the Department of Automatic Control at LTH, Lund University.

2.1 Hardware tools

Robot

Nowadays there are many kinds of industrial robots available in the market. They can be sorted following different criteria [1]. Robots are built with a certain number of solid links connected by joints. Concerning joints two main distinctions can be found:

- Revolute joints when they rotate around an axis coincident with the link.
- Prismatic joints when they move along the link.

Other types like spherical or cylindrical joints can also be used but they are adapted to more specific requirements.

The number of joints is also a typical criterion for the classification and it usually defines the number of degrees of freedom (DOF) of the robot. Many robots just use three joints because they are designed for very concrete jobs. For instance, SCARA robots have two revolute joints and one prismatic joint and they can be found in pick and place tasks because of their simplicity.

Now it is also very common to work in pick and place jobs with parallel kinematic robots, also known as delta robots. They are built as a closed kinematic chain with some rigid links hanging from a base where the motors are located and connected in the last joint. These robots are really fast and precise but the space reached is small and their stiffness is low.



Figure 1: SCARA and parallel kinematic robots

Anthropomorphic or robotic arms are widely found in industry. They consist of three revolute joints with three extra joints in the end of the robot, in this way more positions and orientations can be reached.

During this thesis the main tool used was the ABB [2] robot manipulator IRB 2400-16 like the one shown in Figure 2. Thanks to its six revolute joints this robotic arm has six degrees of freedom which means that every position of the end effector can be described with six independent coordinates (three for the position

and three for the orientation), the workspace or volume of space reached is bigger and there is more than one way to get to some points [3].



Figure 2: IRB 2400-16 and S4CPlus control cabinet with the Teach Pendant produced by ABB

This kind of robot in particular is ready to work in many different environments or tasks just by changing the tool attached to its wrist. For the present work some requirements have to be fulfilled:

- The machining tasks, depending on the material, could need high strength and robustness.
- Precision and repeatability are absolutely necessary in order to obtain a proper final piece.
- High velocity is highly recommended to make the process as much optimal as possible.

Other robots could also work if they meet the previous requisites but some of the requested characteristics make inappropriate the use of parallel robots or with less than six DOF.

S4CPlus cabinet

The robot is controlled with the help of the S4CPlus and its Teach Pendant, see Figure 2. This computer performs the emergency stops when the required velocity is too high and supervises the right behaviour of the different joints in the robot. It also communicates the robot with the external computer that acts as controller with programs created by the user, providing all the data required: joint positions, velocities, accelerations, etc.



Figure 3: Teach Pendant

The Teach Pendant that appears in Figure 3 is the user's console. RAPID programs can be uploaded and reviewed there. It also allows the user to jog the robot manually with its joystick. The dead man's switch is included there, it has to be pressed to release the brakes and start the movement, if the behaviour of the robot is not the expected the brakes can be activated by pressing deeper or releasing this button. A screen informs about different data coming from the robot and it is also used to configure some parameters.

Force Sensor

The robot by itself is unable to feel the environment. Once it is programmed with one path it will repeat it even when an obstacle is placed in the middle of the trajectory. In the same way, if the task involves contact the robot needs a complement to let it know how hard it is touching the workpiece.

In order to obtain this information a force sensor is attached to the wrist of the robot. This is only valid to know the force exerted on the tool so the environment should be really controlled to avoid accidents.



Figure 4: JR3 force sensor with the tool holder attached to the wrist of the IRB 2400-16 in the Robotic Lab

For this Thesis a JR3 100M40 I63 force sensor is used [4]. It provides measurements of force in [N] and torque in [Nmm] in the three axes of the sensor frame thanks to the foil strain gauges that it contains. A measure coming from the sensor is expressed as:

$$F = \begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

The sensor does not make any distinction between the origin of the force (tool weight, contact force, inertia due to the movement, etc) so the measurements have to be corrected. This will be explained further ahead in the text.

2.2 Software tools

The main software necessary for this Thesis is now briefly described:

- Pro/ENGINEER [5]: It is a CAD/CAM/CAE program with a 3D interface that helps in the design of accurate industrial devices. It can produce different outputs starting from the most simple scheme to advanced animations containing the whole movement of a complex device. It also can generate the G-code for a specific CNC machine with the necessary movements, velocities, positions, etc.

- Winpost: It is a really simple application that converts the file containing the G-code coming from Pro/ENGINEER into a RAPID program. This way the movements for the CNC machine are understandable for the robot. Some reference frames and input/output signals must be declared afterwards to finish the process. For this Thesis the Winpost program with the necessary ABB modules was provided by Giorgos Nikoleris from the Department of Design Sciences at Lund University.

- RobotStudio [2]: This program created by ABB helps with the simulations of the workspace. It is a very useful application to get a good idea how the robot and its controller works. It is possible to load the generated RAPID code and check how the robot will move.

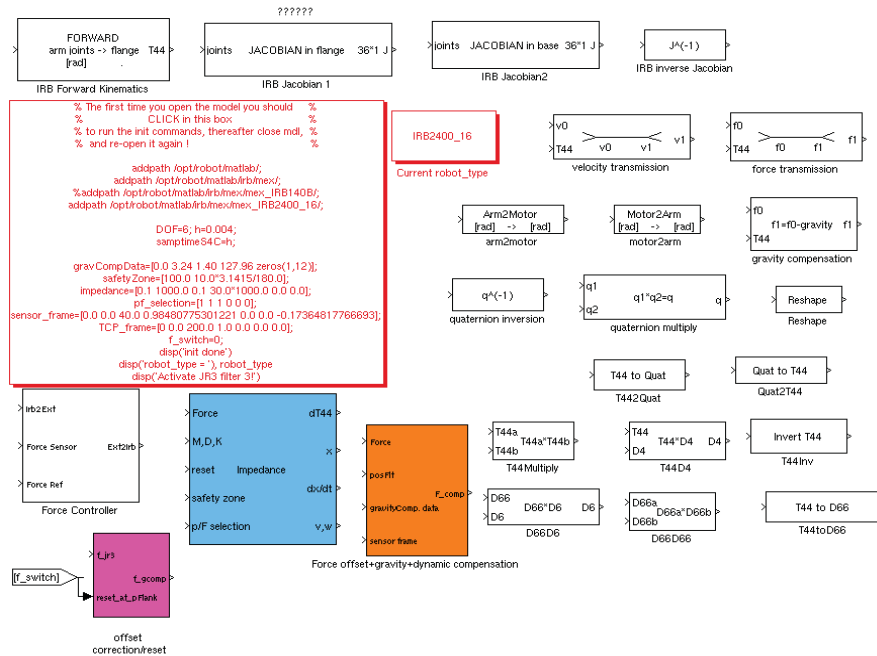


Figure 5: Extctrl library available in the Robotic Lab

- MATLAB [6]: This mathematical program is the main software tool used in this Thesis. All the calculations will be done using MATLAB. Its tool Simulink will

be also widely used. With the Real Time Workshop (RTW) it is possible to build controllers and programs to communicate with the robot and the sensor. A block library called extctrl [7] developed by Isolde Dressler at the Department of Automatic Control at LTH, Lund University is also necessary to obtain the kinematic and dynamic equations of the robot, see Figure 5.

- Opcom: This is the user interface available in the Robotic Lab that shows the state of the robot computer and the sensor. It also allows to control some variables coming from the controller created by the user in Simulink.

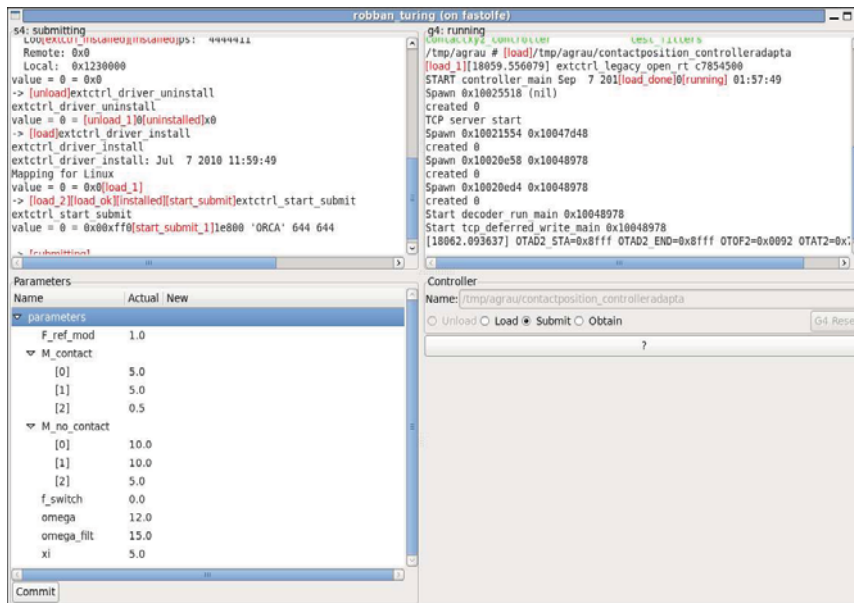


Figure 6: Opcom interface

2.3 Interconnection between software and hardware elements

All the components described previously have to be interconnected in order to run the system. Starting from the robot the first controller is the S4CPlus with its axes computer. A main computer controls both devices but to allow experiments an external regulator is introduced. This extra part acts in between the axes computer and the main computer and loads the programs created in Simulink with the RTW. The user can supervise all this communications with the Opcom interface through a local area network.

The force sensor is directly connected to a different computer and this one to the local area network. It can communicate with the external controller built in Simulink but the S4CPlus does not have any direct interaction with it.

Finally, the user's computer where the programs are created can access the local area network and send the Simulink controller. The user can see the Opcom interface in this computer, so it acts as a data center for the whole system.

A graphic with the connections between all the elements can be seen in Figure 7.

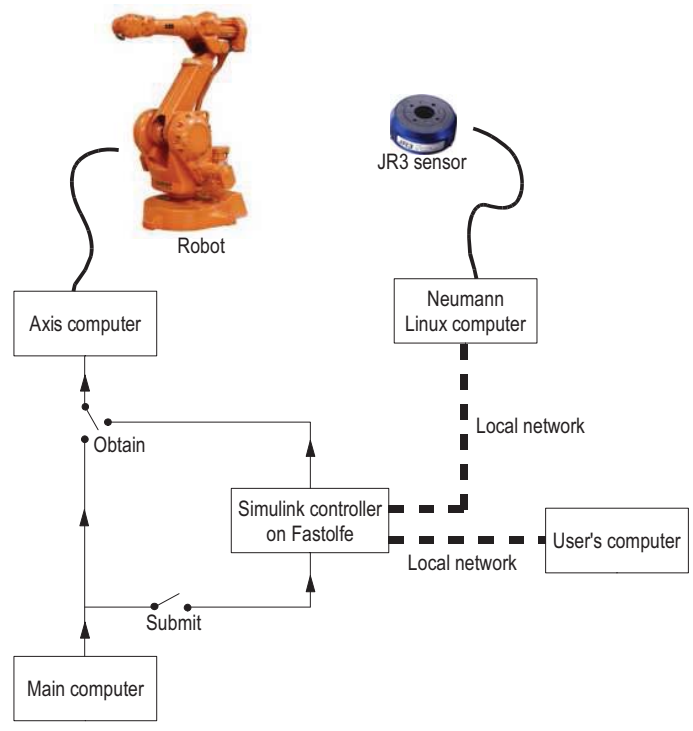


Figure 7: Working environment: connection between the different elements of the system

3. Theoretical base

In the next chapter a brief description of the necessary mathematical tools associated to the control of a robot is presented. Further information about these topics can be found in many book such as [1] or [8].

First of all the way to represent the position of a body in the space is explained. Then specific theory about robotics is introduced as well as some problems that can appear.

3.1 Homogeneous transformations

Geometrics play a huge role in robotics world. When it comes to the description and control of the robot it is crucial to understand how to obtain the position and orientation of every joint and specially the end effector or the tool.

It is usual to number the joints of a robot beginning with the base frame as 0. Unless the robot is mounted on a rail, this frame can be considered stationary. The last joint is numbered as i, in the case of the IRB 2400-16, as 6. Other very common names for some special joints are: the Tool Center Point (TCP) when a tool is attached, the wrist for the last three joints due to their similarity with a human wrist and the flange for the last joint. It is also considered that every joint in the robot has a right-handed coordinate system attached, even the TCP.

It is possible to obtain the position from one joint to another just using rotations and translations. To express this movements the so-called transformation matrices are used [1]. Each column in these matrices represents the relation of the initial origin in relation with the new frame. The whole transformation can be written as:

$$S_b = Rot_a^b \cdot S_a + Trans_b^a \tag{1}$$

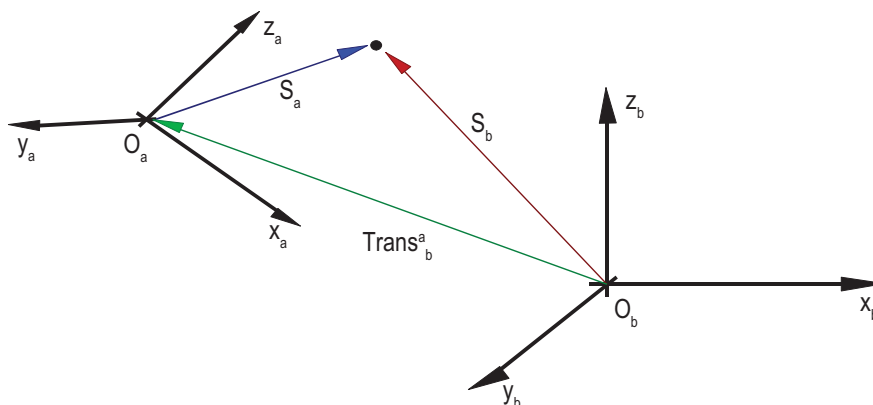


Figure 8: Vector S_a translated and rotated to be expressed as S_b in the new coordinate system O_b

The robot is considered a kinematic chain in the space, then the Rot and Trans matrices are 3x3. S_b is the new vector resulting from the rotation and translation of S_a . As can be observed in the equation it (1) is necessary to work

with two matrices. In order to simplify this notation the homogeneous matrices were introduced.

Homogeneous matrices are based on homogeneous coordinates [1]. To express a position in an n-dimensional space with homogeneous coordinates n+1 elements are necessary, for example, in 3 dimensions it would look like $p=(wx,wy,wz,w)$ where w is a scale factor. Using this notation it is possible to represent in only one 4x4 matrix all the possible transformations:

$$T = \begin{bmatrix} \text{Rotation}_{3 \times 3} & \text{Translation}_{3 \times 1} \\ \text{perspective}_{1 \times 3} & \text{scale}_{1 \times 1} \end{bmatrix} \quad (2)$$

The scale matrix contains the w factor. Perspective and scale matrices will not be used in this Thesis and will remain as $[0 \ 0 \ 0]$ and $[1]$ respectively.

The values in the rotation and translation matrices depend on the specific robot and its geometrics. In the next sections these values will be derived.

3.2 Joint and frames description

In order to allow the control of a robot a mathematical description of its kinematics is needed. The geometry of the links and joints can be found in the data sheet provided by ABB [3]. The most general way to work with these devices is by using the Denavit-Hartenberg (D-H) representation [1].

Denavit and Hartenberg developed a method to describe the position of each joint and its frame in a jointed chain so the kinematics can be derived easily. Four parameters allow to get from one coordinate system S_{i-1} to the next one S_i where i represents the number of joints in the robot:

- d_i : distance along z_{i-1} to get to frame origin O_i . In prismatic joints this parameter is variable.
- a_i : distance along x_i .
- θ_i : rotation around z_{i-1} . In revolute joints this parameter is variable.
- α_i : rotation around x_i .

Using this values it is possible to obtain an injective system to describe the robot and hence to know the position of each joint, see Figure 9. For IRB2400-16 the D-H parameters are:

	d_i (mm)	a_i (mm)	θ_i (°)	α_i (°)
1	615	0	θ_1	-90°
2	0	840	θ_2	0°
3	0	0	θ_3	-90°
4	755	0	θ_4	90°
5	0	0	θ_5	-90°
6	d_6	0	θ_6	0°

The value of d_6 depends on the tool or sensor attached to the flange.

