Hélio José Batista Ochoa

Compliant Control of the Kinova Robot for Surface Polishing

Master's Dissertation in MIEEC, supervised by

Professor Dr. Rui Pedro Duarte Cortesão and presented to the

Faculty of Science and Technology of the University of Coimbra

Coimbra 2018

· U C ·

Universidade de Coimbra

· U C ·

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Compliant Control of the Kinova Robot for Surface Polishing

Hélio José Batista Ochoa

Coimbra 2018

Compliant Control of the Kinova Robot for Surface Polishing

Supervisor:

Professor Dr. Rui Pedro Duarte Cortesão

Jury:

Professor Dr. Jorge Manuel Moreira de Campos Pereira Batista

Professor Dr. Rui Alexandre de Matos Araújo

Professor Dr. Rui Pedro Duarte Cortesão

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra 2018

Agradecimentos

Em primeiro lugar, agradeço aos meu pais, José e Maria, por fazerem de mim o que sou hoje e porque sem eles nada disto seria possível. Agradeço as minhas irmãs, Marta e Lúcia, pelo apoio, compreensão e por serem um exemplo a seguir.

Em segundo lugar, queria agradecer ao meu orientador, o Professor Doutor Rui Cortesão, por me ter concedido esta oportunidade, pelo seu acompanhamento, partilha de experiência e aconselhamento durante a realização deste trabalho.

De seguida um especial obrigado a Ana Rita Tavares, por ter estado sempre ao meu lado, pela paciência e por nunca me ter deixado baixar os braços.

Queria também agradecer ao Engenheiro Pedro Lino, por me ter ensinado as bases do ROS e pela sua amizade, e ao Engenheiro Miguel Mendes, por ter partilhado comigo o seu conhecimento e experiência à cerca do robô.

Por último, aos meus amigos, em especial aos LedZener, obrigado pela convivência e camaradagem durante o meu percurso académico.

Coimbra, Julho de 2018

Resumo

Este trabalho consiste em testar diferentes algoritmos de controlo no robô Kinova JACO^2 e propor uma solução para o polimento automatizado. Maioritariamente na indústria o processo de polimento é ainda realizado manualmente, o que requer muito tempo do ser humano a desempenhar a mesma tarefa, o que o limita em termos de quantidade produzida.

Inicialmente são testados algoritmos de controlo simplificados por forma a testar e avaliar o desempenho do robô. Os algoritmos testados correspondem a arquiteturas de controlo avançadas, como o controlo de binário computorizado, quer no espaço das juntas quer no espaço de tarefa. Posteriormente é testado o controlador de impedância, que permite relacionar uma determinada posição, velocidade e aceleração com a força necessária. Afim de validar os controladores anteriormente mencionados estes são testados em ambiente de simulação e em ambiente real. Finalmente o controlador de impedância é testado numa tarefa real de polimento.

Verificou-se que o controlador de impedância pode ser usado para desempenhar tarefas de polimento. Contudo o desempenho do robô não é perfeito, devido à fricção dos atuadores e a outros fatores externos, o que pode ser mitigado com um estudo minucioso dos vários parâmetros considerados no controlador e dos parâmetros internos do manipulador.

Abstract

This work consists of developing different control algorithms in Kinova JACO^2 robot and propose a solution for automated surface polishing. Nowadays, surface polishing is performed manually, which requires a long time for a human performing the same task, being limited in terms of production capacity.

Initially, several simplified control algorithms are tested so as to evaluate the robot performance. The tested algorithms correspond to computed torque controller in the joint and task space. Then, the impedance control is tested since it allows to regulate the relationship between the required force and position, velocity and acceleration. In order to validate the control architectures they are tested in simulated and real environment. Finally, the impedance controller is tested in a real surface polishing task.

In conclusion, the impedance controller can be used in polishing tasks. However, the robot performance needs improvements due to actuators friction and other factors. This problem can be mitigated by an in-depth study of the controller parameters as well as the internal parameters of the manipulator.

"Look up, get up and don’t ever give up!"

— Michael Irvin,

Contents

Agradecimentos **iii**

Resumo **v**

Abstract **vii**

List of Acronyms **xv**

List of Figures **xvii**

List of Tables **xxi**

1 Introduction **1**

1.1 Background 1

1.2 Objectives 2

1.3 Contributions 2

1.4 Organization 3

2 The Kinova JACO^2 Robotic Arm **5**

2.1 System Overview 5

2.2 Communication Modes 7

2.3 Kinova-ROS 8

3 JACO^2 Kinematics and Dynamics **13**

3.1 Direct Kinematics 13

3.2 Differential Kinematics 18

3.3 Dynamics 20

4 Control Architectures **23**

4.1 ROS Control 23

4.2 Computed torque control in the joint space 25
4.3 Computed torque control in the task space 27
4.4 Joint Space Control with Task Posture Reference 30
4.5 Impedance Control 35

5 Experimental Results 43

5.1 Gazebo Simulator Results 43
5.1.1 Computed torque control in the joint space (simulator) 43
5.1.2 Computed torque control in the task space (simulator) 45
5.1.3 Joint space control with task posture reference (simulator) 47
5.1.4 Impedance control in the task space without force sensing (simulator) 49
5.1.5 Impedance control in the task space with force sensing (simulator) . . 50
5.1.6 Impedance control in the task space with force sensing for redundant robots (simulator) 52
5.2 Real JACO^2 Results 54
5.2.1 Computed torque control in the joint space 55
5.2.2 Computed torque control in the task space 56
5.2.3 Impedance control in the task space without force sensing 57
5.2.4 Impedance control in the task space with force sensing 58
5.2.5 Impedance control in the task space with force sensing for redundant robots 59

6 Surface Polishing 63

6.1 Polishing task in Gazebo simulator 63
6.2 Polishing task in the real JACO^2 65

7 Conclusion 71

7.1 Future Work 71

8 Bibliography 73

A JACO^2 Product Specification 77

B Actuators Product Specification 81

C Kinova-ROS: Installation 85

D Kinova-ROS: How to use the stack **87**

D.1 Gazebo Simulator 87

D.2 Real JACO^2 88

E Robot Operating System **89**

F Polishing Task Example **97**

List of Acronyms

API	Application Programming Interface
COM	Center of Mass
DH	Denavit-Hartenberg
DOF	Degrees of Freedom
DSP	Digital Processing Unit
EL	Euler-Lagrange
KDL	Kinematics and Dynamics Library
NE	Newton-Euler
OS	Operating System
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
ROS	Robot Operating System
SDK	Software Development Kit
URDF	Unified Robot Description Format

List of Figures

2.1 The Kinova JACO^2 Research Edition system. 5
2.2 Kinova's joystick possible commands [7]. 6
2.3 General Architecture from low level up of *kinova_driver*. 8
2.4 The joint position control on the left (2.4a) and the cartesian position control on the right (2.4b). 9
2.5 Kinova JACO^2 arm in Gazebo simulator. 10
2.6 Data flow of *ros_control* and Gazebo. 11
2.7 Kinova JACO^2 arm in RViz MoveIt Plugin. 12