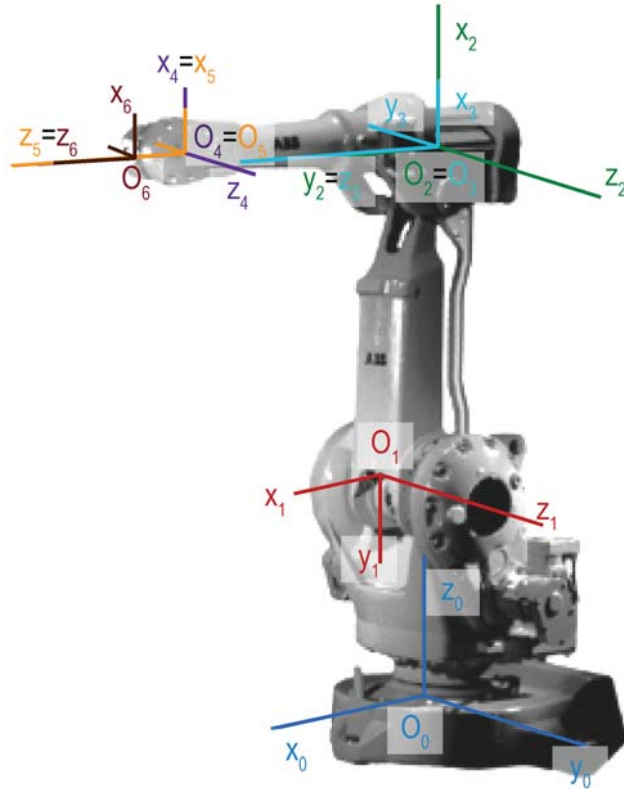


Figure 9: Coordinate system associated to each joint

3.3 Forward kinematics

Once explained how to shift between frames and how to obtain the geometrical parameters of a robot the natural question is how to apply these concepts. By taking them to the robotic world it is easy to find out how to switch from one joint frame to another. The homogeneous matrix corresponding to the change from joint $i-1$ to joint i can be derived by using the D-H parameters and considering the generic form:

$${}^{i-1}A_i(q_i) = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cdot \cos \alpha_i & \sin \theta_i \cdot \sin \alpha_i & a_i \cdot \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cdot \cos \alpha_i & -\cos \theta_i \cdot \sin \alpha_i & a_i \cdot \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

In matrix (3) the element q_i stands for the characteristic parameter of the joint. If it is prismatic it will be d_i and if it is revolute then it will be θ_i . The other elements remain always constant for each joint.

With all this, the transformation matrix from the flange frame of the robot to the base frame can be calculated just by multiplying in the right order:

$$T_{44} = {}^0A_6 = {}^0A_1(q_1) \cdot {}^1A_2(q_2) \cdot {}^2A_3(q_3) \dots {}^5A_6(q_6) \quad (4)$$

As shown in equation (4) the result is a matrix called T44 or simply T. It is very useful to derive the different changes between vectors expressed in the flange frame that want to be moved to the base frame. Also, in the same way that T44 has been calculated other transformation matrices can be derived to shift to other joints or frames farther than the flange, for instance the TCP or sensor frames.

This method is the solution for the so-called forward kinematics problem [1]: how to obtain the position of the flange or the TCP once the values of the characteristic parameters of each joint are known.

3.4 Inverse kinematics

It is possible to think that the position of each joint can be found out just by coming back by the same way used before to obtain the forward kinematics, but that is not mathematically true. This is known as the inverse kinematics problem [8]: how to place the joints in order to move the TCP to a desired position. It is much harder to solve than the forward kinematics problem and some extra algorithms need to be considered.

The main difficulty is that computing the inverse of the forward kinematics problem represents to work with twelve equations although only six unknown values are necessary. And most of the times the solution is non-trivial or even does not exist because some of the matrices can be singular. It also happens that more than one solution could arise, that means that the robot can reach the same position with different joint configurations and in the worst case with infinite if two revolute joints are coincident.

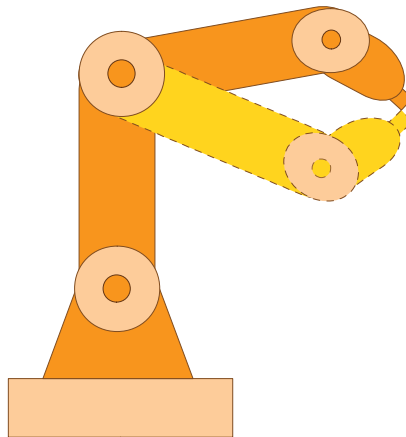


Figure 10: Two possible joint configurations to reach the same point

Because of all these reasons it is usual to work with other methods to solve the inverse kinematics problem. With a big number of industrial robots such as the one used here the first three joints are contained in the same plane and they determine the position of the TCP. The last three joints intersecting in the same point and forming the wrist are responsible for the tool orientation. This way a sixth order equation system can be split into two of third order. First of all the

position of the first three joints is derived and finally the orientation of the wrist is obtained.

In addition some extra constraints are used in the solution to make it as optimal and predictable as possible, for instance: if more than one solution is possible the one that is closer to the actual position will be chosen or limit the necessary route of the joints from one point to another.

3.5 Jacobian (differential kinematics)

Knowing the joint velocities is as important as knowing the position. Once the position and orientation of the TCP, written as (5) and (6), are derived, it is possible to differentiate them with respect to the time. By doing this the linear and angular velocities of the TCP are obtained. All this can be expressed using matrices:

$$x_{TCP} = f_x(q_1, q_2, q_3 \dots q_6) ; y_{TCP} = f_y(q_1, q_2, q_3 \dots q_6) ; z_{TCP} = f_z(q_1, q_2, q_3 \dots q_6) \quad (5)$$

$$\alpha_{TCP} = f_\alpha(q_1, q_2, q_3 \dots q_6) ; \beta_{TCP} = f_\beta(q_1, q_2, q_3 \dots q_6) ; \gamma_{TCP} = f_\gamma(q_1, q_2, q_3 \dots q_6) \quad (6)$$

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_{TCP} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}_{TCP} = J(q) \cdot \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_6 \end{bmatrix} \quad (7)$$

$$\text{where } J(q) = \begin{bmatrix} \frac{\delta f_x}{\delta q_1} & \frac{\delta f_x}{\delta q_2} & \dots & \frac{\delta f_x}{\delta q_6} \\ \frac{\delta f_y}{\delta q_1} & \frac{\delta f_y}{\delta q_2} & \dots & \frac{\delta f_y}{\delta q_6} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta f_\gamma}{\delta q_1} & \frac{\delta f_\gamma}{\delta q_2} & \dots & \frac{\delta f_\gamma}{\delta q_6} \end{bmatrix} \quad (8)$$

The notation v_i and ω_i in equation (7) refers to linear and angular velocities respectively. The matrix J that can be seen in (8) is known as the Jacobian matrix and relates how changes in the joint variables affect the velocity of the TCP. The upper half elements affect the linear velocity and the lower half elements affect the angular velocity. All the elements contained in J change for different space locations.

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_6 \end{bmatrix} = J^{-1} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad (9)$$

The inverse Jacobian can be used to calculate the values of the joint velocities as shown in equation (9). Depending on the robot some adjustments could be necessary such as using the pseudoinverse or keeping some values constant. Further information can be found in [1] and [8].

In addition to all this, some extra considerations have to be done. Some times it can happen that $\det(J)$ is equal to 0 because the robot is in a singular configuration. In those cases it is impossible to find the inverse Jacobian. Besides, using this method will block the motors because an extremely high velocity would be required in the surrounding areas of a singular point.

These situations were mentioned in the inverse kinematics section. They can be avoided by adding constraints or forbidden areas to the path of the robot. Obviously, they will vary from one robot to another depending on its working space, the joint configuration, the movement required, etc.

The Jacobian can be also used to find out accelerations and torques in the motor joints as explained in [1] or [8].

3.6 Force control

Two main strategies for force control can be found as can be read in [8]. The first one does not need a real force measurement. This is called passive interaction control and can be used for tasks where the geometry of the environment is, at least, roughly known. The motors of the robot or the tool present the necessary compliance to interact with the surroundings. It is common to find this control in procedures such as inserting pieces in holes and they usually need special tools with springs to avoid collisions and to allow smooth movements.

The other strategy is active interaction control. The force is measured with sensors and takes active part in the controller. Sometimes the idea is to emulate a compliance system like the ones used for passive control with more precision but slower. This will be the option chosen in this Thesis. Some of the different types of controllers with active interaction control will be explained.

Direct force control

It is the simplest regulator that can be built. It acts on the force error between a desired value and the measured force. A PI controller is very usual in order to cancel the position error. The output is used to keep the contact with the environment by altering the position or velocity of the TCP.

Hybrid position/force control

Position is also a parameter that sometimes needs to be controlled. The main problem with pure force regulators is that when no force appears it is somehow hard to control the behaviour of the robot. To solve this problem hybrid controllers are introduced.

While there is no force measured by the sensor the position of the robot is derived with one regulator but when the contact starts the force regulator begins to work. This way none of the controllers affects each other.

Although it is a very common solution it is not easy to control the switch from one controller to the other and depending on the application this may lead to problems. Many times the controller also has to allow a position error to avoid high forces acting on the joints or the tool so the force control is predominant over the position control.

Impedance force control

This controller tries to obtain a compliance behaviour of the robot when it comes to following position and velocity references but also to environment restraints [9]. This means that position and velocity will be the control signals while no force is measured but if any contact appears the robot will adapt its path to the requirements. The idea is to imitate a mass-spring-damper system with certain dynamics to ensure no collisions but following as much as possible the desired trajectory with the requested velocity.

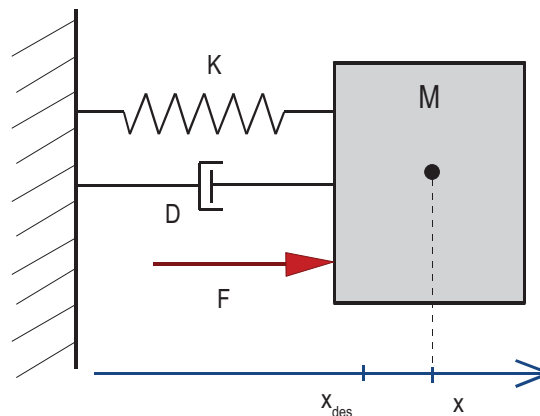


Figure 11: Mass-spring-damper system

In many books impedance and admittance controllers are treated with no distinction. However, some authors [8] point out that admittance controllers modify position or velocity to reach the force reference while impedance controllers try to follow the desired trajectory by producing force due to the external restraints. The practical difference is that a force reference may be needed as an input to the controller.

To obtain the behaviour of this controller all the forces acting on the robot have to be considered. As the system wants to imitate the mass-spring-damper system this will mean: a mass moving at a certain acceleration, a damper affected by the velocity, a spring changing its position and an external force. Adding all

those elements that describe the dynamics, the equation (13) can be written. Then, substituting (10),(11) and (12) it is possible to derive (14) as shown below.

$$\sum F = M \cdot \ddot{x} \quad \text{Newton's second law} \quad (10)$$

$$F_s = -K \cdot \Delta x \quad \text{Hooke's law (spring equation)} \quad (11)$$

$$F_d = -D \Delta \dot{x} \quad \text{Damper equation} \quad (12)$$

$$\sum F = F_{ext} + F_s + F_d \quad (13)$$

$$F_{ext} = M \ddot{x} + D \cdot \Delta \dot{x} + K \cdot \Delta x = M \cdot \ddot{x} + D \cdot (\dot{x} - \dot{x}_{ref}) + K \cdot (x - x_{ref}) \quad (14)$$

The parameters M, K and D represent the mass, the spring constant and the damping coefficient respectively. They depend on the properties of the physical element and for the controller they need to be derived to obtain the right behaviour. The values of x and x_{ref} designate the real position of the mass and the desired position.

By rearranging the equation it is possible to work out the acceleration, which can be integrated in order to obtain the speed and position that will be sent to the robot.

$$\ddot{x} = \frac{F_{ext} - D \cdot (\dot{x} - \dot{x}_{ref}) - K \cdot (x - x_{ref})}{M} \quad (15)$$

Due to the discrete nature of the controller the integrals are substituted by a summation with the initial position x_0 and the sample period T_s as necessary extra parameters. So the calculations to be done by the controller will be like:

$$\dot{x}_k = \sum_{i=0}^k \ddot{x}_i \cdot T_s \quad (16)$$

$$x_k = x_0 + \sum_{i=0}^k \dot{x}_i \cdot T_s \quad (17)$$

The coefficients K and D have a physical meaning but when it comes to give them a value in the controller it turns out to be a tricky task. An analysis about the nature of the system can be done to get a better idea of how this variables affect the dynamics. By switching from time domain to frequency domain the second order differential equation (14) can be expressed as:

$$s^2 + \frac{D}{M} \cdot s + \frac{K}{M} = 0 \quad (18)$$

The characteristic equation of a second order system is represented as:

$$s^2 + 2 \cdot \zeta \cdot \omega \cdot s + \omega^2 = 0 \quad (19)$$

Then, comparing both of the equations the physical coefficients can be easily derived and related to ζ and ω to explain their relevance in the behaviour of the system. The parameter ζ represents the damping factor. For values in $[0,1[$ the system is underdamped so oscillations will appear, if it is equal to 1 it is said that the system is critically damped and it will not oscillate, this tendency remains for higher values although the transient will last longer, the system then is overdamped. The value of ω refers to the natural frequency of the system, also called cutoff frequency.

Then D and K can be calculated in order to obtain the desired dynamics. They are expressed as:

$$D = 2\zeta \omega M \quad (20)$$

$$K = \omega^2 M \quad (21)$$

4. Pro/ENGINEER

This is the first step necessary for the manufacturing process. With the help of Pro/ENGINEER, the user will design the piece that the robot will produce later.

During the design process it is really important to keep in mind that the robot will be the responsible for the machining tasks. This has an obvious influence when it comes to parameter definition such as velocities or tool paths.

As the use of this software represents a big part of this Thesis a brief manual to create a piece with the most general alternatives can be found in Appendix 1. It can be avoided if the user has already experience with the program.

The usual procedure is to declare a raw block of material, also known as workpiece, where the final part is embedded. Then, the next steps consist on creating volumes that the milling tool will remove until the desired shape is reached. Of course, depending on the complexity of the design, some other options are available, such as facing flat surfaces or drilling holes. Along the manual in Appendix 1 different techniques are presented and it is a user's decision to choose the most optimal for each job.

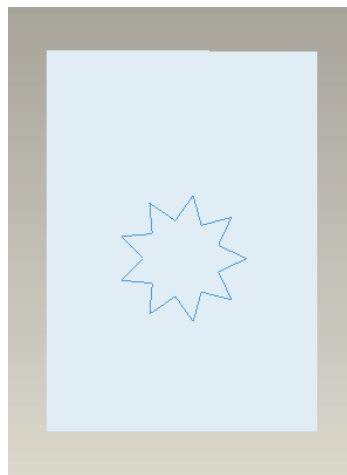


Figure 12: Picture created to test the machining algorithm

For the tests in the Robotic Lab a flat picture was create in order to polish up any mistake before using a drilling or milling machine, see Figure 12. This way a simpler tool such as a marker or pencil can be attached to the robot so the algorithm of control, positions and velocities may be corrected.

After completing all the steps, the output from Pro/ENGINEER is a generic G-code that can be adapted to different CNC machines once it is postprocessed. At this point it is important to remark that during the execution on the robot, in order to obtain a proper behaviour, some of the positions reached by the tool and declared in the G-code will be used to indicate to the regulator a change in the control strategy. Therefore, the user must select carefully things such as retract surfaces or initial points in order to obtain a consistent error-free algorithm. These details will be further explained in the next characters.

5. Postprocessor

The postprocessor just translates the code coming from Pro/ENGINEER into some specific code that the tool machine can understand. It is usual that each company owns a different language for these machines so it is kind of tricky to program just by hand. Nevertheless, the code mainly involves movement commands which can be roughly understood.

The exact postprocessor used here is Winpost V5.0 provided by Giorgos Nikoleris from the Department of Design Sciences at Lund University. The software is a really simple application developed just to perform the translation from one language to another, no further information is available.

It is necessary to add in the Winpost folder a small program called abb.exe containing the equivalences between the G-code and the ABB RAPID code so the postprocessor can generate a new file containing the movements for the robot.

5.1 Postprocessor output: Corrections and notes

The input to Winpost is the ncl.1 coming from Pro/ENGINEER. The output after clicking on *Postprocess* is a .prg file. This file can be executed by the S4CPlus after some corrections or even simulated in RobotStudio. The raw program without modifications looks like:

```
%%%  
VERSION:1  
LANGUAGE:ENGLISH  
%%%  
  
MODULE MILLING  
  
PROC main()  
  ConfL\Off;  
  Mill;  
  
ENDPROC  
  
PROC Mill  
  Reset DO10_9;  
  Reset DO10_10;  
  SetAO AO10_1, 1500.0;  
  MoveAbsJ omor,v40,z1,tool0;  
  MoveAbsJ home,v40,z1,tool0;  
  MoveL [[82.50,0.00,15.00],[0.0000,1.0000,0.0000,0.0000],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v40,z0,t11Wobj:=;  
  MoveL [[82.50,0.00,6.00],[0.0000,1.0000,0.0000,0.0000],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v40,z0,t11Wobj:=;  
  Set DO10_10;  
  MoveL [[82.50,0.00,0.00],[0.0000,1.0000,0.0000,0.0000],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v17,z0,t11Wobj:=;  
  .  
  .  
  .  
  MoveL [[38.50,100.00,-20.00],[0.0000,1.0000,0.0000,0.0000],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1,z0,t11Wobj:=;  
  MoveL [[38.50,100.00,15.00],[0.0000,1.0000,0.0000,0.0000],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v1,z0,t11Wobj:=;  
  Reset DO10_10;  
  MoveAbsJ home,v40,z0,tool0;  
  MoveAbsJ omor,v40,z0,tool0;  
ENDPROC
```

The .prg containing the RAPID code can be easily modified with any text editor. Probably, due to some errors in abb.exe, some corrections need to be done by hand. Otherwise a warning will arise when trying to execute the .prg on the

S4CPlus or RobotStudio and will not allow to load the program. The necessary changes are, at least:

- The postprocessor produces a misspelling in the RAPID code. The 'main' procedure calls the procedure 'Mill' but the definition of this procedure has an error in its header. A procedure with no input arguments should be declared as '*PROC procedure_name()*'. This means that '()' is missing after the name of the procedure. It has to be included by hand.

- The program, as shown before, does not have an end. *ENDMODULE* is missing and has to be added after the last line.

- The commands *Reset*, *Set* and *SetAO* are related to digital and analog outputs. If the tool attached to the robot is controlled externally or does not have those inputs it is necessary to remove or, at least, comment those commands along the program (to comment '!' has to precede the line).

- A very important detail is the machine coordinate system. The text '*Wobj:=*' is an incomplete command. The '\' indicates that it is an optional parameter but for this Thesis it has to be declared to obtain a proper behaviour. *Wobj* refers to a coordinate system associated to the work object. That is, the coordinate system attached to the part that the robot will machine and it has to be defined in order to be coincident with the one created in Pro/ENGINEER and used as *Machine Zero*. So, to solve the error that arises from this indeterminate parameter it is enough with writing a name after the equal, for instance '*Wobj:= Workobject_1*'. This is only a partial solution because the coordinate system named *Workobject_1* has to be exactly declared later on with the rest of the variables.

With all these changes the program is ready to be exported to S4CPlus or RobotStudio but the variables such as positions or speed values have not been defined yet. Until all those parameters are not included in a .sys file the movement can not be simulated or run by the real robot.

6. RobotStudio

ABB offers a powerful simulator to test RAPID programs. RobotStudio allows to create tools, conveyors, workers, etc. to check the behaviour of the robot under certain circumstances and avoid or correct wrong movements depending on the environment [2]. It is also very useful to become familiar with the Teach Pendant that controls the real robot.

In this Thesis RobotStudio was helpful to create the .sys file containing the variable declarations necessary to run the postprocessed program.

6.1 Program structure

Once the user has corrected all the errors in the .prg program coming from the postprocessor it can be loaded on RobotStudio.

First of all, open RobotStudio and select a *New Station*, for instance IRB2400_16kg_1.5m. Name it and press *Ok*. The virtual controller will be started and the picture of the robot will be displayed on the screen. In the *Offline* tab go to *Load Program* and select the .prg program from the folder where it is stored.

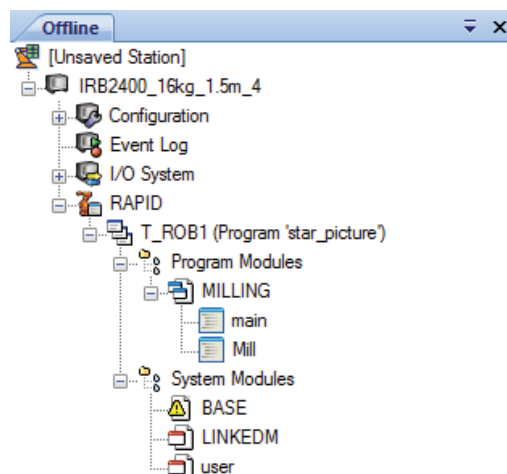


Figure 13: Component tree in the Offline tab

A component tree is shown on the left of the screen. Under the *RAPID* label it is possible to see the structure of the program. *Program Modules* label refers to the different parts of the .prg stored as .mod, in this case they are the main procedure and the machining procedure. *System Modules* has three sub-elements that can be found with the extension .sys: *BASE* and *LINKEDM* are automatically created and contain default values and variables, *User* can be edited and the necessary variables will be declared there. By doing double click on each of those files they are opened on the main window so they can be modified or reviewed.

6.2 Variable details

In the program coming from Windpost some variables are used but not declared:

- *omor*: Swedish abbreviation for reorientation. It is a position that the robot will use to begin and finish the movement. Depending on the task this position may be useful to indicate to the regulator changes in the state of the system so the control can be adapted to the new situation.

- *home*: Initial position. It can be declared so the robot is closer to the first location of the machining task. With *omor*, it can also be used to indicate changes to the controller.

- Tool: Accepting that the tool will remain invariable during the whole process it is necessary to declare it. Notice that *tool0* (default tool) appears in some lines that imply a *MoveAbsJ* movement. The displacement with this command is not affected by the tool or the work object but it would be more accurate to change *tool0* for *t11* in order to allow the controller a proper load calculation.

- Zone data: This kind of parameter indicates how accurate the movement has to be. It represents the maximum admissible space that can appear between the programmed final position and the real location before the robot starts the next movement. The default value is *z0* but smaller values can be defined if they are necessary. The user has to consider that depending on the material or the tool path this parameter may affect the final aspect of the machined part.

- Speeds: Some variables defined in Pro/ENGINEER have direct repercussion in the RAPID code. Every movement needs an execution speed. This speed derives from the *Cut_Feed* parameter declared in Pro/ENGINEER, which represents the relative velocity between the tool and the table where the workpiece is fixed. Its units are consistent with the ones selected in the Pro/ENGINEER file and tool machine. So, if mm were selected and the default units were unchanged in the tool machine definition the *Cut_Feed* is expressed in mmpm, that is mm/min. RAPID programs work with mm/s so this has to be taken into account.

Three possible options are available:

- 1) To consider the change of units since the beginning and apply the necessary calculations: to change from mm/min to mm/s divide by 60. So if the *Cut_Feed* is 1000 the speed will be 1000/60. The number is rounded towards the next integer, so the speed will look like *v17*. For a desired speed the operation will be a product: to obtain *v20* the *Cut_Feed* should be $20 \cdot 60 = 1200$ mmpm.

- 2) To substitute all the speed parameters in the .prg file for the desired values. This can be done with the *Replace* option in a text editor or once the program is loaded in RobotStudio.

- 3) To declare a new speed variable if the desired velocity is not available in the default definitions.

- Work object: Represents the coordinate frame attached to the part to be machined. It has to be declared as a variable and it must be coincident with the one defined in Pro/ENGINEER as *Machine Zero*. Otherwise the tool path will be performed out of the bounds of the real block of material.

6.3 Variable definition

The previous variables have to be included in the user.sys file. According to RAPID syntax [10] the declaration of variables has to be done as follows:

-Positions: For an absolute movement, with no reference to a coordinate system the angle of each joint has to be specified, that is: six angles in degrees for the six joints of the robot and six more angles for external extra joints if necessary, otherwise the last six terms remain as 9E9. Special attention must be paid in order to avoid errors such as using singular positions.

```
CONST jointtarget omor=[ [ 0, 0, 30, 0, 0, 90], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];  
CONST jointtarget home=[ [ 0, 0, 0, 0, 10, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
```

- Tool: Some physical aspects of the tool need to be described here. In the tool definition shown below the different elements described are, in order:

- a) If the robot is holding the tool (TRUE).
- b) The tool coordinate system placed in the TCP in relation with the wrist coordinate system. The first three elements describe the position and the other four express the orientation in the form of a quaternion.
- c) The load of the tool with its weight in kg, the center of gravity, axes of moment as a quaterion and moments of inertia in kgm^2 . If the axes of moment and the moments of inertia are not defined the tool will be considered as a spot load.

```
PERS tooldata t1:= [ TRUE, [[0, 0, 200], [1, 0, 0, 0]], [5, [0, 0, 100], [1, 0, 0, 0], 0, 0, 0];
```

- Speed: If the desired speed is not a multiple of five it will not be defined by default. It can be declared in the user.sys. The elements describe the velocity of the TCP in mm/s, the velocity of reorientation of the TCP in degrees/s, and the same two parameter but for external axes. The real speed performed is limited by the reachable velocity of some of the parts.

```
CONST speeddata v17:= [ 17, 500, 5000, 1000];
```

- Work object: This parameter expresses where the coordinate system of the work object is placed in relation with the coordinate system of the workspace. The required elements for this variable are:

- a) If the robot is holding the work object (FALSE).
- b) If the workspace coordinate system, also know as user coordinate system, is fixed (TRUE).
- c) If the workspace coordinate system is movable a trajectory has to be included, otherwise this field is empty.
- d) User or workspace coordinate system in relation with the world coordinate system which is usually located at the base of the robot. The two sub-elements are position and rotation.
- e) Work object coordinate system expressed in the user coordinate system. Again declared with position and orientation.

```
PERS wobjdata workobject_1:=[FALSE, TRUE, "", [[700, 0, 800], [1, 0, 0, 0]], [[50, 0, 0], [1, 0, 0, 0]]];
```

Once all the variables are declared the program can be run to see if the trajectory of the robot is the desired. Just press *Start* on the editor bar in

RobotStudio and go to *View task* to see the simulation. If any error is still uncorrected a warning will appear in the *Output* tab located in the lower part of the window. If the behaviour is right the user.sys file can be saved.

Notice that if the whole program is saved with RobotStudio it will be stored as a .pgf plus a .mod. This files are more modern than the .prg and S4CPlus can not manage them. So the best option is to save only the user.sys and edit by hand the .prg file with the necessary changes derived from the simulation on RobotStudio.

7. Robot control: ideas and basis

In this chapter the practical part of the Thesis is explained. The robot needs to perform some routines before running the machining process in order to calibrate the force sensor, obtain an accurate picture of the workspace and ensure bumpless movements.

Matlab and Simulink will be the main tools to build the robot controllers.

7.1 Gravity compensation

The main objective in this Thesis is to machine a piece which involves contact between the robot carrying the tool and the environment. The force sensor attached to the wrist of the robot provides the force and torque measurements necessary to find out the value of the force and apply a control loop. The main problem with this sensor is that it is affected by many factors other than the contact force. So the first step is to calibrate the sensor and try to obtain real force readings.

When the sensor is first started an offset value appears. The block called *Offset correction/reset* in the extctrl library cancels the initial offset when a trigger is activated but another problem appears. The reset is only valid when the sensor is not reoriented. The weight attached to the flange is considered as part of the offset so it is initially deleted but, when the wrist is moved, the effect of the weight appears in the sensor distorting the measurement.

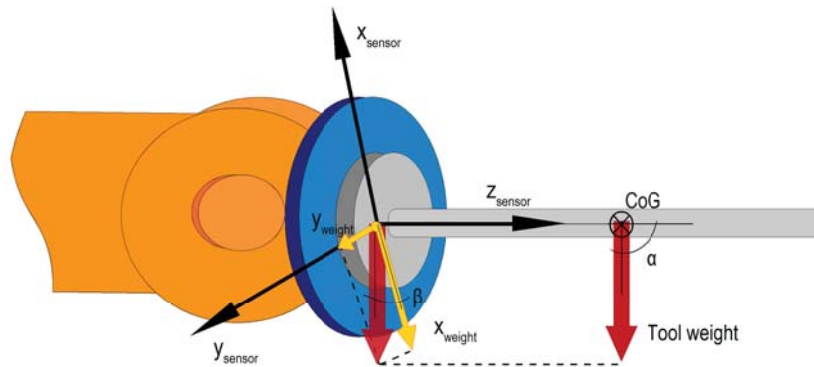


Figure 14: Tool weight projected over the sensor frame. Weight perpendicular to z_{sensor} so $z_{\text{weight}}=0$

If there is no contact with the environment the force and torque reading is due to the tool weight and the own weight of the sensor, as can be seen in Figure 14. It can be considered as a force pointing to the floor applied in the center of gravity of the union tool/sensor, so the sensor registers the projection of this magnitude on its axes. Therefore, if the angles between the weight and the sensor axes are known it is possible to generate the profile of forces and torques and, this way, correct the measurement as expressed in (22). This is the main idea behind the gravity compensation.

$$F_{sensor} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} (weight \cdot \sin \alpha) \cos \beta \\ (weight \cdot \sin \alpha) \sin \beta \\ weight \cdot \cos \alpha \end{bmatrix} \quad (22)$$

To generate an accurate law that simulates the effect of the weight on the sensor some parameters need to be obtained. The sensor frame is not perfectly parallel to the flange frame so the angle and offset from one frame to the other are necessary, that is the Denavit-Hartenberg parameters from flange to sensor. Then the weight of the elements outside the sensor has to be derived. The center of gravity is also required in order to know the application point. The other element necessary that can be obtained easily is the orientation of the flange. Once all these parameters are available the algorithm to compensate the effect of the gravity can be roughly understood as:

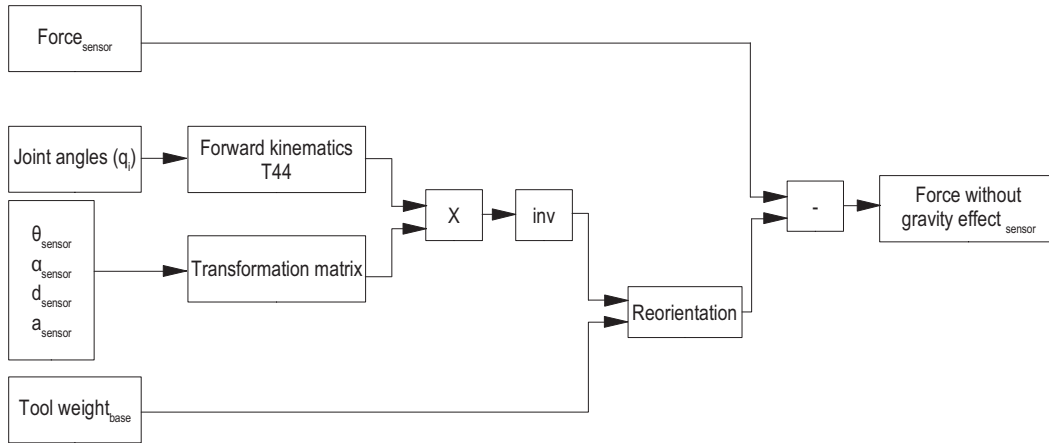
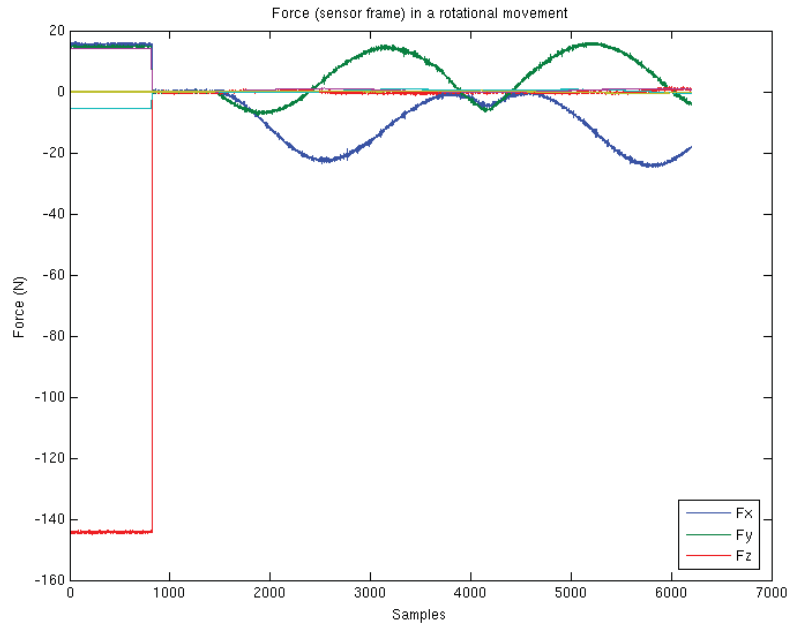


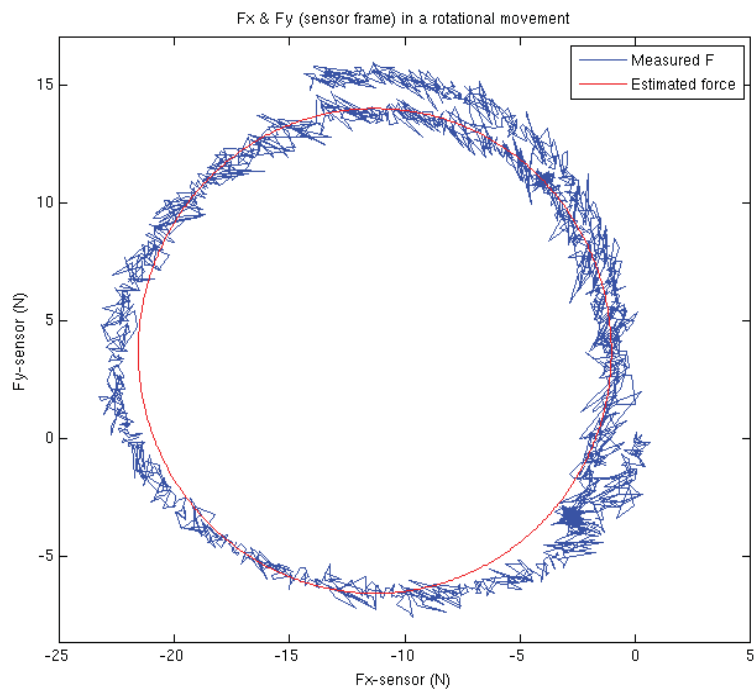
Figure 15: Gravity compensation algorithm. The weight of the tool is rotated to be expressed in the sensor frame such as $Weight_{sensor} = R_{flange}^{base} \cdot R_{sensor}^{flange} \cdot Weight_{base}$. Then it is subtracted from the measurement coming from the sensor. The result is a new force in the sensor frame without the effect of the tool load.