3.1 Description of the position and orientation of the end-effector frame [25]. . . 14
3.2 JACO^2 6DOF curved wrist components [7]. 15
3.3 Basic geometric parameters of the JACO^2 6DOF curved wrist configuration [7]. 16
3.4 Auxiliary parameters that are useful for describing the geometry for direct kinematics of the JACO^2 6DOF curved wrist configuration [7]. 16
3.5 JACO^2 6DOF curved wrist coordinate frames for the DH algorithm [7]. . . . 18

4.1 Diagram of the relationship between controller interfaces, controller managers, transmissions and hardware interfaces in ros_control [18]. 24
4.2 PID computed torque control in the joint space. 26
4.3 Position/Orientation PD control in the task space. 28
4.4 PD joint space torque control with task space posture reference. 33
4.5 The two pictures above shows the 3 different situations that occur between robot and environment. [3] 35
4.6 Impedance control scheme in the task space without force sensing. [23] . . . 36
4.7 Impedance control architecture in the task space without force sensing and without inertia shaping. 37
4.8 Impedance control scheme in the task space with force sensing. 40

4.9 Impedance control scheme in the task space with force sensing for redundant robots and with quaternion orientation. 42

5.1 PID computed torque control in the joint space (Gazebo Simulator). 45
5.2 Position/Orientation PD computer torque control in the task space (Gazebo Simulator). 47
5.3 Position PD joint space torque control with task space posture reference (Gazebo Simulator). 48
5.4 Orientation PD joint space torque control with task space posture reference (Gazebo Simulator). 48
5.5 Position/Orientation impedance control without force sensing (Gazebo Simulator). 50
5.6 Position/Orientation impedance control with force sensing (Gazebo Simulator). 51
5.7 Position impedance control with force sensing for redundant robots (Gazebo Simulator). 52
5.8 Quaternion orientation impedance control with force sensing for redundant robots(Gazebo Simulator). 53
5.9 PID computed torque control in the joint space, with respect to the first solution. 56
5.10 Position/Orientation PD computer torque control in the task space. 57
5.11 Position/Orientation impedance control without force sensing. 58
5.12 Position/Orientation impedance control with force sensing. 59
5.13 Position impedance control with force sensing for redundant robots. 60
5.14 Quaternion orientation impedance control with force sensing for redundant robots. 60
5.15 Position impedance control with force sensing for redundant robots using a low pass filter with a cut-off frequency of 100 Hz. 61
5.16 Quaternion orientation impedance control with force sensing for redundant robots using a low pass filter with a cut-off frequency of 100 Hz. 62

6.1 Polishing task demonstration in simulated environment. 63
6.2 End-effector position during the polishing task (Gazebo Simulator). 64
6.3 The circular defined trajectory and end-effector force measured (Gazebo Simulator). 64
6.4 End-effector orientation during the polishing task (Gazebo Simulator). . . . 65
6.5 Polishing tool with a sandpaper embedded in the robot end-effector. 66

6.6 State-machine for the polishing task. 67
6.7 End-effector position during the polishing task. 68
6.8 The circular defined trajectory and end-effector force measured. 68
6.9 End-effector orientation during the polishing task. 69

E.1 ROS File-System level. [10] 91
E.2 Structure of the ROS Graph layer. [10] 92
E.3 ROS basic concepts. [10] 94

List of Tables

3.1 JACO^2 6DOF curved wrist basic geometric parameters. 15

3.2 Auxiliary parameters of the JACO^2 6DOF curved wrist. 17

3.3 JACO^2 6DOF curved wrist DH parameters. 17

3.4 Transformation from DH algorithm to robotic arm physical angles. 17

5.1 The control gains were done by the following expressions: $K_{p_j} = w_{n_j}^2$ and $K_{d_j} = 2w_{n_j}$. 45

5.2 Position and Orientation Gains. 46

5.3 Joint control architecture gains. 49

5.4 Position and Orientation Gains. 50

5.5 Impedance task space control gains, where the variable I in 5.5a and 5.5b represents the identity matrix. 51

5.6 Impedance task space control gains, where the variable I in 5.6a and 5.6b represents the identity matrix. 53

5.7 Internal parameters that need to be changed. 54

5.8 Control Gains. 56

5.9 Position and Orientation Gains. 57

5.10 Position and Orientation Gains. 58

5.11 Impedance task space control gains, where the variable I in 5.11a and 5.11b represents the identity matrix. 59

5.12 Impedance task space control gains, where the variable I in 5.12a and 5.12b represents the identity matrix. 61

6.1 Impedance task space control gains in Gazebo Simulator. 66

6.2 Impedance task space control gains obtained in the real JACO^2. 69

1 Introduction

1.1 Background

Nowadays, polishing process is considered as one of the essential final machining processes in various precision industries including die and mould manufacturing, air-foils, sculpture, camshaft and crankshaft. It is used to remove surface and subsurface damages and improve its roughness. However, the polishing processes of these parts are primarily conducted manually which is not only time-consuming and exposes labourers to high noise levels and metal dust environments, but also it is difficult to maintain a stable polishing operation for long time [28]. For example, to manufacture a mould the time spent on the polishing process accounts for 37-50% of the total manufacturing time [13]. In addition, labours conducting manual polishing for long time may get "vibration white finger" or other musculoskeletal diseases. Furthermore, to obtain quantitative and qualitative processing some companies may have difficulty in recruiting and training sufficient numbers of highly skilled manual workers [13].

In order to address the mentioned limitations, some industries are strongly motivated to seek and implement alternative solutions in their manufacturing processes such as Computer Numerically Controlled (CNC) machines and industrial robots. CNC machines have remarkable positioning accuracy and splendid ability to simultaneously adjust the trajectory, posture, and force during polishing tasks. However, the limited available working space of these machines usually leads to process one part in multiple stages and restricts the size of work-piece. Apart from the working space, polishing processes do not require a high positioning accuracy otherwise human operators could not be able to perform them nevertheless it requires accurate force control.

Recently, robotic machining and finishing have attracted many researchers due to its advantages compared with CNC machines, such as low cost, higher flexibility and greater capability of integration with actuators, sensors and different end-effectors.

In order to achieve a human-like polishing, the robot tool should have some flexibility which includes passive compliance or active compliance control. In passive compliance control, by using a passive mechanical element such as springs, the contact force between the work-piece and the polishing tool is converted to a natural obedience deformation. On the other hand, the active compliance control, also known as force control, employs a closed control loop to regulate the contact force between the polishing head and the work-piece [13].

The impedance control is another possible solution. This controller does not simply regulate force or position, instead it regulates the relationship between force on the one hand and position, velocity and acceleration on the other hand. It requires a position, velocity or acceleration as input and has a resulting force as output. The controller imposes a mass-spring-damper behaviour on the mechanism by maintaining a dynamic relationship between force and position, velocity and acceleration [4].

1.2 Objectives

First of all, this work consists in finding and developing strategies to help operators in polishing processes. For this reason, the Kinova JACO^2 6DOF robot arm is tested to achieve a human-like polishing.

In order that, it is required to study the capabilities and limitations of the robot purposed. After that, it is necessary to understand the different control architectures that could be used in order to accomplish the principal objective. Then, it is essential to implement the suitable controllers, being the impedance control one of the possibilities.

Finally, it is needed a demonstration or demo of the developed work.

1.3 Contributions

The work presented gives the possibility to test different computed torque control architectures in Gazebo simulator as well as in real JACO^2 robot.

In addition, this work offers a low-cost solution in polishing processes and portability because of the reduced weight of JACO^2.

Lastly, open doors to an automatic polishing system and also to human-machine interaction.

1.4 Organization

- **Chapter 1:** Covers the background, objectives, contributions and document structure;
- **Chapter 2:** Deals with the robot JACO^2 system overview, the possible communication modes with the robot and presents the Kinova-ROS stack;
- **Chapter 3:** Explains the theoretical background of the robot kinematics, differential kinematics and dynamics;
- **Chapter 4:** Presents the control architectures to be implemented in the Gazebo simulator and in the real JACO^2 robot arm;
- **Chapter 5:** Analyses and comments the experimental results in the Gazebo simulator and in the real robot;
- **Chapter 6:** Is devoted to the polishing task in the simulator and in the real JACO^2 arm;
- **Chapter 7:** Sets out the work final conclusions and suggestions for future work.

2 The Kinova JACO^2 Robotic Arm

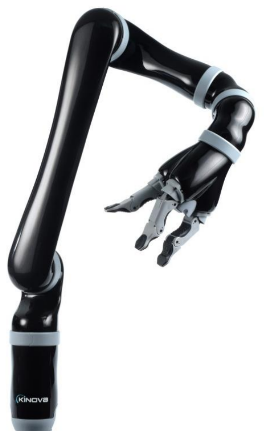
2.1 System Overview

The robotic arm JACO^2 was developed by Kinova Robotics [1], a Canadian company focused on assistive/rehabilitation devices. At first, Kinova tried to help people with reduced mobility or upper limb impairments, as a result they created a commercial version in 2010, JACO Rehab Edition. Later, in 2012 they developed a scientific version called JACO^2 Research Edition.

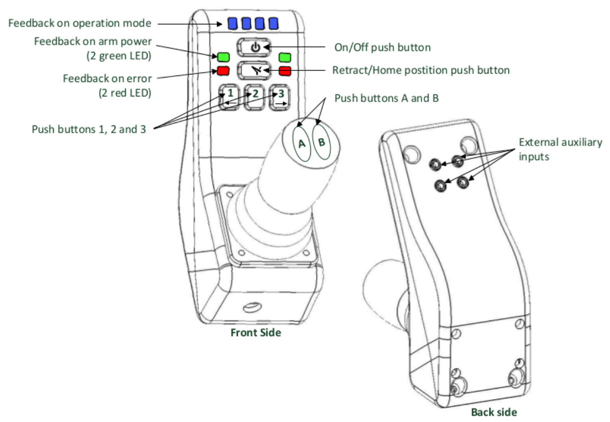
The version used in this work is JACO^2 Research Edition, this version is a 6 degree of freedom (DOF) with a removable 3 fingered end-effector (figure 2.1).

The robot is composed of six interconnected carbon fibber links, jointed together by six aluminium brushless direct current actuators. It is important to refer that the actuators do not possess mechanical limitation allowing unlimited rotation around their axis. Because of the nature of the carbon fibre, the main structure has a very lightweight composition.

The end-effector consists of three fingers which can be individually controlled. They are made of plastic allowing them to precisely adjust to different sizes and types of objects.



(a) The Kinova JACO^2 6DOF curved wrist [7].


Feedback on operation mode
Feedback on arm power (2 green LED)
Feedback on error (2 red LED)
Push buttons 1, 2 and 3
On/Off push button
Retract/Home postition push button
Push buttons A and B
1
2
3
A
B
External auxiliary inputs
Front Side
Back side

(b) Kinova's joystick part identification [7].

Figure 2.1: The Kinova JACO^2 Research Edition system.

Appendix A details a description of the robot specifications and appendix B describes the actuators specification, both taken from Kinova user guide [7].

The robot arm could be controlled through a three-axis joystick with five independent push-buttons and four external auxiliary inputs placed at the back side of the controller (fig. 2.1b). The front side of the controller (fig. 2.1b) has the power button, HOME button and other five buttons for the switching of operation modes. The blue light on the front side displays the current operation mode while the green lights show that the robot is ready and powered to be used.

The HOME button moves the arm to a preprogrammed pose. The rest of the buttons are assigned to the shift between three-axis and two-axis mode. Fundamentally, the robot can be easily changed into translation mode, wrist mode, "drinking" mode and finger mode. The control using the joystick is considered cartesian as the user can only change the position or orientation of the gripper.

As mentioned previously, the translation mode allows the controlling of the hand in the three axis of the cartesian coordinate system. In the wrist mode, the arm is controlled around a reference point set the middle of the end-effector and the arm remains stationary around that point changing its orientation. The "drinking" mode, allows the wrist to produce a rotation around another point in the space through an offset in height and length from the reference point. The name "drinking" mode is because it is used to help during the grasping of bottles and cups. Ultimately, the finger mode lets the user open and close the hand.

The joystick possible commands or movements are presented in (fig. 2.2). The joystick provided by Kinova Robotics is intuitive at start, but the number of different modes that can be chosen could be confusing for unfamiliar users, it requires practice.

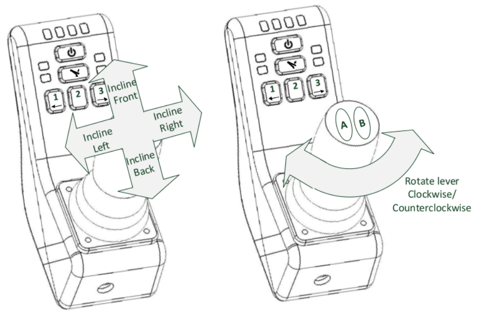


Figure 2.2: Kinova's joystick possible commands [7].

The JACO^2 robot arm could be considered as a portable device due to its very light frame. Additionally, the power supply is considerable small which effectively puts into evidence the easiness to carrying the robot to everywhere. Furthermore, the arm has sensors to ascertain temperature, voltage, current and torque parameters. With respect to hardware connectivity, apart from the necessary connections for an approved power supply and joystick, the robot comes with an ethernet port for wireless connections and a universal serial bus (USB) port to connect to a computer or laptop in order to send commands through the available software development kit (SDK) [5] [12].

2.2 Communication Modes

One of the communication modes is offered by Kinova, a complete graphical user interface to control the robot arm, both in trajectory and torque control. The SDK grants to the user complete control over the functionalities of the manipulator without the need to use the joystick.

Another one is the Kinova Application Programming Interface (API). The API is available with a lot of tools, which gives the possibility to create specific software programs. The programming language from where the API is built is C++ and the internal communication protocol used to control the actuators is the RS485. It is possible to interact directly with the actuators by using an appropriate RS485 interface or USB-RS485 communication modules.