To obtain all those elements the best option is to execute some experiments reorientating the tool. By doing this the weight and position of the sensor frame can be estimated although the sensor is not perfect and white noise will affect the values.

The first procedure places the z axis of the sensor parallel to the floor and then rotates the wrist. This way the load vector is perpendicular to the floor and, in some points coincident with x and y axis. Just considering the forces it is possible to accept that F_z is roughly zero and F_x and F_y record the weight. A plot with the results of the experiment is presented in Figure 16.



(a)



(b)

Figure 16: Experiment: Rotate wrist with no tool to measure the empty-load.

(a) Force measurement. The sensor is reset after sample 825.

(b) F_x in front of F_y coming from the same experiment. The estimation is done with a circle of radius 10,27 and offsets equal to -11,3 N and 3,73 N for F_x and F_y respectively.

The radius of the circle is coincident with the weight or external load and the center indicates the offset, see Figure 17. The general equation of a circle is:

$$r = \sqrt{(x-a)^2 + (y-b)^2} \quad (23)$$

With the force parameters measured the equivalences are quiet straightforward and the next equation can be written:

$$Load_{tool+sensor} = \sqrt{(F_x - F_{(xoffset)})^2 + (F_y - F_{(yoffset)})^2} \quad (24)$$

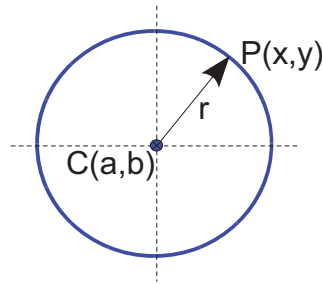


Figure 17: General shape and parameters for the mathematical description of a circle

Other parameter that can be obtained once the weight and offsets are known is the angle between the flange frame and the sensor frame. In Figure 18 it is shown the difference between the estimated force and the real measurement. This helps to identify the different phases due to the orientation of the sensor with respect to the flange. It is also useful to correct other values such as the weight.

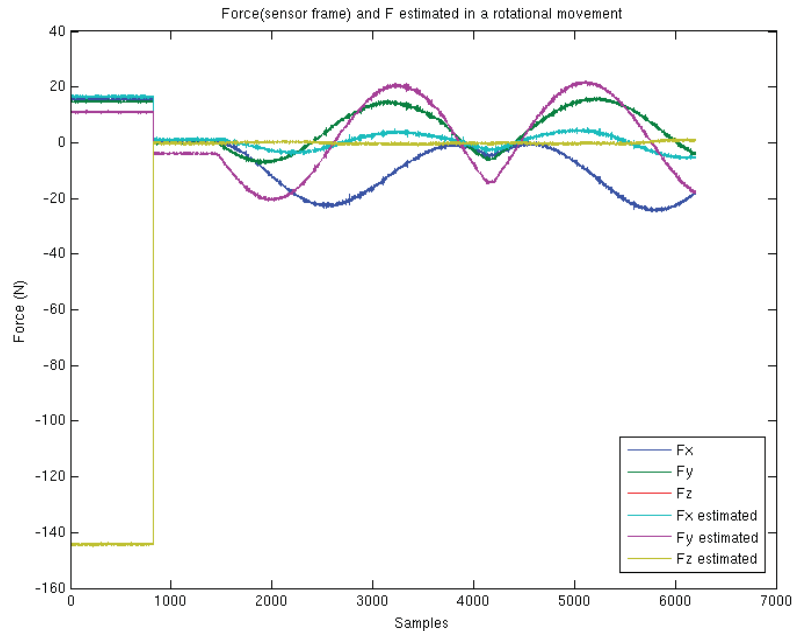


Figure 18: Real measured force and estimated values. Reset after sample 825

When the offsets in x and y are calculated the same process has to be done for z. The robot has to be jogged to some key positions where the weight is coincident with the different axes. With those values and the orientation of the tool the shape of the curves estimating the forces due to the tool movement can be derived and then, subtracted from the real sensor reading to obtain the value of the contact force without weight or offset effect.

With this concept in mind the algorithm that calculates the tool parameters has to assure enough accuracy. The best option is to generate a program which moves the robot to different positions, logs the data coming from the sensor and the orientation of the tool and then, analyzes all those values by approaching them to parametric curves and obtains the average values during the whole movement. This program is conceptually simple but really time-consuming when it comes to its creation.

The task explained before may seem trivial but the reality shows that it is not so easy. As can be seen in the previous images, the real plot is not perfect: the noise introduces a big error, so it is necessary to find the average of the values and repeat the experiment several times to obtain accurate parameters good enough to be used in the gravity correction. In addition, every time that a new tool is attached it is necessary to obtain its weight, center of gravity, inertia, etc.

In the beginning a very raw approximation was done to check how demanding the calculation was. It showed up to be a tough task that requires many tests and not a simple estimation. A program to move the robot to different positions and get the necessary data was started but, by that time, a Simulink program was developed by Andreas Stolt at the Department of Automatic Control in order to compensate the gravity effect in force measurements. The program works with a huge amount of data and it takes almost an hour to analyze them, but it reaches very precise values.

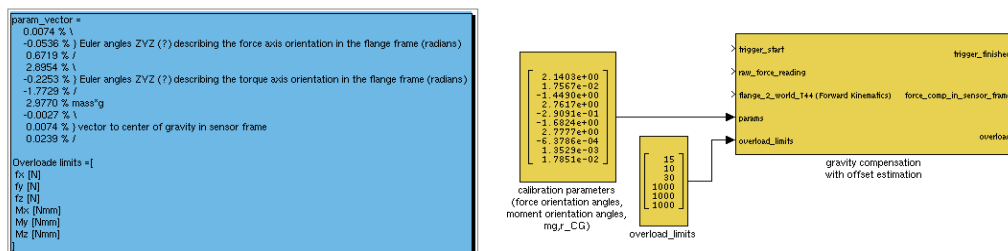


Figure 19: Gravity compensation blocks developed by Andreas Stolt at the Department of Automatic Control in Lund University

The procedure behind this Simulink program is similar to the one presented before: the robot is moved to certain positions and the data is logged. Then a mathematical analysis is carried out by a .m file that approaches the force measurement to a non linear function of weight and orientation. This way the coefficients of this function are worked out and they can be used to predict the value of the force depending on the position and the tool. The program outputs a group of parameters such as tool weight, axes orientation and center of gravity position. With them it is possible to remove the effect of the tool weight from the sensor reading and provide reliable measurements with no gravity action just by using the block shown in Figure 19.

7.2 Impedance controller

When the characteristic parameters of the tool are known they can be used to correct the force measurements coming from the sensor. Although a small noise affects the reading it can be easily reduced by using a low pass filter. So, from now on, the force reading can be considered accurate enough and just due to the interaction between the tool attached to the robot and the environment.

In order to perform activities which involve contact the input from the force sensor can perfectly act as a signal in a control loop. An impedance controller is a good option when the shape of the contour is not perfectly known. The application presented in this Thesis uses raw blocks of material with irregular or unknown shapes so it is important to allow a certain degree of flexibility. If the robot acts as a very stiff system the movements can cause damages and collisions. On the other hand, the robot must resist some forces to ensure that the tool machines the material in the right way.

The best option to meet this two demands in one controller is to use the impedance control. With this method it is possible to reach a specific force value with certain elasticity and also follow a given trajectory.

Two main tasks are necessary to complete the machining process: first of all the raw piece has to be placed and located in the workspace in order to provide the right coordinates to the program, the second task is the machining itself. In both procedures the impedance controller can be used with some variations.

7.3 Contact algorithm

The robot needs to know the exact location of the workpiece. It can be provided by the user but in order to obtain a really accurate position the robot will perform a program which follows the edge of the raw block of material.

The impedance controller usually needs three inputs: force, position and velocity. The problem in this particular case is how to provide an accurate position reference. This input can be removed so the controller will work with velocity and force. When an impedance controller receives only force and velocity is also called damping control [8].

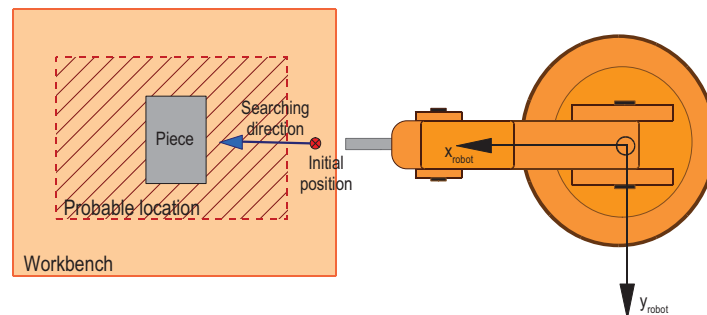


Figure 20: Working scenario

First of all some design requirements need to be established for the algorithm. A possible scenario is the robot located in front of the workbench where the raw block of material is placed, see Figure 20. The position of the

workbench is considered invariable along the machining process. Then, on that surface a 'probable area' where the workpiece can be found is defined, it is also invariable. These elements should be fixed because the algorithm needs some certainties in order to be optimal.

Once the workspace is defined the movement can be planned. The TCP of the robot needs to be placed at an initial position, from there it will move towards the working area with a certain direction until a force appears, meaning that the contact with the workpiece has been reached. The next step is to direct the TCP along the edge of the block of material trying to keep a constant contact. If at some point the contact is lost because of a corner or some irregularity the algorithm should lead the TCP back to the edge.

Finally, when the TCP has turned around the complete perimeter of the workpiece it is possible to obtain its silhouette. To make it as much accurate as possible only the points where some force was measured will be represented, this way if the contact was lost or some bounce happened it will not appear in the graph. With this contour-map a point that will work as coordinate system origin for the workpiece can be chosen.

7.4 Machining algorithm

The G-code postprocessed to obtain the RAPID program with the movements of the tool can be used to machine the workpiece. The problem again is the accuracy. No command in the program offers information about the force required for the machining task. This can affect the output from the process positively if the material interacts properly with the force developed by the robot but, on the contrary, if the robot acts with high stiffness as it is supposed to do it can break the material, the tool or produce collisions. To solve the problem and obtain a more flexible behaviour the impedance controller is introduced.

The difference with the contact algorithm is that now there is certainty about what position the tool must reach, so the reference inputs to the controller will be the position and velocity coming from the RAPID program and the desired force. The force reference should be adjusted depending on the material and the tool to avoid any damage.

Notice that other possibility is to run in parallel the impedance controller modifying the velocity and position from the RAPID program but this will produce more distortion in the trajectory and, therefore, in the final machined part.

8. Robot control: practical tests

In previous chapters the necessary concepts and ideas to carry out this Thesis have been presented. In this point the experimental results obtained and the programs generated will be shown. Solutions or changes adopted to correct errors or optimize the process will be explained too.

8.1 Contact program

This program controls the robot in order to obtain the shape of the workpiece, it is generated in Simulink. In Figure 24 the block diagram corresponding to the contact program can be seen. The different boxes will be described to allow a good understanding of the controller. Two points talking about the activation signals and the script that pictures the shape of the workpiece are also included.

- 1st box:

This box contains the blocks responsible for gravity compensation. The inputs are the signal f_switch which controls the activation of the system, force measured by the sensor, the homogeneous matrix with the rotation and orientation of the flange, parameters describing the tool (mass, center of gravity, etc) and finally some limit values for the sensor to avoid any damage.

The outputs are three: a signal that changes from 0 to 1 to indicate the beginning of the correction (it takes 50 samples to calculate some parameters), the corrected force measurement in the sensor frame and a flag to show if the sensor is overloaded.

- 2nd box:

This block processes the corrected force and translates it from the sensor frame to the base frame. The sub-blocks inside this box can be seen in Figure 21.

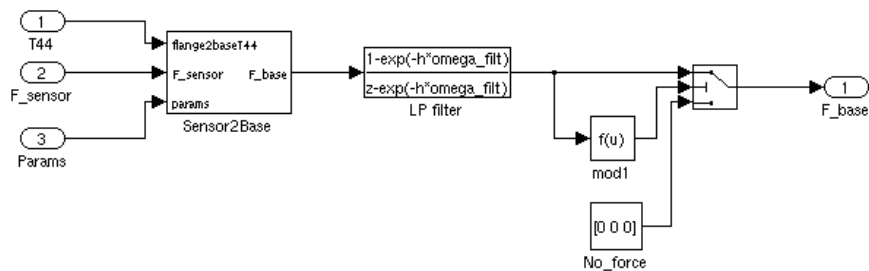


Figure 21: Content in box 2. Force processor: prepares the signal for the controller

A low pass filter is applied to reduce the noise in the reading and a dead-zone is included to assure no force measurement when there is no contact.

A discrete low pass filter is shown in (25). The parameters ω_c and T_s refer to the characteristic cutoff frequency and the sampling period of the system. The filter introduces a certain delay in the force signal so it is important to find a balance between the noise reduction and the introduced delay by using the right cutoff frequency. During the experiments the frequency was 30 and the sampling

period is fixed by the system in 0.004 s (250 Hz). It is possible to modify ω_c in the Opcom while the program is running, the name of the variable is `omega_filt`.

$$G(z) = \frac{1 - e^{-\omega_c T_s}}{z - e^{-\omega_c T_s}} \quad (25)$$

The dead zone outputs no force if the incoming reading is contained between 1 and -1. This prevents errors in the control when there is no real force applied on the sensor. A possible problem with this block is the fact that it removes some real data within the mentioned boundaries but the sensor has an important noise level. The filter should be really restrictive to remove that residual noise but this would distort the signal. So no other option is left with this sensor. Anyway, for the present task this measure is good enough.

- 3rd box:

This part is the core of the program, the impedance controller in itself.

The signals on the left of the box represent the inputs, in this case: force and velocity. The long block is the controller, the internal components are shown in Figure 22 and they perform the operations described in previous chapters.

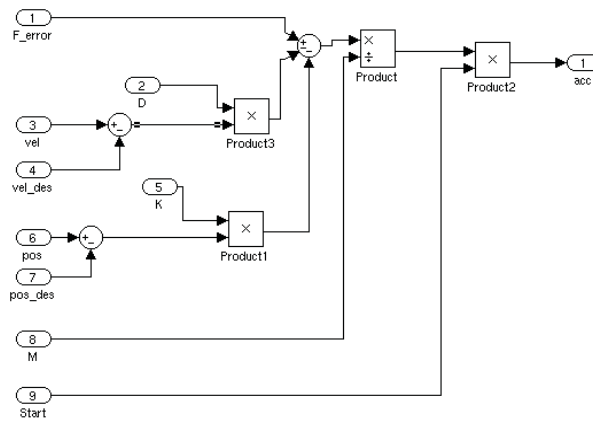


Figure 22: Impedance controller

At this point it is important to explain how the input signals were chosen because they mark the behaviour of the robot. The idea is to follow the edge of a general workpiece. Considering the surface atomically it is easy to see that in each point a force appears perpendicular to the tool as shown in Figure 23. The trajectory has to be reoriented according to this force direction in order to avoid collisions if the shape is irregular.

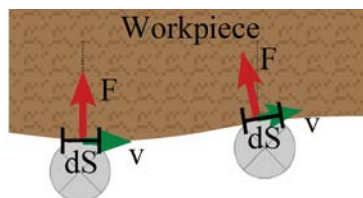


Figure 23: Velocity derived as a vector perpendicular to the force

The filtered force in Newtons and expressed in the base frame is split in two parts: x-y component and z. This is done because the program is designed to move the TCP on a constant xy-plane so the z component should not influence the position. Of course, the user has to take this into account to avoid collisions because if any contact appears in z the robot will not correct its trajectory and may cause an accident. Another possibility could be to introduce z with some penalizing factor that reduces, but not completely removes, its influence on the trajectory. With the x-y force components the modulus and the direction are obtained. The modulus is then compared to the force reference. This reference should have a low value to avoid collisions. Then the error is multiplied by the force direction before being sent to the controller. Notice that the force reference is subtracted from the force modulus instead of the other way around. This is done to include the necessary change of sign in the force coming from the sensor to represent that it is exerted on the environment by the robot.

The desired velocity in the base frame is derived using the force direction, producing a perpendicular vector contained in the xy-plane with a certain modulus that can be chosen externally with *Vel_mod_mms*. Notice that a change from mm/s to m/s must be done with the help of a gain block.

These two signals are fed into the impedance controller. The output is an acceleration that, once integrated to obtain a velocity, will be sent to the robot and back to the controller as velocity signal.

- 4th box:

The blocks contained in this box transform the velocity derived in the controller so it can be performed by the robot. Notice that a vector with (0,0,0) is linked to the linear velocity. This is to avoid the reorientation of the tool so that it only modifies its trajectory linearly. These elements representing the angular velocity are required for the calculation with the inverse Jacobian matrix.

Also a restriction to prevent high velocities can be seen in red color. If the parameters are well stated in the controller this restriction will not be activated.

In addition to the velocity derived in the impedance controller, some others are necessary in order to produce a proper control. They should be added here, so when they are activated by other blocks, they can be sent to the robot.

- 5th box:

Here position and velocity signals are sent to the rest of the program. Matrix T44 from flange frame to base frame, the tool coordinates in the base frame, the Jacobian and the inverse Jacobian are also calculated here.