However, there is another way of establishing a communication link with the robot arm, the Kinova Robot Operating System (Kinova-ROS) [9]. The Kinova-ROS stack provides a ROS interface for the Kinova Robotics JACO, JACO2 and MICO robotic manipulator arms. The stack is developed above the Kinova C++ API functions, which communicate with the DSP inside robot base. It is important to refer that the recommended configuration is ROS Indigo with 64 bit Ubuntu 14.04. Concerning to the programming language, the Kinova-ROS has two alternatives, the C++ and python. The programming language used in this work is python because is simple and intuitive.

The control system frequency depends on what communication method is used, if it is high level, like via USB, the rate is between 100 to 500 Hz, but the refresh rate of the controller of the DSP is 100 Hz. On the other hand, if the communication is made directly with the actuators, the low level approach, then the communication rate is 500 Hz. Nonetheless, this work was made following a high level approach, which means that all the control architectures follow the refresh rate represented by the DSP controller, that is 100 Hz.

The communication mode chosen in this work is the Kinova-ROS, because ROS is a flexible framework for writing robot software and simplifies the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. Appendix E briefly describes the functionalities, history and objectives of the ROS.

2.3 Kinova-ROS

Description

As it is said before, the kinova-ros stack [9] provides a ROS interface for the Kinova Robotics JACO, JACO^2, and MICO robotic manipulator arms. The robot used in this work is JACO^2. The stack is composed for the following file system:

- *kinova_bringup:* This package provides a launch file to start *kinova_driver* and apply some configurations.

- *kinova_driver:* This package is composed of the most essential files to run kinova-ros stack. Under the include folder, Kinova C++ API headers are defined, and ROS package header files are in *kinova_driver* folder. The *kinova_api* source file is a wrap of Kinova C++ API, *kinova_comm* builds up the fundamental functions. Some advanced accesses regarding to force/torque control are only provided in *kinova_api.* Most parameters and topics are created in *kinova_arm.* Figure 2.3 shows a general architecture from low level up.

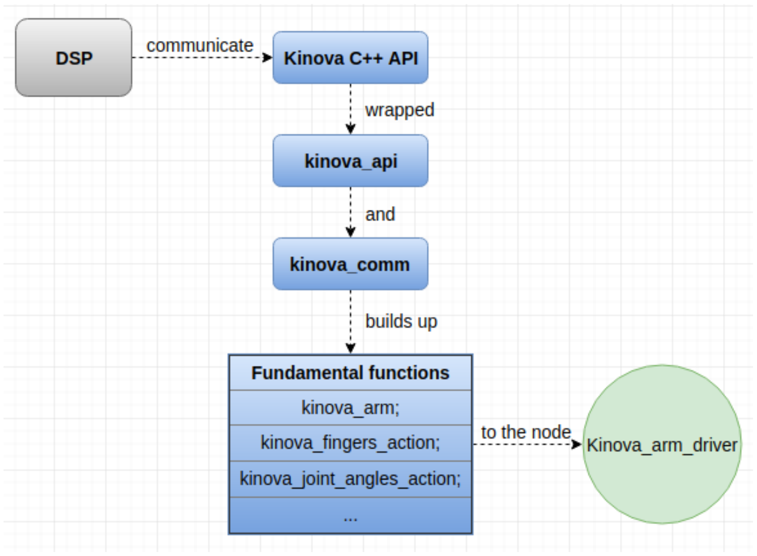
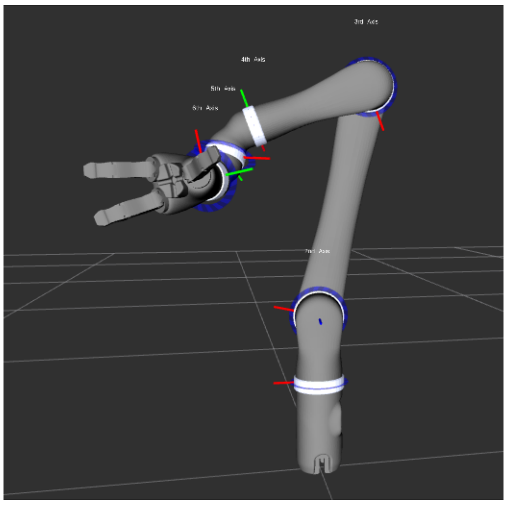


Figure 2.3: General Architecture from low level up of *kinova_driver.*

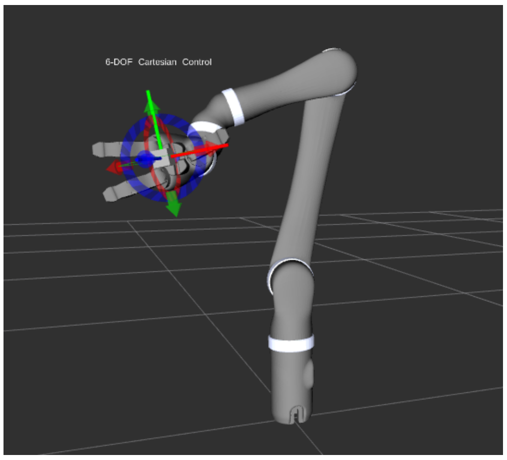
- *kinova_demo:* This package provides some python scripts for *actionlibs* in joint space and cartesian space.

- *kinova_msgs:* This package contains all the *messages*, *servers* and *actionlib* format.
- *kinova_description:* This package contains a Unified Robot Description Format (URDF), that is a standard ROS XML format for describing robot models and meshes.
- *kinova_docs:* Contains a folder, *kinova_comm*, that is a reference for html files generated by doxygen.

The stack offers different ways to control the robot in ROS. The stack came with a **Joint position control** and a **Cartesian position control**. To access this controllers you need follow some steps that are presented in appendix D. Another way to control the robot is to use the interactive markers in Rviz [21]. Figure 2.4 shows the two controllers mentioned before.



(a) A blue ring at each joint, that allows you to move the robot by dragging the rings.



(b) A cubic with 3 axis (translation) and 3 rings (rotation) at the end-effector, that give you the possibility to move the robot by dragging the axis or rings.

Figure 2.4: The joint position control on the left (2.4a) and the cartesian position control on the right (2.4b).

There is a **Finger position control** that allows you to open/close each finger individually and a **Velocity Control** for joint space and cartesian space.

The users have to their disposal some ROS service commands, like bring the robot to pre-defined home position, enable and disable the ROS motion, switch to **Cartesian Admittance mode**, switch to **torque control** from **position control**, and a service that permits to move robot in **Null Space**. The cartesian admittance mode lets the user control the robot by manually (by hand).

Furthermore, the torque control allows the user to publish torque/force commands just like joint/cartesian velocity.