- 6th box:

This is an important part of the program. Its main mission is to generate signals to activate the different tasks, that is: to reorient the robot until it is placed in the initial position, to start the movement until contact is reached, to store the first point where contact occurred and, finally when the tool is back at the first contact point, to move it back to the initial position. It basically works as a state machine. The whole code with the considered states is shown in Appendix 2.

Three of the output signals are: *vel_out* which controls the robot while it is being reoriented to the initial position, *enable* changes to 1 when the first position is reached indicating that the robot is ready to move towards the workpiece and goes back to 0 when the trajectory has been completed, finally, *gen_trigger* is equivalent to *enable* but remains equal to 1 even when the movement is over. This last flag may be useful for other tasks but it will not be used in this program.

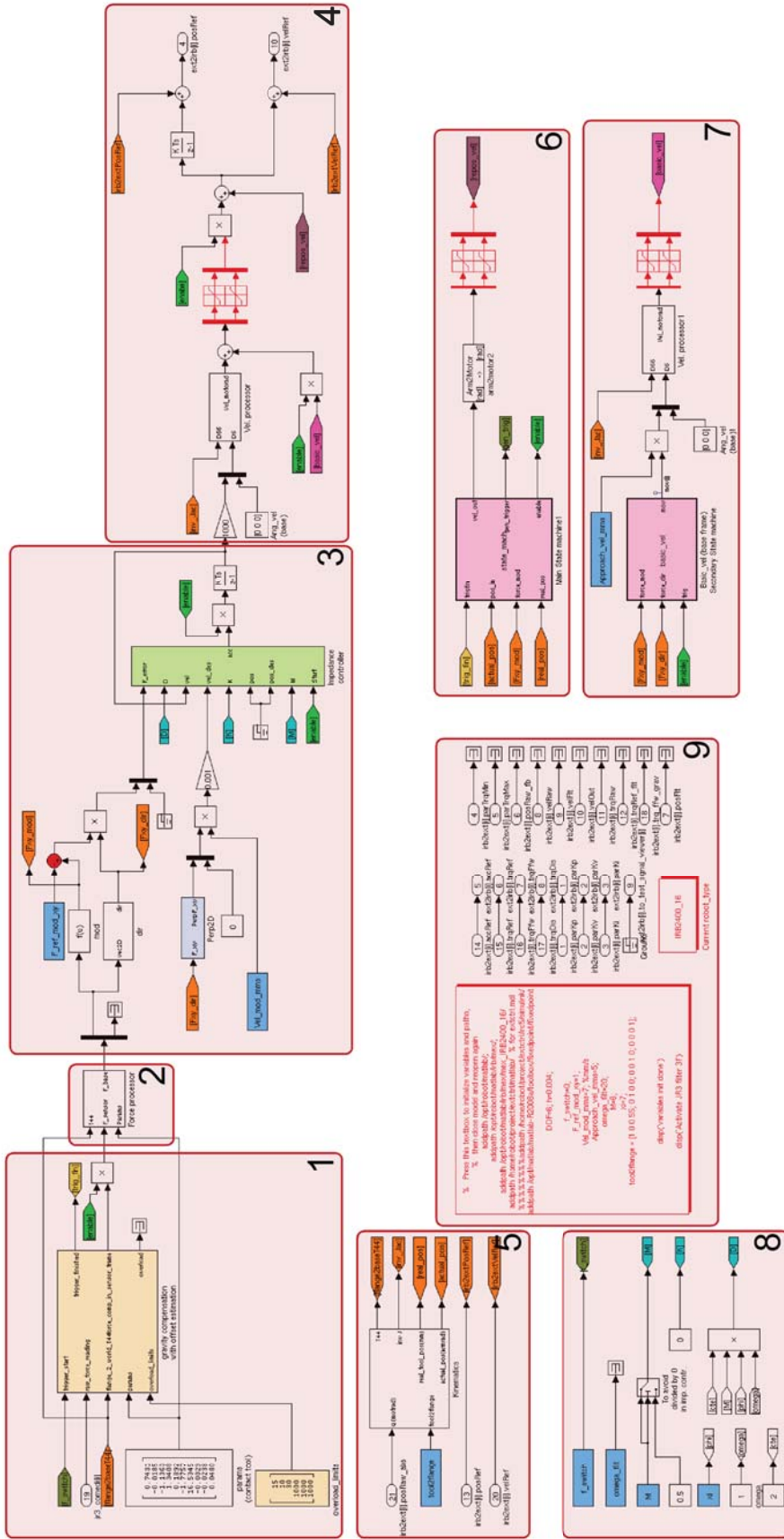


Figure 24: Contact controller

- 7th box:

In this blocks a complement to the previous state machine can be found. It is a subtype which acts when the robot is placed in the initial point and stops when the tool has reached again the first contact point.

It has a big influence in the behaviour of the whole system. The idea is to provide a basic velocity when no force is measured, this means that no force direction can be obtained and therefore the inputs to the impedance controller are zero and the movement is completely stopped.

The concept is similar to hybrid controllers. When no force produces no movement this block generates a velocity that complements the impedance controller.

First of all the block guides the tool towards the workpiece and once a force appears producing the desired velocity this block stops. If any bounce or irregularity such as a corner produces a discontinuity in the contact then this block leads the tool back to the edge. The code included in this block can be found in Appendix 3.

- 8th block:

In this part of the program some variables coming from the Opcom are processed. M, D and K parameters are calculated here depending on the value given by the user. Notice that, as there is no position reference, K will remain equal to zero and D is directly controlled with the value of \dot{x}_i .

- 9th block:

This part is common to all the programs generated in Simulink and contains general inputs and outputs to communicate with the robot and the sensor. The red box has to be executed in the beginning to declare the necessary variables and the path for the Matlab files.

- Activation signals:

The signals that trigger the system deserve special attention. The first one is f_switch , it starts the gravity compensation algorithm but it can not be used to activate other parts.

The gravity compensation takes 50 samples to be initialized and once it is done a flag called $trigger_finished$ changes from 0 to 1. That signal is the one that should be used to start the system.

An important consideration has to be done: the force before the correction has a big offset that goes into the low pass filter and after that into the controller. Apart from the delay due to the inherent calculations of the gravity compensation the filter produces an extra delay. This delay has to be taken into account when the controller is started, otherwise an initial error will be derived. Two possible solutions are: to intentionally include a delay in the $trigger_finished$ signal to cover the one produced by the filter or to multiply this flag by the corrected force in order to avoid any input to the controller before the correction is ready. The best option is the second one because it is not affected by changes in ω_filt which do affect the first solution if the introduced delay is not enough to cover the effect of the filter.

Notice that $trigger_finished$ is only controlling the activation of the block in box 6. In that block the movement toward the initial position, close to the workpiece, is started and when the robot is correctly placed $enable$ and $gen_trigger$ change to 1. $Enable$ is the responsible signal for the control of the hybrid controller formed by the impedance controller and the basic velocity.

The shown problem about the delays does not disappear just by generating *enable* as control signal. If the robot is since the beginning at the correct initial position then *enable* and *trigger_finished* are equivalent and the problem will keep happening, so the solution mentioned is applied: corrected force coming from the gravity compensation block is multiplied by *enable* to prevent force inputs from going into the impedance controller and causing an initial position error.

- Surface program:

Once the execution of the Simulink program is finished a very simple script can be run to obtain a picture of the workpiece and its position related to the base frame. The code can be seen in Appendix 4. The program generates a matrix with those positions where a force was measured so no jumps or discontinuities in the contact are plotted.

8.2 Contact algorithm results

During the different tests carried out to check the program it was necessary to deal carefully with the control parameters: M , xi , $F_{ref_mod_xy}$ and Vel_mod_mms .

The combination between M and xi determines the kind of movement. If M and xi are very small the tool tends to bounce if any contact appears, so the trajectory is hardly continuous. For bigger values of M the influence of the force is smaller. When xi is bigger than one it produces a more stable movement but if the value is too big the transient to reach the force an velocity reference also grows.

$F_{ref_mod_xy}$ should be fixed with care, depending on the material that will be tested. If it is too high the tool will exert big pressure on the surface and may break itself or the workpiece.

Two other parameters that the user needs to settle are $Approach_vel_mms$ and $Vel_mod_xy_mms$. The former has to be defined finding an equilibrium between time reduction for the execution and possible negative effects when the tool is close to reach contact. $Vel_mod_xy_mms$ should not adopt high values to avoid collisions or slips when the shapes are irregular.

In some cases $omega_filt$ can be reduced to introduce less noise in the force measurement, and in this way, reduce the effect in the movement. After all, this part is not about following strictly the references but the shape of the workpiece.

Some initial tests were done without contact with the environment. The algorithm was slightly changed to produce a movement in the same direction of the force so the user could move the TCP by hand and feel the effect of a change in any of the variables. Once the parameters were calibrated to produce a low stiffness and to prevent collisions the program was ready for real interactions.

The experiments are done using a big wooden box placed on a table, see Figure 25. First of all the robot is jogged manually to the desired initial position. The joint angles are introduced in the state machine that can be seen in box 6. That will be the point where the robot will go in order to start the movement. It is important to remember that the program will not correct the position in z or reorient the tool, so this initial point must be chosen wisely in order to assure a right contact with the tool but not with other parts such as the mounting plate for the tool or even the flange. In the same way, when the matrix $tool2flange$ describing the transformation from tool to flange is declared in box 9, the z

parameter is defined as 0 because for this experiment z is considered irrelevant and no effort is made to measure it with precision. If more accuracy in z were necessary then that parameter could be changed in the matrix.

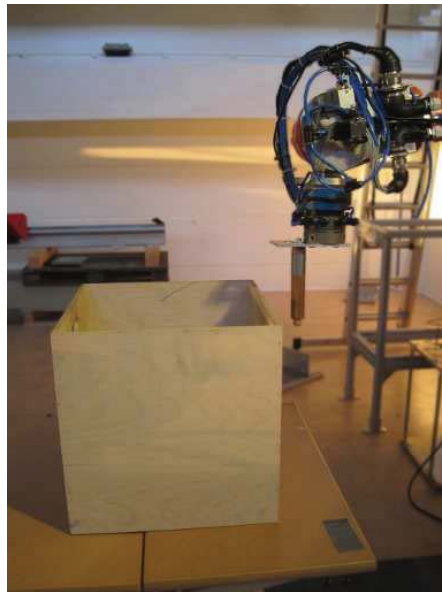


Figure 25: Tool placed in a possible initial position, ready to begin the Contact program.

The execution protocol must be as follows: click on the red box to load the variables in Matlab, then build the Simulink program using the Real Time Workshop tool. Open the Opcom and load the program. Go to Submit, change the necessary values for the test and press Commit. Release the robot brakes by pressing the dead man's switch and change to Obtain.

The final values for the parameters are shown in the next table:

$F_{ref_mod_xy}$	1
$Vel_mod_xy_mms$	9
$Approach_vel_mms$	5
M	20
Ω_{filt}	20
ξ	20

When f_switch changes from 0 to 1 the movement starts. The state machine checks if the position is equal to the initial one to take the tool there. Once at that point the robot will move towards the workpiece in x direction and when it touches the surface the movement will change to follow it. When the trajectory is completed the robot retracts back to the initial position. While the program is on execution the user should not jog the robot using the Teach Pendant or the behaviour may become unpredictable. If the robot loses the contact the program is not ready to recover it when it was due to an external signal.

The state machine that controls the different tasks in the controller is ready to guarantee a good behaviour even when the program is not unloaded for the next execution. This means that the user can change f_switch from 0 to 1 the first time to start the program and once it is finished, with the tool back at the initial position, a new change in f_switch from 1 to 0 and back from 0 to 1 can restart the routine.

Once the test is done it is possible to run a script that plots the shape of the box. The name is surf.m and receives as inputs the Cartesian position of the tool in mm, the flag that indicates the end of the gravity compensation, the force modulus and the tool radius. Notice that it only plots the xy-plane. See Figure 26.

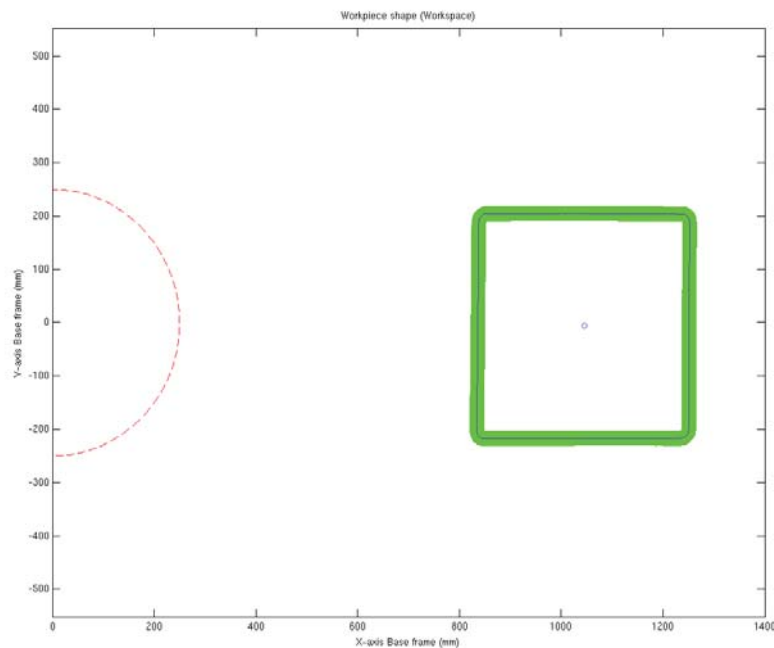


Figure 26: Plot coming from surf.m. The red semi-circle represents the base of the robot, the blue line shows the TCP trajectory and the green thick line represents the silhouette of the tool in each contact point. The point in the center of the box is the geometric center.

8.3 Machining task

This is the most demanding part of the project. High accuracy is required but also security and compliance in the controller.

In order to check the results and before using a real drill or mill bite another tool was used, it consists of a marker. The provided workspace was a box filled with foam to avoid hard collisions. The RAPID program contains the necessary orders to draw a picture on a flat surface such as a paper or a cardboard. If the algorithm is able to control the robot and it produces a precise picture then it is likely that other tools can be used instead just by changing the parameters to adapt them to the requirements.

The Simulink program appears in Figure 27. Most of the blocks do not change with respect to the contact algorithm but some others are included. The boxes indicate the ones that are considered important.

- 1st box:

The variables for the impedance controller are derived here. The main difference with the contact algorithm is that now there is one M for each direction. That is, the force is penalized differently depending on the direction. The idea is to produce a smooth movement in z but more rigid in x and y . This can assure good compliance with the working surface avoiding collision and a reduction of the friction effect in the other two directions.

Other important difference that deserves special attention is the block called M_adapt . The RAPID program with the instructions for the machining task has two main states: when the robot is being reorientated and when the tool is touching the workpiece to produce the machining. The first state implies large movements with no contact, unless it is accidental. The second one is mainly designed for contact. In each case different parameters for the impedance controller are required to avoid trembling and effects coming from the inertia or to adapt the stiffness.

To fulfill these objectives a block that switches between different sets of values is built. In this way, the M_adapt block performs a change in the value of M . When no contact is expected the behaviour of the robot will be stiffer than when the tool is touching the workpiece. This produces better results than the use of a common value for M . The whole code that performs these changes is shown in Appendix 5.

The user is the responsible for the values introduce in each part and must be aware of the consequences if a really high stiffness is settle when the tool is working or, on the contrary, the stiffness is too small when the robot is approaching the working position.

- 2nd box:

This box contains the inputs to the impedance controller. Now the three components of the force expressed in the base frame are used. As in the contact algorithm, the reference is subtracted from the force to indicate the necessary change of sign which shows that the force is done by the robot. On the other hand, desired velocity and position are independent from the force. They come from the RAPID program loaded on the Teach Pendant. They are converted to Cartesian coordinates in the base before being fed into the impedance controller.

Notice that the the angular velocities coming from the RAPID program remain unchanged. They are joined together with the linear velocity derived by the controller. This way the calculation of the joint velocities with the inverse Jacobian is possible and it produces the orientation movements equal to those included in the RAPID code with the linear adjustments produced due to the force influence.

- 3rd box:

The content of this box generates the control signals responsible for starting the controller and draining the integrals when the final position has been reached so the remaining position error is removed. It works in a similar way to the state machine created for the contact algorithm, the code is available in Appendix 6. The blocks in 3.1 and 3.2 are the drains for the integral.