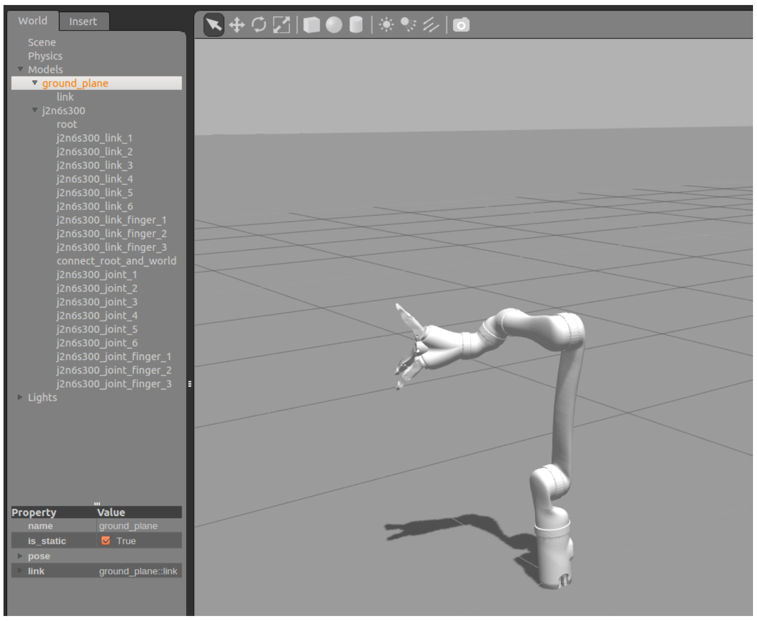
It is possible that the torque sensors develop offsets in reporting absolute torque over time, they need to be re-calibrated. Due to this, it is necessary to follow a very simple calibration process.

It is important to refer, that to make kinova-ros part of your workspace you need to follow some steps which are available in appendix C.

In appendix D a description showing how to use the stack is presented and more information is available in Kinova-ROS stack [9].

Gazebo Simulator

Moreover, the kinova-ros stack come with a Gazebo package [6], *kinova_gazebo*, this package make available a robot simulation, and that is an essential tool in every roboticist's toolbox, see figure 2.5. A simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train system using realistic scenarios [15].



Property	Value
name	ground_plane
is_static	True
pose	
link	ground_plane::link

Figure 2.5: Kinova JACO^2 arm in Gazebo simulator.

To control the robot model in Gazebo the stack use *ros_control* [19]. As a result, three different types of controllers are available - effort, position and velocity. The configuration files for controlling the joints using *ros_control* are available in *kinova_control* package. Trajectory controller has also been added to provide interface for MoveIt.

An overview of the relationship between simulation, hardware, controllers and transmissions is shown in figure 2.6.

As a consequence, *ros_control* parameters have been added to Gazebo. Firstly, inertial parameters are added to all mesh models to support gazebo, each link has the inertia model

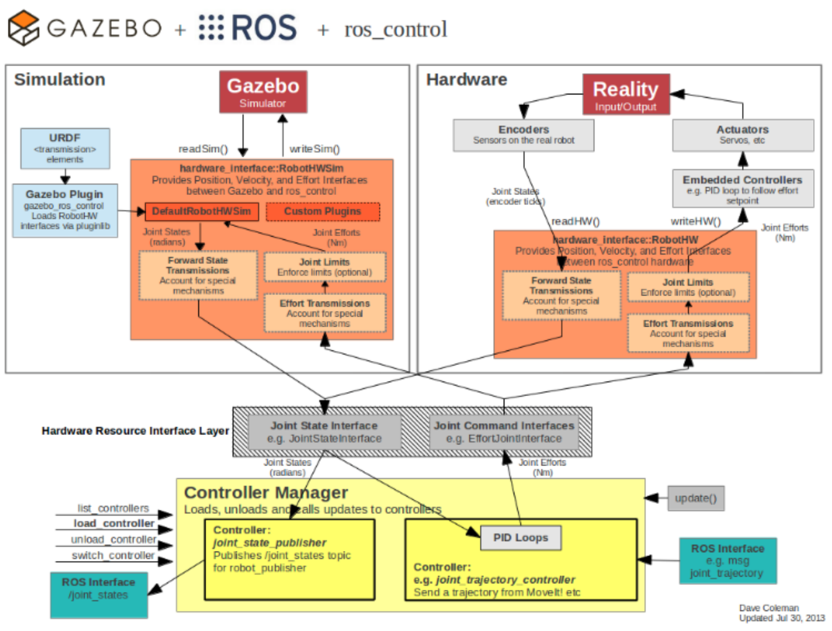


Figure 2.6: Data flow of *ros_control* and Gazebo.

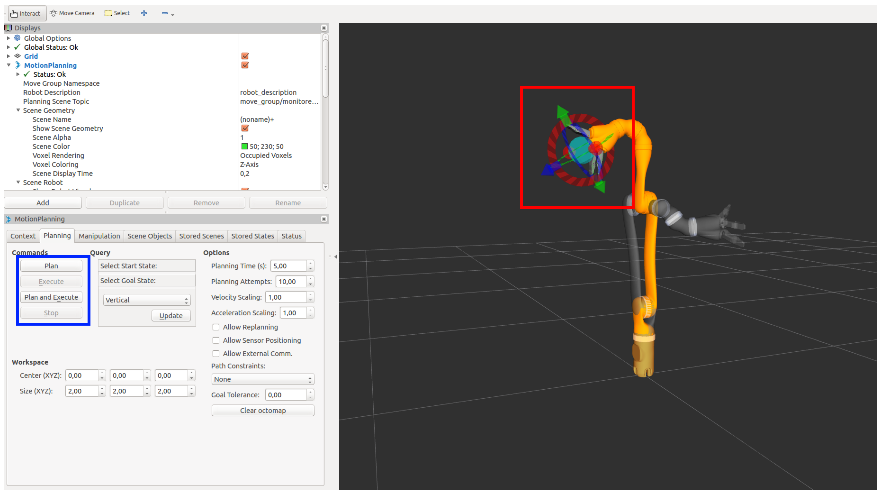
for the link and half of the actuator on one side and half on the other side. Center of mass (COM) position and mass of the links are accurate to result in correct torque readings. Secondly, the joint dynamics parameters, like damping, friction, stiffness, do not accurately represent the hardware. Thirdly, the joints are configured to have an effort joint interface [18]. Lastly, the inertia matrix is an approximation, assuming uniform cylinders for links.

MoveIt

As Gazebo the kinova-ros stack provides a MoveIt package [8], *kinova_moveit*. This package allows choosing to launch MoveIt with a virtual robot, useful for visualization and testing, or to launch MoveIt with the *kinova_driver* node which controls the real robot [20].

Figure 2.7 shows the RViz window with the MoveIt plugin. The user can move the robot's end-effector using the interactive markers (marked in the figure by a red rectangle). While you move the end-effector MoveIt runs inverse kinematics to update the joint positions while you drag the marker.

The planing tab (marked in the figure 2.7 by a blue rectangle), gives the ability to plan and execute the trajectory. Clicking on *plan* button, MoveIt visualizes its planned path to move from the start state to the goal state, and if the visualized plan is acceptable it is possible to command the robot to execute the plan by clicking *execute* button. Connected to the real robot, the robot moves according to the visualized plan, or running the virtual robot the joints of the model moves to the goal state. In addition, the moveIt RViz plugin



Global Options	
Global Status: Ok	
Grid	✓
MotionPlanning	✓
Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitore…
Scene Geometry	
Scene Name	(noname)+
Show Scene Geometry	✓
Scene Alpha	1
Scene Color	🟩 50; 230; 50
Voxel Rendering	Occupied Voxels
Voxel Coloring	Z-Axis
Scene Display Time	0,2
Scene Robot	

Select Start State:
Select Goal State:
Vertical

Planning Time (s): 5,00
Planning Attempts: 10,00
Velocity Scaling: 1,00
Acceleration Scaling: 1,00
☐ Allow Replanning
☐ Allow Sensor Positioning
☐ Allow External Comm.
Path Constraints: None
Goal Tolerance: 0,00

Workspace
Center (XYZ): 0,00 0,00 0,00
Size (XYZ): 2,00 2,00 2,00

Figure 2.7: Kinova JACO^2 arm in RViz MoveIt Plugin.

can also be used to add obstacles and edit planning parameters.

3 JACO^2 Kinematics and Dynamics

This chapter focus on the derivation of the direct kinematics, differential kinematics and dynamics of the JACO^2 6DOF curved wrist arm.

In order to compute the kinematics and dynamics of the JACO^2 it is used the Kinematics and Dynamics Library (KDL), distributed by the Orocos Project [27]. The KDL gives the possibility to use solvers to compute anything from forward position kinematics to inverse dynamics and includes support to construct a KDL chain from a XML Robot Description Format (URDF) file. It is important to emphasise, that the kinova-ros stack provides a package that contains the URDF model of JACO^2. This include the physical geometry, the kinematic and dynamic properties of the robot arm. For more information on URDF's consult the ROS URDF wiki page [22].

Due to the fact that kinematics and dynamics are calculated using the KDL, a theoretical background is granted in the next sections.

3.1 Direct Kinematics

A manipulator consists of a series of rigid bodies (links) connected by means of kinematic pairs or joints. Joints can be essentially of two types: revolute and prismatic. The whole structure forms a kinematic chain. One end of the chain is constrained to a base and an end-effector (gripper, tool) is connected to the other end.

The mechanical structure of a manipulator is characterized by a number of degrees of freedom (DOFs) which uniquely determine its *posture* [1]. Each DOF is typically associated with a joint articulation and constitute a joint variable. The aim of **direct kinematics** is to compute the pose of the end-effector as function of the joint variables.

The pose of a body with respect to a reference frame is described by position vector of the origin and the unit vectors of a frame attached to the body. As result, with respect

[1]The term *posture* of a kinematic chain denotes the pose of all rigid bodies composing the chain.

to a reference frame $O_b - x_b y_b z_b$, figure 3.1, the direct kinematics function is expressed by the homogeneous transformation matrix

$$T_e^b(q) = \begin{bmatrix} n_e^b(q) & s_e^b(q) & a_e^b(q) & p_e^b(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_e^b(q) & p_e^b(q) \\ O^T & 1 \end{bmatrix} \tag{3.1}$$

where q is the $(n \times 1)$ vector of joint variables, n_e, s_e, a_e are the unit vectors of a frame attached to the end-effector, and p_e is the position vector of the origin of such a frame with respect to the origin of the base frame $O_b - x_b y_b z_b$. In addition, if you combine the three unit vectors in figure 3.1, n_e, s_e, a_e, they result in the (3×3) matrix, which is termed rotation matrix, R_e.

The frame $O_b - x_b y_b z_b$ is denoted base frame. The frame attached to the end-effector is termed end-effector frame and is conveniently chosen according to the particular task geometry. For example, if the end-effector is a gripper, the origin of the end-effector frame is located at the centre of the gripper, the unit vector a_e is chosen in the approach direction to the object, the unit vector s_e is chosen normal to a_e in the sliding plane of the jaws, and the unit vector n_e is chosen normal to the other two, so the frame is right-handed.

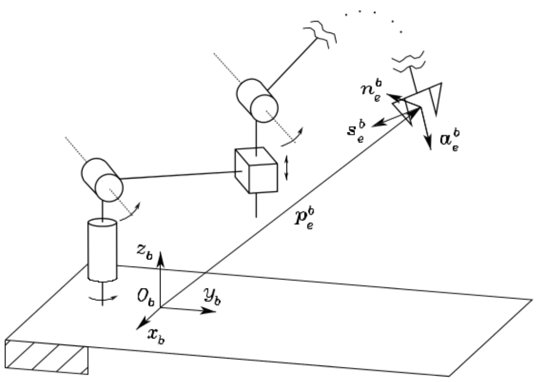


Figure 3.1: Description of the position and orientation of the end-effector frame [25].

Analogously, the JACO^2 6DOF curved wrist robotic arm, figure 3.2, can be represented by a set of reference frames between each of its links, in a similar fashion as displayed in figure 3.1. Then the transformation between the end-effector and the base frame is given by

$$T_e^b(q) = T_1^b(q_1) T_2^1(q_2) ... T_n^{n-1}(q_n) \tag{3.2}$$

where $n = 6$ for the robotic manipulator in study.

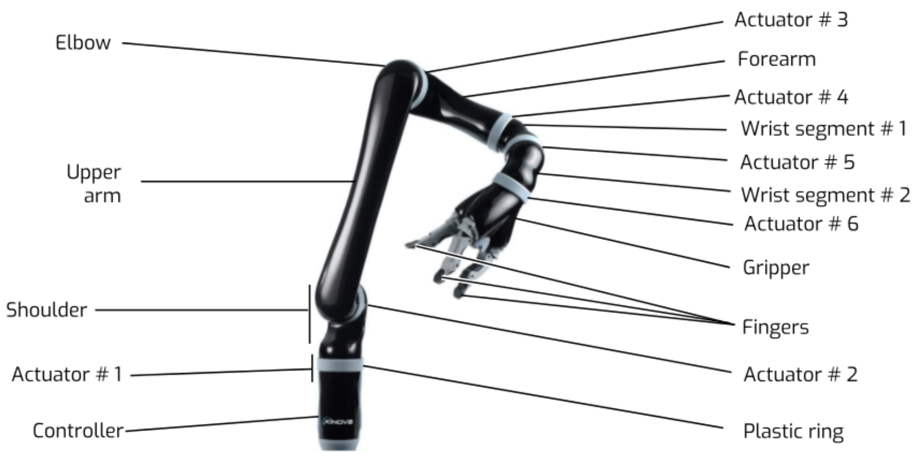


Figure 3.2: JACO^2 6DOF curved wrist components [7].

Denavit-Hartenberg Convention

In order to compute the direct kinematics equation for an open-chain manipulator like JACO^2, according to the recursive expression 3.2, a systematic, general method is to be derived to define the relative position and orientation of two consecutive links; the problem is determining two frames attached to the two links and computing the coordinate transformations between them. The Denavit-Hartenberg convention (DH) gives a possible solution to ascertain that. One good description of the DH method is presented in this book [25].