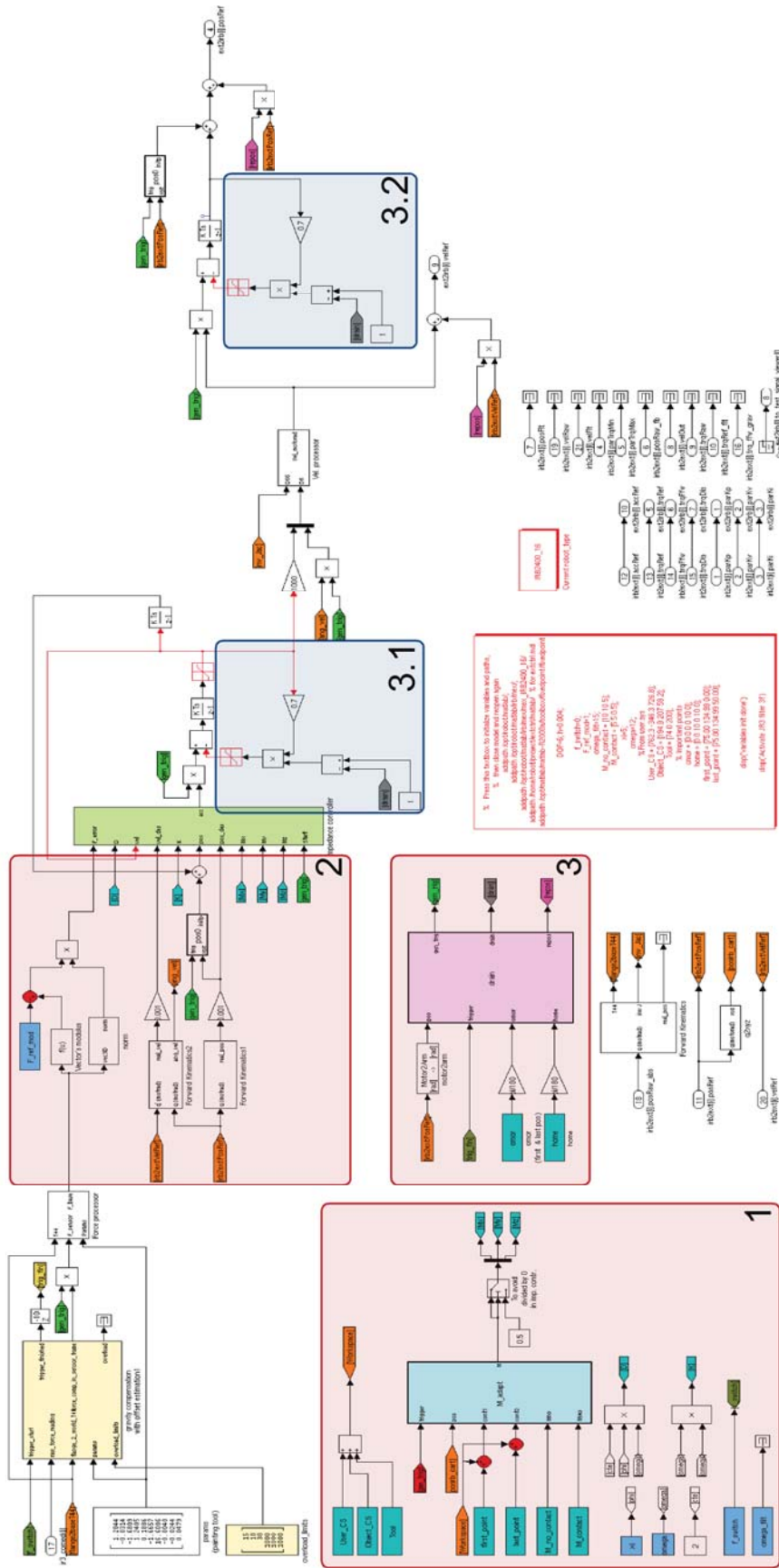


Figure 27: Machining controller

8.4 Machining algorithm results

First of all, notice that a delay has been included in the output flag from the gravity compensation block. When the controller was run the very first time in Obtain mode a small force appeared causing a slight movement. After analyzing the logged data it turned out to be a remaining value of the force before ending the correction. Just to prevent that effect in future execution the signal was delayed ten samples, which does not affect in a significant way the rest of the program.

For these experiments it is really important to calibrate the parameters in the right way. In the beginning the block responsible for the changes in the variables was not included. During the execution there is a part before the robot reaches the working position where the joints are reoriented, it was common to see vibrations. To reduce that effect ω was increased and the effect was reduced but not completely removed.

The problem was caused by some forces appearing due to the movement while there was no real contact with the environment. By increasing ω the controller was giving priority to the position reference but when the tool was in contact with the surface the behaviour was too stiff. Another option was increasing M to reduce the effect of the force, but again, the compliance with the contact surface was reduced.

To solve this problem about adaptability the M_adapt block was created. The algorithm included there only modifies the M , making it bigger while the robot is not close to the contact position and decreasing it for the machining tasks that need less stiffness. The program which controls the change could be easily adjusted to include as well ω or ξ in order to obtain even better results.

Concerning M it is also important to mention that very small values could lead to vibrations, so even when there is contact between the tool and the environment it is not advisable to reduce it under 0.4-0.5.

Other block that was included after some experiments is the one that drains the integrals to remove the position error. When the program is launched it checks if the position is equal to the initial position called $omor$ in the RAPID code. If it is different, the program waits until the robot reaches the right position and stores it. When the RAPID code is finishing and the robot has to go back to $omor$ the drain algorithm removes the position error accumulated during the execution and moves the robot to the right final position.

In order to obtain a proper control over the robot it is important to change f_switch from 0 to 1 before starting the RAPID program on the Teach Pendant.

The values for the different parameters appear in the next table:

F_ref_mod	0
$M_no_contact$	[10 10 5]
$M_contact$	[5 5 0,5]
ξ	5
ω	10
ω_filt	15

Although for the experiments only a marker was used it is important to understand the effect of each parameter and their changes in the behaviour of the system. In a regular machining task high stiffness is required so probably M will have big values to reduce vibrations. Ω will rule the movement imposing the position to obtain accurate shapes and it will have high values too. Other options to make the controller more flexible is to separate each variable in three component to regulate x, y and z independently.

In the next pictures some outputs from different tests performed in the lab can be seen.

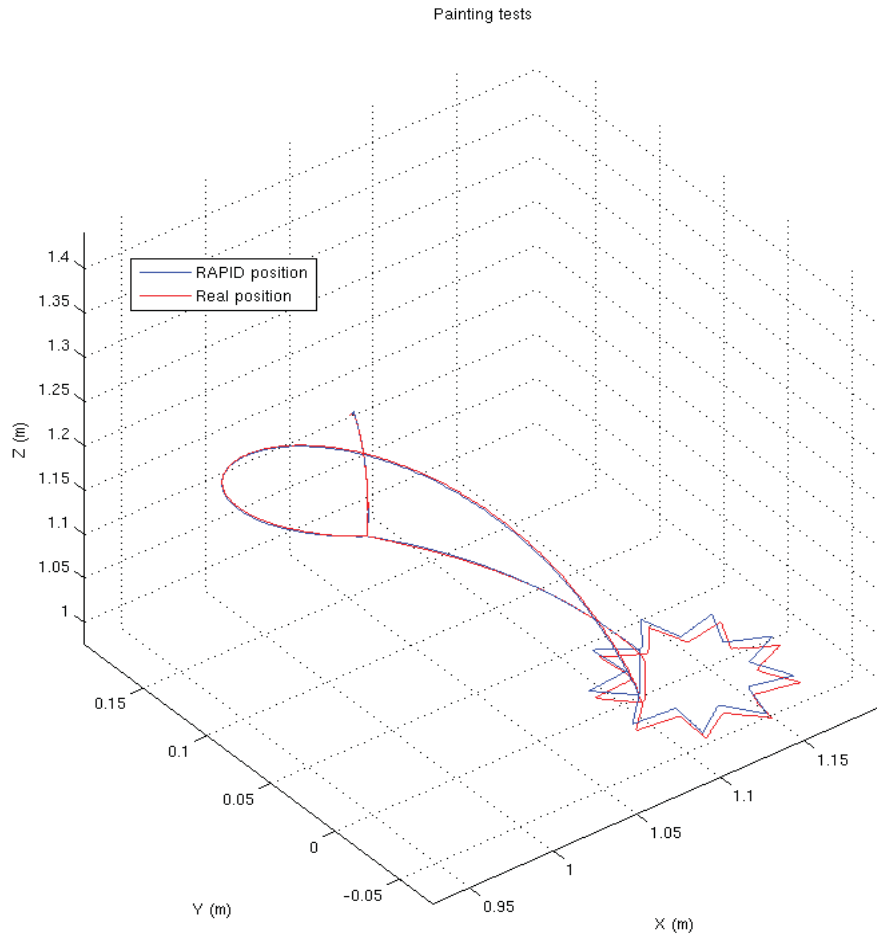
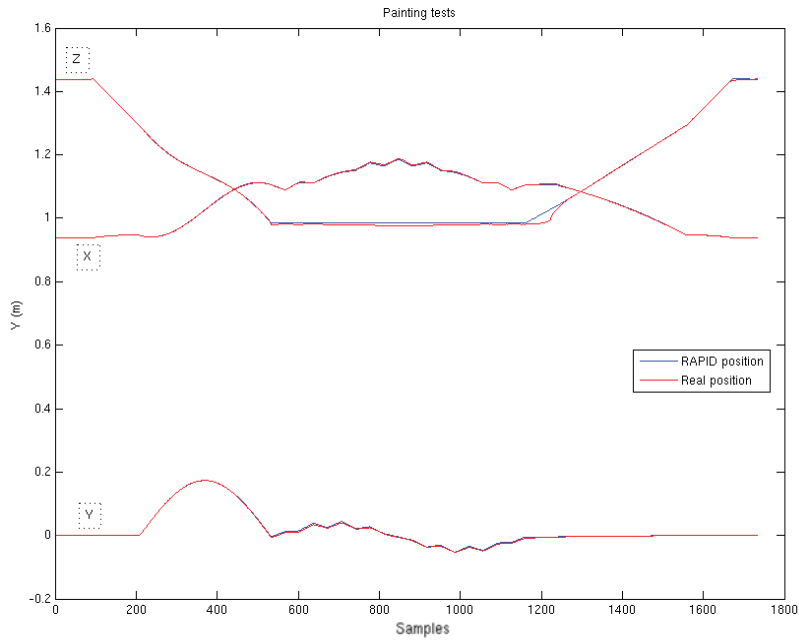
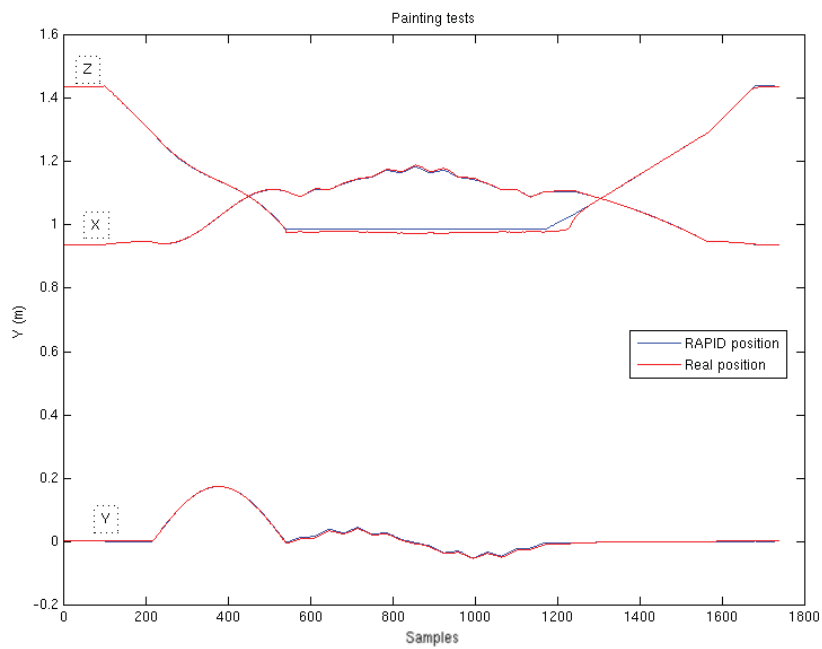


Figure 28: Three-dimensional path provided by the RAPID program and the real one. Real contact achieved.

In Figure 29 two different results from some tests are plot. The first graphic shows that smaller values of F_{ref_mod} act reducing the position error. On the second plot it is possible to appreciate that lower values of M in x and y when the contact has been reached introduce a bigger error in the position. This is because the force term is increased when M is small and, therefore, the effect on the controller output is bigger which produces more error and distortions.



(a)



(b)

Figure 29: (a) Path for low variable values of force reference: 0.5, 0.2 and 0.
(b) Path with variations in $M_{contact}$ for x and y: from 5 to 3.

9. Conclusion

After many experiments it is possible to conclude that the system reaches the desired behaviour. The controller is able to perform the necessary corrections if the variables are well calibrated. It is important to remark the relation between the material and the value of the parameters. It is highly recommended to run several tests before starting the production in order to optimize as much as possible the regulator.



Figure 30: Pictures of stars resulting from several tests

The two main programs developed along this Thesis are based on an impedance controller but each one uses different aspects. This shows the big flexibility of the regulator to carry out diverse tasks. The imitation of a physical system such as the mass-spring-damper led to proper results, leaving a door open for other possible controllers inspired by other complex systems.

The work done during this Thesis could be used to generate more sophisticated tasks, for instance: integrating artificial vision and developing a method to convert the treated picture into a RAPID code or similar the robot could act as a human painter.

Of course, many improvements can be done. Some of them could be the correction of the force measured due to the movement or a higher degree of automation. This means create a protocol to integrate all the steps carried out during this work in order to reduce the need for user intervention. By doing this the program could receive as input only the Pro/ENGINEER file and then generate the whole postprocess, error correction, etc. by itself.

Other option could be including human interaction. This way an operator could guide the robot to some key positions, run the machining program and then

place it in the next position. Consequently the robot can be understood as a tool and no longer as the main performer of the machining task.

Finally, an extra task could be added to this chain of steps to close the whole process. As the project started with the idea of creating a substitute for a CNC machine a natural step would be to include an extra process in order to check the results after the machining procedures. It is possible to find in the market sensing devices to obtain accurate measurements of the finished surface. Integrating these 'feelers' or palpators with the robot the productive cycle would be completed, that is: design of the model, manufacture and inspection.

References

- [1] Barrientos, A. *Fundamentos de robótica*. McGraw-Hill, 1999.
- [2] ABB: www.abb.com
- [3] ABB IRB 2400 data-sheet.
- [4] JR3 sensor: <http://www.jr3.com>
- [5] Pro/ENGINEER: www.ptc.com/products/proengineer/
- [6] MATLAB: <http://www.mathworks.com/>
- [7] Isolde Dressler. *Force control interface for ABB S4*. LTH.
- [8] Siciliano, B; Khatib, O. *Springer Handbook of Robotics*. Springer, 2008.
- [9] Siciliano, B; Sciavicco, L; Villani, L; Giuseppe, O. *Robotics: Modelling, Planning and Control*. Springer, 2009.
- [10] ABB Automation Technologies AB, Robotics. *RAPID Reference Manual*, 2004.
- [11] Parametric Technology Corporation. *Pro/NC Topic Collection*. Pro/ENGINEER, 2001.

Appendix A: Pro/ENGINEER guide

This Appendix will show roughly how to obtain a G-code for a CNC machine using the software Pro/ENGINEER. The purpose is not to make a complete manual for the program but to draw some guidelines about some options, menus, etc. In order to get a deeper idea about Pro/ENGINEER many tutorials can be found with a brief search on the Internet thanks to its popularity in industry. The specific version used here is Pro/ENGINEER Wildfire 4.0. Some options or menus may change from one version to another.

It is important to emphasize the particular philosophy that Pro/ENGINEER uses. In the beginning the program could seem difficult because of its menus and options. It is prepared to work with closed tasks. This means that, unlike other programs, one task has to be finished before continuing to the next one and most of them are ordered rigidly so the user has to follow a fixed workflow.

When a task is completed the user has to press a check mark in the *Dashboard* or the *Done/Return* option in the *Menu Manager* to close that step and keep moving forward to the next one.

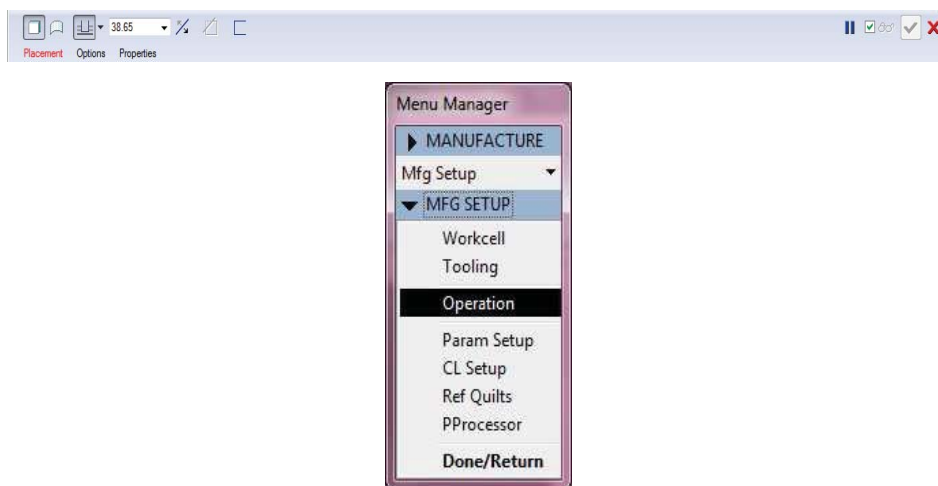


Figure 31: Dashboard and Menu Manager

Some parameters or properties have to be defined along the process. Once the program has led the user to the point where a setting is declared and this task is closed it is possible to change the value by coming back. The other way around is not allowed. It is very advisable to get a good idea of what is done in each stage of the program in order to make future changes easier.