The Kinova Ultra lightweight robotic arm User Guide [7] describes the classic DH parameters for the 6DOF curved wrist manipulator as well as the basic geometric parameters. Next, table 3.1 provides the basic geometric parameters, pictured in figure 3.3.

Table 3.1: JACO^2 6DOF curved wrist basic geometric parameters.

Parameter	Description	Length (m)
D1	Base to shoulder	0.2755
D2	Upper arm length (shoulder to elbow)	0.4100
D3	Forearm length (elbow to wrist)	0.2073
D4	First wrist length (center of actuator 4 to center of actuator 5)	0.0741
D5	Second wrist length (center of actuator 5 to center of actuator 6)	0.0741
D6	Wrist to center of the hand	0.1600
e2	Joint 3-4 lateral offset	0.0098

To simplify the analysis of figure 3.3 it is useful to break down each of the curved wrist segments into two component straight-line sub-segments of equal length, with the second

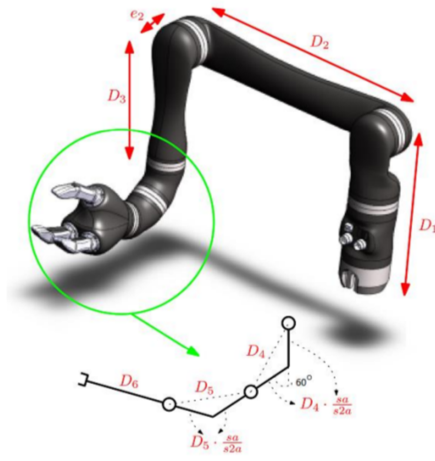


Figure 3.3: Basic geometric parameters of the JACO^2 6DOF curved wrist configuration [7].

sub-segment angled 60° from the first. Figure 3.4 shows the procedure in detail.

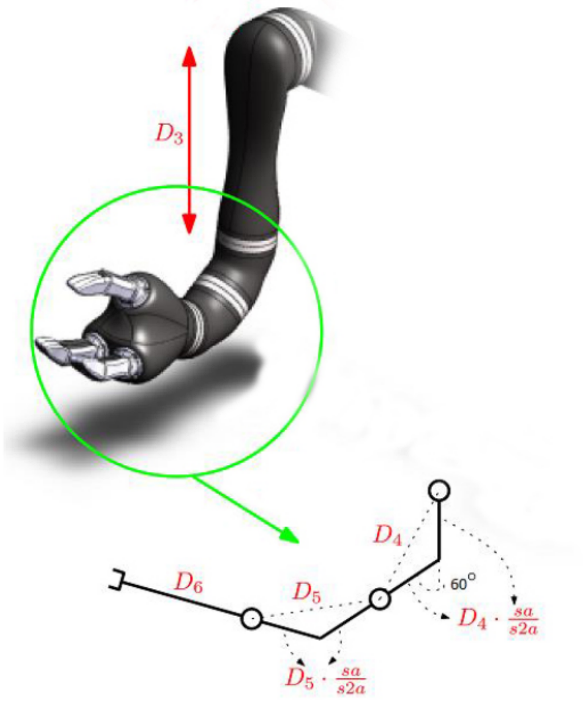


Figure 3.4: Auxiliary parameters that are useful for describing the geometry for direct kinematics of the JACO^2 6DOF curved wrist configuration [7].

Table 3.2: Auxiliary parameters of the JACO² 6DOF curved wrist.

Parameter	Description	Value
aa	Half of the angle of curvature of each wrist segment (60°), measured in radians.	$\frac{30.0\pi}{180.0}$
sa	Sine of half the angle of curvature of wrist segment.	$sin(aa)$
s2a	Sine of angle of curvature of wrist segment.	$sin(2aa)$
d4b	Length of straight-line segment from elbow to end of first sub-segment of first wrist segment.	$D3 + (\frac{sa}{s2a})D4$
d5b	Length of straight-line segment consisting of second sub-segment of first wrist segment and first sub-segment of second wrist segment.	$(\frac{sa}{s2a})D4 + (\frac{sa}{s2a})D5$
d6b	Length of straight-line segment consisting of second sub-segment of second wrist segment and distance from wrist to the center of the hand.	$(\frac{sa}{s2a})D5 + D6$

Finally, the classic DH parameters are shown in table 3.3, presented below.

Table 3.3: JACO² 6DOF curved wrist DH parameters.

T_i^{i-1}	α_{i-1}	a_{i-1}	d_i	θ_i
T_1^0	$\frac{\pi}{2}$	0	D1	q_1
T_2^1	π	D2	0	q_2
T_3^2	$\frac{\pi}{2}$	0	-e2	q_3
T_4^3	$2aa$	0	-d4b	q_4
T_5^4	$2aa$	0	-d5b	q_5
T_6^5	π	0	-d6b	q_6

Table 3.4: Transformation from DH algorithm to robotic arm physical angles.

q_1(physical)	$-q_1(robot)$
q_2(physical)	$q_2(robot) - 90°$
q_3(physical)	$q_3(robot) + 90°$
q_4(physical)	$q_4(robot)$
q_5(physical)	$q_5(robot) - 180°$
q_6(physical)	$q_6(robot) + 90°$

Basically, the JACO² physical angles need to be converted into the angles of the DH algorithm, which is given by the relation shown in table 3.4. Therefore, the DH parameters are obtained after establishing the coordinate frames for each link exhibited in figure 3.5.

The transformation matrices associated to each link could be obtained following the DH convention. That transformation matrix from frame *i* to frame *i-1* is given by

$$T_i^{i-1} = \begin{bmatrix} cos(\theta_i) & -cos(\alpha_i)sin(\alpha_i) & sin(\alpha_i)sin(\theta_i) & a_icos(\theta_i) \\ sin(\theta_i) & cos(\alpha_i)cos(\theta_i) & -sin(\alpha_i)cos(\theta_i) & a_isin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

As soon as the DH parameters are known, the transformation matrices associated to each link could be calculated by applying the equation 3.3, then the forward kinematics can be calculated through equation 3.2.

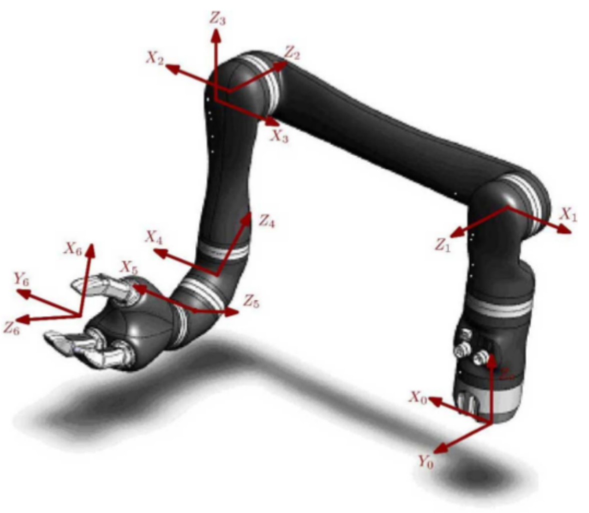


Figure 3.5: JACO^2 6DOF curved wrist coordinate frames for the DH algorithm [7].