With the aim of making this chapter easy to understand a generic part will be created, see Figure 32. This way the different steps and its order can be seen. For more complex designs other additional options of the program can be used depending on the requirements. Extra options can be found in [11].

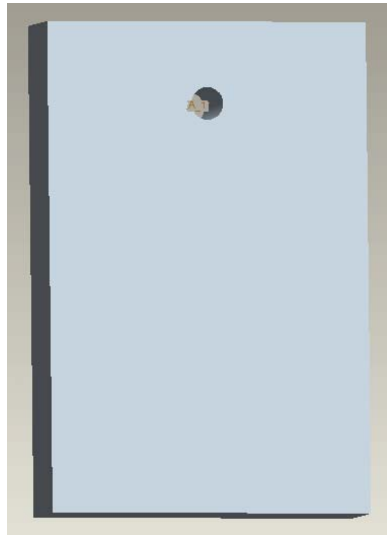


Figure 32: Generic model used along this Appendix to describe the procedure

A.1 Creating the file and adding the model

First of all one file containing information about the desired final piece or part is needed. These files have extension .prt. While working with Pro/ENGINEER this definitive part is also called *Reference Model* since it will be the base for all the work.

Open a Pro/ENGINEER window and set the working directory. Now create and name a new manufacturing file with the subtype NC-assembly. It is advisable to set the units for the file by unchecking *Use default template* and selecting the right option in the window that appears after pressing OK.

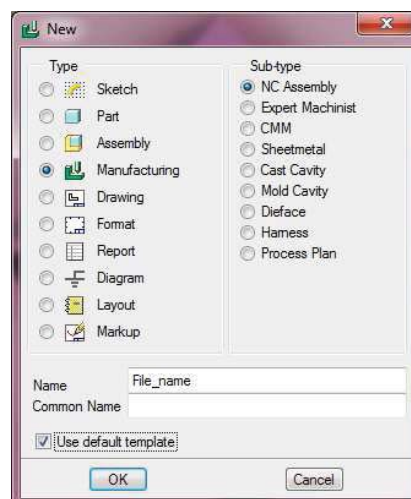


Figure 33: Creating a new file

The next step should be adding the *Reference Model* to this new file. On the *Menu Manager* press *Mfg Model*, then *Assemble* and *Model piece*. Choose the

proper .prt file. Other options are to create a new model instead of using a previous one or even create a workpiece. This last option will be shown later.

When the model appears in the screen and in the Model Tree the program will ask the user to locate it in the workspace. The option *Default* can be selected. The other options in the menu are used when more parts are necessary and they have to be placed in relation to each other.

The menu *Create Reference Model* arises. Choose *Same model* and press *OK*. This will produce a connection between the model and the manufacturing operations: if the initial model is modified the operations will change. The other two options can be selected when working with more parts that need to be linked and, maybe, modified.

Press *Done/Return* to finish this first stage.

A.2 Creating the workpiece, milling volume or milling window

When the model is created and depending on the shape and operations required in the machining process it is possible to provide the program with three types of information: an approximation of the raw block of material that will be used, some 'virtual' milling volumes and surfaces or a silhouette that represents the projection of the model. All three alternatives will be explained and the designer can pick the best option according to the desired final part.

2.1. Option 1: Adding milling volumes or surfaces

This option is easy to use but complex models can require the creation of a big number of 'extra' elements. It also can produce 'air milling', which represents a waste of time in the production if the shape of the milling volume is not accurate with respect to the real aspect of the raw block. Other disadvantage can arise if future changes need to be done because every milling volume or surface has to be modified individually.


A milling volume represents a virtual amount of material that the tool will remove in order to produce the model. A milling surface defines a plane that needs to be faced or shaped. Milling volumes or surfaces can be created right after introducing the reference model or in each step when they are necessary.

-Milling volume  :

Press *Milling volume*. Different options become available. In this example an extrusion for the hole will be created. Choose *Extrude* and then the *Placement* tab on the *Dashboard*. Select the frontal face of the model as *Sketch plane* and the right surface as *Reference*. Click on *Sketch*.



Figure 34: Placement tab on the Dashboard for the extrude option

The part is reoriented and the program allows to draw the edges of the volume. For instance, to create the milling volume of the hole press . In *Type*

window choose *Single* or *Chain*, select the edges of the hole and press *Close*. Now quit the sketch application by clicking on the blue check mark.

Now the extrusion is represented in yellow. If its direction is not right click on the yellow arrow that appears in one of its ends. The depth can be defined with the length in number or by selecting the surface that acts as end. This last option is available in the *Options* tab.

Once the volume has the desired shape press the green check mark on the *Dashboard* to close the extrude window and the other green check mark in the side toolbar to finish the milling volume. The new milling volume will be displayed in the Model tree.

- Milling surface  :

The process is similar to the previous one. Press *Milling surface*. Now a top surface will be created to produce the face milling. Again choose *Extrude* and then the *Placement* tab on the *Dashboard*. In order to create a plane covering the frontal surface of the model the *Sketch plane* must be a side face and the *Reference* can be any other surface. Click on *Sketch*.

Once the sketch window is shown press  and select the edge formed by the top surface and the side surface. Exit by clicking the blue check mark.

Make sure that the yellow arrow points towards the body of the model and correct the size of the new surface by introducing the length in numbers or with the *Options* tab.

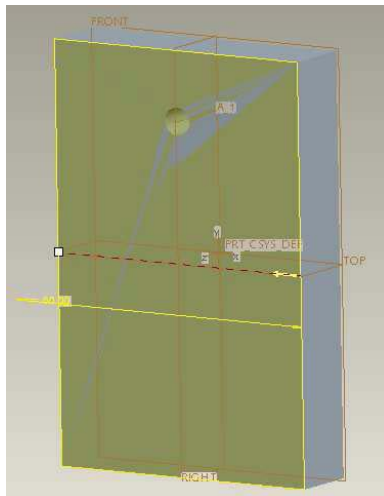


Figure 35: Milling surface created to produce a face milling

When all this is done finish the extrude process with the green check mark on the *Dashboard* and the *Milling surface* process with the green check mark on the side toolbar.

2.2. Option 2: Adding a workpiece

This option is useful when the amount of removed material has to be considered or the shape of the model is complex. Unlike the option of the milling volumes

when a workpiece is added it can easily be modified and all the elements that refer to it will change automatically. When the workpiece is included in the program it will always be considered in all the machining processes as a tangible element. This can avoid problems if the user forgets to define a milling volume.

The workpiece does not generate by itself the space to be machined, it only acts as a handy template. So it is necessary to declare what parts of this workpiece will be processed by combining them with milling volumes, surfaces, mill windows or directly using its faces and edges in the machining sequence.

In order to create a workpiece many alternatives are available. It can be added and placed in the same way that the reference model was introduced previously. Other possibility is to create a new one pressing on *MFG Model* then *Create* and *Workpiece*. The steps are quite straightforward and similar to the creation of an extrusion so they will not be explained in detail.

In this example a workpiece like the showed in Figure 36 will be used. Only the material on the top face and the hole will be milled.

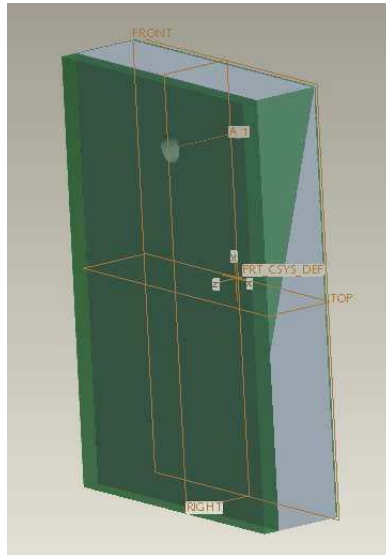


Figure 36: Workpiece containing the Reference Model


Other advantage of working with a workpiece is that it allows the use of a graphical application that shows how the raw material is removed and how the volumes are machined.


2.3. Option 3: Using mill windows


This is the easiest option. If the parameters are stated wisely it can generate in just one task the whole machining process. The combination with a workpiece can be very useful too. One possible disadvantage is that 'air milling' can occur, as with milling volumes.


A mill window works as a 'virtual bell' that covers the model. The projection of this bell over the model produces the space that will be machined. In other words: every surface within the contour of the window will be milled. This

allows to obtain in one single task all the milling processes necessary for volumes and surfaces.

To create a mill window press . Some options become available on the *Dashboard*:

- Silhouette window type : It is a very useful option. The main idea is to obtain the silhouette of the model and its volumes and use it as the contour that the tool will follow. In the *Placement* tab the option *Keep Inside Loops* can be unchecked. By doing this the program considers all the holes, cavities, volumes, etc. within the mill window and therefore they will be machined. In the *Options* tab an offset around the contour can be added to make the mill window bigger.

- Sketch window type : This option is very flexible. The user can draw the mill window using the sketching tools.

- Chain window type : This tool can be used to create the mill window using edges from a closed chain.

Other options that can be defined are the depth of the mill window and also the bounds for the tool. The tool can run strictly inside the window contour, on the contour or outside. This configuration is very important and has to be considered depending on the tool diameter. The tool path may be strongly affected by this parameter.

A.3 Operation setup

In this step the user can define the environment for the manufacturing operations, that is: tool machine, coordinate systems, fixture, etc. It is also possible to do this while defining the operation sequence.

On the *Menu Manager* select *MFG Setup* and then *Operation*. The *Operation Setup* window appears. Name the operation that will use that tool machine or leave the default name. The options with a red arrow have to be defined. The fixture is not required but it could be declared here in case it is necessary.

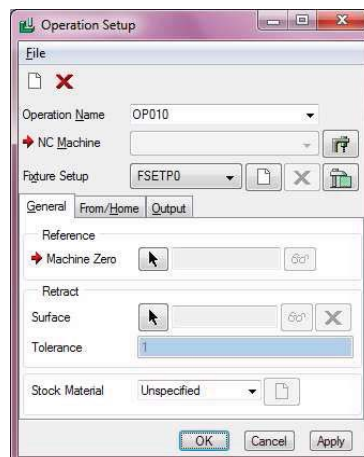




Figure 37: Operation setup window

Press  to enter the *Tool Machine Setup*. Name the machine and select the type. Other options can be provided if necessary such as spindle, units, travel of the tool, etc. For this Thesis the machine type and number of axis are enough. For the example a milling machine with three axes is chosen. Save the changes and close the window by selecting *OK*.

Now the *Machine Zero* has to be defined. The code generated for the tool machine indicates in every step where the tool has to be and always refers to this coordinate system. It is common to place it on one corner of the workpiece or the model. It is also common to find it in the center of the top surface.

In most of the tool machines the Z axis points upwards, X axis points to the right and Y axis points backwards, into the body of the machine. This has to be taken into account for the future because the robot will need this reference and it has to be coincident with the one defined here.

Select the black arrow and choose a coordinate system in the workspace. If none of the systems are right, a new one can be created now. Choose *Coordinate System*  on the *Datum* toolbar. To place the origin on one corner of the part the three surfaces that converge in that corner have to be selected. Pressing Ctrl allows to add more than one surface to the *Reference* field. Once the new coordinate system has appeared it must be oriented in the *Orientation* tab so the Z axis points upwards and X and Y axis are coincident with the edges of the part. When the origin is created accept by pressing *OK*. This new coordinate system can now be chosen to be the *Machine Zero*.

A.4 Machining sequence:

In this step the program already has the model to be created, the surfaces and volumes that will be machined, the tool and the CNC-machine. Now it is time to define the exact process.

For this example the objective is to flatten the top surface of the model and machine the hole. Some alternatives will be presented depending on the different methods to define the surfaces, volumes, etc. The possible strategies are presented in Figure 38. The user can choose depending on the preferences and the characteristics of the model.

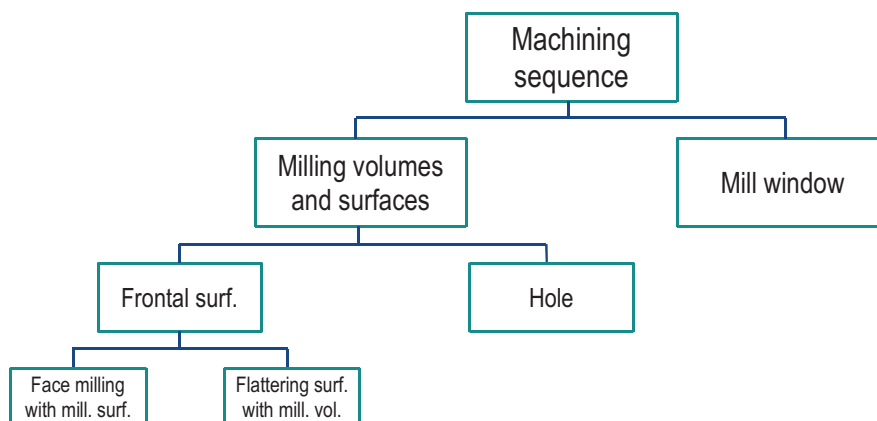


Figure 38: Alternatives available for the machining sequence

On the *Menu Manager* press *Machining* and choose *NC sequence*. If this is the first operation a long list of possible *NC processes* becomes available, if other tasks already exist then choose *New Sequence*.

In general, when one task from the list is chosen and *Done* is pressed the program leads the user to the *Setup Sequence* list. Many options can be checked there in order to completely define the process and make it as flexible as possible. Some options are compulsory and checked by default and some others can be added.

Alternative 1

Task 1: Milling surfaces and volumes

Option 1.1: Face milling using a mill surface

First, the face milling task is configured. This can be done for a raw block with the same thickness of the final part because the process only levels a surface. Select *Face* in the *NC sequence* list and click on *Done*.

The basic options required in the *Setup Sequence* list are: *Tool*, *Parameters*, *Retract surface* and *Surfaces*. Other parameters can be selected to specify further information of the process.

When the user presses *Done* one window appears that allows to state the parameters for the first option checked in the list. When this window is closed by pressing *OK* a second window for the next parameter arises and so on until all the options checked in the *Setup Sequence* list have been fully defined.

For the face milling process the displayed windows are, in order:

- Tool: Description of the specific tool, such as mill bits, drill bits, milling cutters, etc., that will be attached to the machine. Many tools can be introduced and included in a list so that they can be used for different tasks.

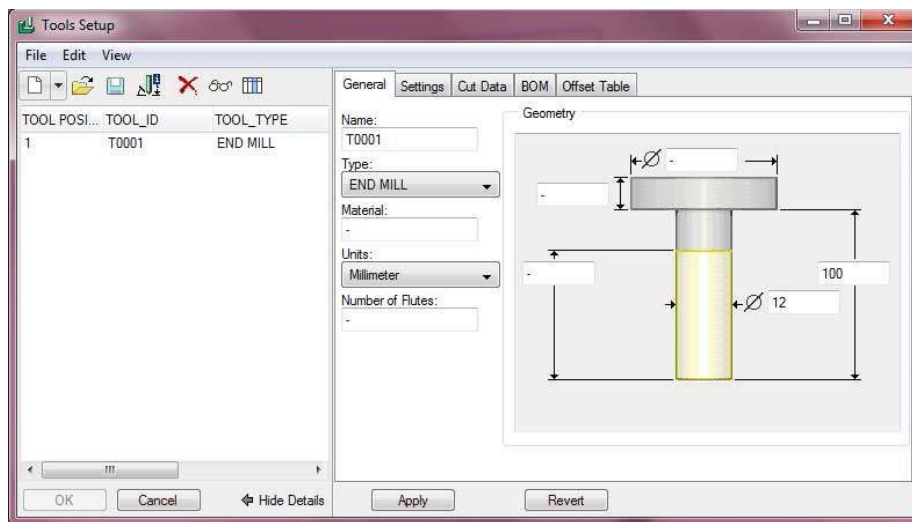


Figure 39: Tools setup window. Parameters for each tool can be chosen on the right . By clicking on Apply the new tool appears on the left.

- Parameters: This window contains a table with precise aspects of the machining process. Some values are set by default but some others need to be introduced (they are shown as -1 or in yellow color). Notice that the units for each box depend on the ones declared while configuring the tool machine. Most of the fields show a picture when they are selected to make them understandable, so they will not be explained here.

In this Thesis this step has capital importance. The precision of the movement, speed, depth and other characteristics for the movement of the robot depend directly on what the user sets here.

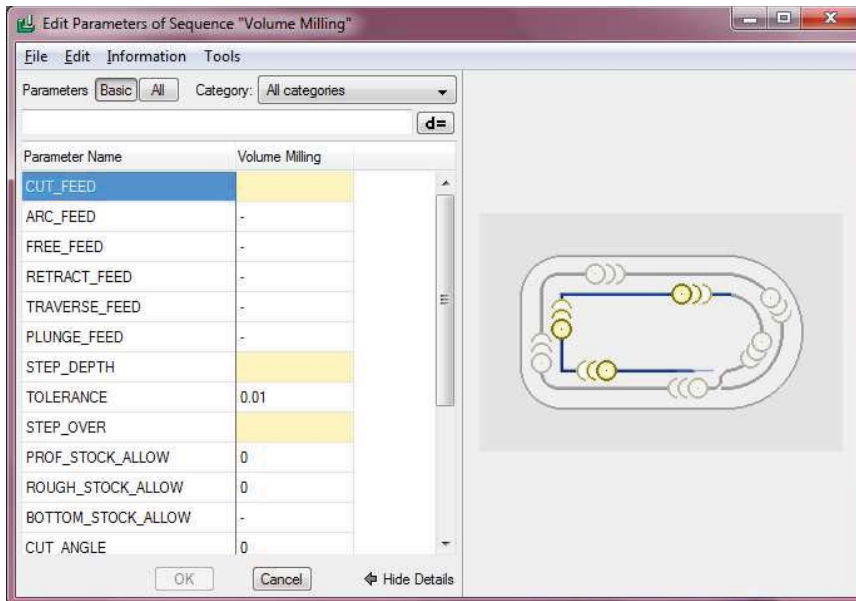


Figure 40: Parameters menu. Some of the parameters declared here will have affect the RAPID code for the robot

- Retract surface: A plane parallel to the work surface where the tool is placed before starting the process and once it has finished. It is usually located at a certain distance from the top surface in the Z axis.

- Surfaces: Now the surface that will be machined has to be selected. If it has been created before as a mill surface the user can choose it from the model tree. Other options are to create a new mill surface or select one plane of the model parallel to the retract surface.

Option 1.2: Flattening a surface using milling volume

If the raw block of material is thicker than the final part it is possible to remove the excess. As it was explained before this can be done just by using a milling volume or in combination with a workpiece.

This *NC sequence* can be configured in the same way that face milling was done previously but now *Volume* has to be checked in the *Setup Sequence* list instead of *Surface*, the other options remain the same.

Set the *Tool*, *Parameters* and *Retract surface* as before and choose the milling volume to be removed from the top of the model. Again, if the milling volume has not been created in earlier steps it can be defined now.

Task 2: Create the hole.

The hole included in this example could be drilled or milled. To drill it, choose *Holemaking* in the *NC sequence* list and follow the steps. Otherwise, to mill it, select *Volume* as it was shown before to remove the excess of material from the top surface.

When it comes to tool definition some considerations have to be done: if the hole is drilled the size of the drill bit has to be consistent with the diameter and depth of the hole. In the same way, to mill the hole the mill bit needs extra space in order to move and produce the cavity with the required dimensions.

Alternative 2: Mill window.

If the alternative chosen is the mill window instead of surfaces or volumes the program can generate in just one task all the necessary movements. For the example the mill window needs to be shaped as the silhouette of the model without keeping the internal loops and placed over the frontal surface, in some cases the depth needs to be specified from the mill window to be the bottom of the model.

In order to create the operation *Volume* has to be checked in the *NC sequence* list. And then, after clicking on *Done*, the user will select *Window* instead of *Volume* (this options are mutually exclusive). The last step is to indicate the window to be used for the process.

Pro/ENGINEER produces the machining of the space within the milling window. Notice that if a workpiece is included in the file and the mill window is not coincident with it the material that would be removed is the one contained in the workpiece covered by the mill window. This tries to avoid 'air milling'.

A.5 Tool path and NC code

Finally, when all the NC sequences required have been defined it is possible to see how the tool path will look like, make some changes if it is necessary and generate de G-code understandable for the tool machine.

On the *Menu Manager* choose *Machining* then *NC sequence* and there select the created process to be checked. Some new options are available there, click on *Play Path* and then *Screen Play*. A video player appears that shows the tool and how it will move in order to machine the model. If any property needs to be changed it can be done by closing the video player and clicking on *Seq Setup*. There the user can check again in the *Setup Sequence* list the parameters that wants to change and modify them.

If the tool path is right and no change is necessary then the G-code can be stored in a new file. This information is also known as CL data or Cutting Location data. On the *Menu Manager* press *Machining* and then *CL Data*. *Output* has to be selected and then, on the Model Tree, the desired operation. In order to include more than one operation in just one file the option *Select Set* is also available. In the example showed here, if a mill window is used only the milling

task that uses this mill window is selected but if mill surfaces and volumes were created instead of the window at least two operations will be necessary to obtain the whole process. Obviously, depending on the shape of the model this can change.

The video player that shows the tool path is brought up. It is a good praxis to check the *Compute CL* option in order to recompute the process and any possible change. In the *File* tab press *Save as*, name the G-code file and press *Ok*. This way the code is saved with extension *.ncl.1* and it can be read or edited in a common text editor.

Another option to create a file containing the G-code of one single task is to press *File* and *Save as* when the video player is displayed in the last step of the creation of that NC sequence.

The files coming from Pro/ENGINEER containing the G-code have to be postprocessed depending on the tool machine that will be used in the workshop. For this Thesis the postprocessor will allow the robot to understand the commands and move the TCP in the proper way.

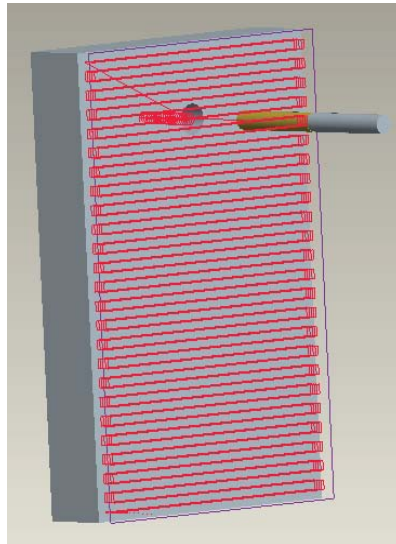


Figure 41: Tool path obtained to mill the frontal surface and the hole of the model

Appendix B: State machine

State machine code used in Contact algorithm. Graphical interpretation of the states and the necessary conditions for the transitions also included in Figure 42 and Figure 43:

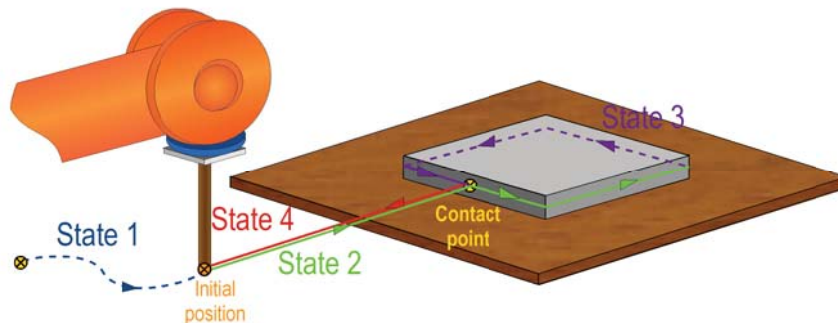


Figure 42: Graphic description of the states used in the State machine code

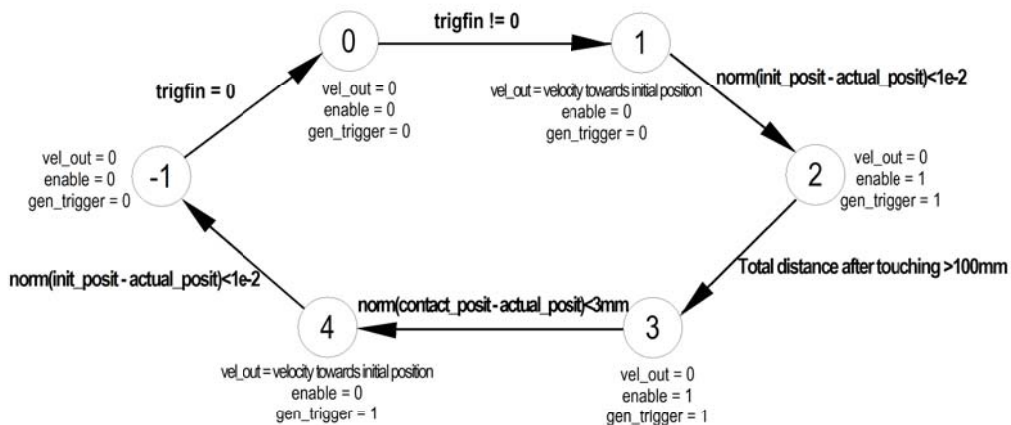


Figure 43: State outputs and transitions declared in the State machine code. Once the trajectory is finished and the state is -1 again the user can restart the program by changing f_switch from 1 to 0 and then back to 1 in the Opcom.

```

function [vel_out, gen_trigger, enable] = state_mach(trigfin, pos_in, force_mod, real_pos)

persistent state
persistent initial
persistent contact_pos_rad
persistent contact_pos_mm
persistent tot_distance
persistent delta_dist
persistent p1

if isempty(state)
    state = -1;
end

if isempty(initial)
    % Arm-rad (pos TCP= [732,6 0 1359] = [0 -10 0 0 90 0]) , to make things easy and
    % depending on the workpiece, the orientation of the joints can be obtained by
    % jogging manually the robot to the desired initial-position and changing here
    % the joint angles read from the pendant, [0; -11; 8.9; 0; 81.1; 0] = [732,5 0 1238,3]
    initial = [0; -11; 8.9; 0; 81.1; 0] * pi / 180;
end
  
```

```

if isempty(contact_pos_rad)
    contact_pos_rad = zeros(6,1);
end

if isempty(contact_pos_mm)
    contact_pos_mm = [0;0;0];
end

if isempty(tot_distance)
    tot_distance = 0;
end

if isempty(delta_dist)
    delta_dist = 0;
end

if isempty(p1)
    p1 = 0;
end

gen_trigger = 0;
enable = 0;
vel_out = zeros(6,1);
vel = 5*pi/180; % speed rad/s (= 5deg/s)

% ---- STATE DEFINITION ----

if trigfin == 0 && state == -1
    state = 0;

elseif state == 0 && trigfin ~= 0
    state = 1; % End of the gravity comp. ready to move to init-position

elseif norm(initial-pos_in) < 1e-2 && state == 1
    state = 2; % Start impedance controller

elseif tot_distance > 100 && state == 2
    state = 3; % Approaching again the initial position after moving 100mm

elseif norm(contact_pos_mm-real_pos) < 3 && state == 3
    state = 4; % If the TCP is inside a circle of 3 mm around the initial contact
               % point then finish the rotation around the workpiece

elseif norm(initial-pos_in) < 1e-2 && state == 4
    % If after the whole execution the robot is back at the initial position then
    % finish the program by sending it to the beginning
    state = -1;

    % Reset all the values just in case the executions starts again
    contact_pos_mm = [0;0;0];
    contact_pos_rad = zeros(6,1);
    delta_dist = 0;
    tot_distance = 0;
    p1 = 0;

end

% ----- STATE OUPTUP -----

switch (state)
case -1
    gen_trigger = 0;
    enable = 0;
    vel_out = zeros(6,1);

case 0
    gen_trigger = 0;
    enable = 0;
    vel_out = zeros(6,1);

case 1 % Approaching the initial position
    vel_out = min(abs(initial-pos_in), vel).*sign(initial-pos_in);
    gen_trigger = 0;
    enable = 0;

```

```

case 2 % Approaching + contact
gen_trigger = 1;
enable = 1;
vel_out = zeros(6,1);

if force_mod > 1 && norm(contact_pos_rad) == 0
    contact_pos_rad = pos_in;
    contact_pos_mmm = real_pos;
end

case 3 % After touching but before ending the trajectory
gen_trigger = 1;
enable = 1;
vel_out = zeros(6,1);

case 4 % Algorithm finished: coming back to initial position
gen_trigger = 1;
vel_out = min(abs(initial-pos_in),vel).*sign(initial-pos_in);
enable = 0;
end

% Counter for the distance to make sure that the trajectory is started and far
from the initial point
if norm(contact_pos_rad) ~= 0
    pos_prev = p1;
    p1 = norm(real_pos - contact_pos_mmm) ;
    delta_dist = abs(p1-pos_prev);

    tot_distance = tot_distance + delta_dist;
end
end

```


Appendix D: Surf.m

Program surf.m to plot the position of the workpiece:

```
function box = surf (modul,position,trig,radi)

% Inputs: modul = force modulus (Fmod), position = position of the TCP in
% mm (real_pos), trig = end of the gravity correction (fin_trig)

a = 1; % Row counter for the new matrix containing the positions where contact
      % was found
force = trig .* modul; % Removes the values of force without gravity compensation
temp = zeros(length(modul),3); % Temporary storage matrix

% Temp receives those positions where some force was detected
for i=1:length(modul)
    if force(i) ~=0
        for j=1:3
            temp(a,j) = position(i,j);
        end
        a = a+1;
    end
end

box_tmp = zeros(a-1,3);

% Zeros in the last positions of temp are removed
for i=1:a-1
    for j=1:3
        box_tmp (i,j) = temp (i,j);
    end
end

box = box_tmp;

% ---- Generates the base of the robot as a circle to plot it ----

b= -pi/2:0.01:pi/2;
rob_base= zeros(length(b),2);

for i=1:length(b)
    rob_base(i,1) = 250*cos(b(i));
    rob_base(i,2) = 250*sin(b(i));
end

% ---- Plots circles with the tool diameter at each contact point ----

cgx = sum(box_tmp(:,1))/length(box);
cgy = sum(box_tmp(:,2))/length(box);

ang = 0 : 0.01 : 2*pi;
dot = zeros (length(box_tmp)*length(ang),2);
c = 1;
block1 = zeros(length(ang),1);
block2 = zeros(length(ang),1);
interna = zeros(length(box_tmp),2);

for i = 1:length(box_tmp)
    for j = 1:length(ang)
        dot(c,1) = box_tmp(i,1) + radi*cos(ang(j));
        dot(c,2) = box_tmp(i,2) + radi*sin(ang(j));
        block1(j)= dot(c,1);
        block2(j)= dot(c,2);
        c = c + 1;
    end

    [x,y] = min (diag);
    interna(i,1)=block1(y);
end
```



```

        interna(i,2)=block2(y);
end
% ---- Graphic ----
    figure;
    plot(dot(:,1),dot(:,2),'g');
    hold on;
    plot(cgx,cgy,'o');
    hold on;
    plot(box(:,1),box(:,2),'b');
    hold on;
    plot(rob_base(:,1),rob_base(:,2),'r--');
    axis([0 1400 -350 350],'equal');
    xlabel('X-axis Base frame (mm)');
    ylabel('Y-axis Base frame (mm)');
    title('Workpiece shape (Workspace)');
end

```

Appendix E: M_adapt

Code which controls the change in the value of M for the Machining program:

```
function M = M_adapt (trigger, pos, cont1, cont2, Mno, Myes)

% pos = pos_irb ---> Cartesian
% trigger = gen_trigger
% pos cont1 = low (initial position of contact)
% pos cont2 = high (last position for machining task, on retract surface)
% Mno = M values for no-contact
% Myes = M values for machining task

persistent state

if isempty(state)
    state = 0;
end

cont_cont1 = norm(cont1-pos);
cont_cont2 = norm(cont2-pos);

if trigger ~= 0
    if cont_cont1 < 10 && state == 0 % First time at contact pos
        state = 1;
    elseif state == 1 && cont_cont2 < 10 % Second time at contact pos
        state = 2;
    end
end

else
    state = 0;
end

switch(state)
    case 0
        M = Mno;
    case 1
        M = Myes;
    case 2
        M = Mno;
    otherwise
        M = Mno;
end

end
```

Appendix F: Integral draining

Code for draining the integrals in the Machining program:

```
function [gen_trig,drain,repos] = drain (pos,trigger,omor,home)

% pos = pos_irb

% repos = responsible for the right orientation of the robot. Equal to 1 until
% the robot reaches omor for the first time to start the whole routine. No
% force control is applied while repos=1. The pendant rules the movement.

% gen_trig = changes to 1 when the robot is ready to start once placed at
% omor. This flag is used as trigger for the whole system

% !drain (low level active) = activates the drains for the integrals when
% the robot is finishing the RAPID program. Eliminates position error

persistent state
persistent state_omor
persistent state_home
persistent trig
persistent drai
persistent rep

if isempty(state)
    state = -1;
end

if isempty(state_omor)
    state_omor = 0;
end

if isempty(state_home)
    state_home = 0;
end

if isempty(trig)
    trig = 0;
end

if isempty(drai)
    drai = 1;
end

if isempty(rep)
    rep = 0;
end

cont_omor = norm(omor-pos);
cont_home = norm(home-pos);

if cont_omor < 1e-2 && state_omor == 0 % First time at omor
    state_omor = 1;
elseif state_home == 0 && cont_home < 1e-2 % First time at home
    state_home = 1;
elseif state_home == 1 && cont_home > 5e-2 % Indicates robot out of home because
    % program in execution
    state_home = 2;
elseif state_home == 2 && cont_omor < 1e-2 % Back to omor (last movement)
    state_omor = 2;
end

if trigger == 0 && state == -1 % Initial state fswitch = 0
    state = 0;
elseif trigger ~= 0 && state == 0 % fswitch != 0
    state = 1;
elseif state_omor == 2 && state == 1 % Back to omor (end of the program)
    state = 2;
```

```

end

switch(state)
case -1
    trig = 0;
    drai = 1;
    rep = 1;
case 0
    trig = 0;
    drai = 1;
    rep = 1;
case 1
    if state_omor == 1 % Once at omor give control to controller
        trig = 1;
        drai = 1;
        rep = 0;
    else
        trig = 0;
        drai = 1;
        rep = 1; % If not at omor give priority to pendant
    end
case 2 % By draining the integrals the robot is positioned where it should be
    trig = 0;
    drai = 0;
    rep = 0;
end

gen_trig = trig;
drain = drai;
repos = rep;

end

```