3.2 Differential Kinematics

The differential kinematics gives the relationship between the joint velocities and the corresponding end-effector linear and angular velocity. This mapping is described by a matrix, termed geometric Jacobian, which depends on the manipulator configuration. The Jacobian constitutes one of the most important tools for manipulator characterization, it is useful for finding singularities, analysing redundancy, determining inverse kinematics, and can be useful for computing forces applied to the end-effector and resulting torques at the joints.

Firstly, it is desired to express the end-effector linear velocity $\dot{p}_e$ and angular velocity w_e as a function of the joint velocities $\dot{q}$. The relation described above is given by

$$\begin{cases} \dot{p}_e = J_P(q)\dot{q} \\ w_e = J_O(q)\dot{q} \end{cases} \tag{3.4}$$

where J_P is the $(3 \times n)$ matrix relating the contribution of the joint velocities $\dot{q}$ to the end-effector linear velocity $\dot{p}_e$, while J_O is the $(3 \times n)$ matrix relating the contribution of the joint velocities $\dot{q}$ to the end-effector angular velocity w_e. In compact form, 3.4, can be written as

$$v_e = \begin{bmatrix} \dot{p}_e \\ w_e \end{bmatrix} = J(q)\dot{q} \tag{3.5}$$

which 3.5 represents the manipulator differential kinematics equation. Where the $(6 \times n)$

matrix J is the manipulator geometric Jacobian

$$J = \begin{bmatrix} J_P \\ J_O \end{bmatrix}, \tag{3.6}$$

which in general is a function of the joint variables.

Secondly, to compute the Jacobian, the linear and the angular velocity can be computed as

$$\begin{cases} \dot{p}_e = \sum_{i-1}^{n} \frac{\partial p_e}{\partial q_i} \dot{q}_i = \sum_{i=1}^{n} J_{P_i} \dot{q}_i \\ w_e = w_n = \sum_{i=1}^{n} w_{i-1,i} = \sum_{i=1}^{n} J_{O_i} \dot{q}_i \end{cases} \tag{3.7}$$

The joint velocities $\dot{q}_i$ are expressed differently if the joints are prismatic or revolute. Therefore, by distinguishing the case of a prismatic joint ($q_i = d_i$) from the case of a revolute joint ($q_i = \vartheta_i$), the contribution to the linear velocity it is:

- **If joint i is prismatic**:

$$\dot{q}_i J_{P_i} = \dot{d}_i z_{i-1} \Leftrightarrow J_{P_i} = z_{i-1} \tag{3.8}$$

- **If joint i is revolute**:

$$\dot{q}_i J_{P_i} = w_{i-1,i} \times r_{i-1,e} = \dot{\vartheta}_i z_{i-1} \times (p_e - p_{i-1}) \Leftrightarrow J_{P_i} = z_{i-1} \times (p_e - p_{i-1}) \tag{3.9}$$

Where p_e is the distance from the origin of the end-effector coordinate frame to the base frame, and p_{i-1} the analogous distance from link $i-1$.

For the contribution to the angular velocity it is:

- **If joint i is prismatic**:

$$\dot{q}_i J_{O_i} = 0 \Leftrightarrow J_{O_i} = 0 \tag{3.10}$$

- **If joint i is revolute**:

$$\dot{q}_i J_{O_i} = \dot{\vartheta}_i z_{i-1} \Leftrightarrow \dot{q}_i J_{O_i} = z_{i-1} \tag{3.11}$$

In summary, the JACO^2 has six joints and all of them are revolute, therefore the

Jacobian can be translated into

$$J = \begin{bmatrix} J_{P_1} \cdots J_{P_n} \\ J_{O_1} \cdots J_{O_n} \end{bmatrix} \Leftrightarrow \begin{bmatrix} J_{P_i} \\ J_{O_i} \end{bmatrix} = \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} \tag{3.12}$$

Finally, notice that the Jacobian matrix depends on the frame in which the end-effector velocity is expressed. The above equations allow computation of the Jacobian with respect to the base frame. If it is necessary to represent the Jacobian in a different frame u, it is sufficient to know the relative rotation matrix R^u. Therefore, the relationship between velocities in the two frames is

$$\begin{bmatrix} \dot{p}_e^u \\ w_e^u \end{bmatrix} = \begin{bmatrix} R^u & O \\ O & R^u \end{bmatrix} \begin{bmatrix} \dot{p}_e \\ w_e \end{bmatrix}, \tag{3.13}$$

which, substituted in equation 3.5, gives

$$\begin{bmatrix} \dot{p}_e^u \\ w_e^u \end{bmatrix} = \begin{bmatrix} R^u & O \\ O & R^u \end{bmatrix} J\dot{q} \tag{3.14}$$

and then

$$J^u = \begin{bmatrix} R^u & O \\ O & R^u \end{bmatrix} J, \tag{3.15}$$

where J^u denotes the geometric Jacobian in frame u.

3.3 Dynamics

Now, it is time to describe the dynamics of robot manipulators. While the kinematic equations (section 3.1) describe the motion of the robot without consideration of the forces and moments and moments producing the motion, the dynamic equations explicitly describe the relationship between force and motion [11]. The dynamic equations are incorporated by the robot dynamic parameters which entail the mass, inertia, frictions and other unknown parameters that can negatively affect the robot's performance. In addition, the referred equations grant the possibility of designing additional control architectures for both joint space and task space.

The robot dynamics can be obtained by Euler-Lagrange (EL) or recursive Newton-Euler (NE) techniques, or even a combination of both.

The method used in this work to determine the robot dynamics is the EL, for this reason it is described in detail below.

In relation to the NE technique, the method is recursive which means that the computations are made from link to link. First, the velocities and accelerations of the augmented links [2] are calculated in an interactive way, from the base to the end-effector. Then, the iterative process is inverted, in order to find the resulting torques/forces exerted on the links.

Obviously, both methods must yield the same results. However, some discussions regarding the computational efficiency of both techniques. The EL method is computationally less efficient than the NE method, because apart from the fact that it is not recursive, it also makes all the necessary calculations referred to the base frame, which leads to very complex expressions when dealing with a 6DOF robot. Apart from that, the recursive Lagrangian methodology can achieve the same efficiency as the recursive NE technique [11] [12].

Euler-Lagrange

The Lagrangian mechanics is a re-formulation of Newton's Laws which takes into account energy relations [23]. The Lagrangian L is defined by

$$L(q, \dot{q}) = E(q, \dot{q}) - U(q) \tag{3.16}$$

where $E(q, \dot{q})$ and $U(q)$ are scalars representing respectively robot kinetic and potential energies. q is a vector $(n \times 1)$ of generalized coordinates, representing the positions of robot joints.

The kinetic energy is given by

$$E(q, \dot{q}) = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{3.17}$$

where $M(q)$ is the mass matrix $(n \times n)$, which is symmetric and positive definite. While the potential energy is given by

$$U(q) = -g_c^T r_c(q) m_c \tag{3.18}$$

where g_c is the gravity acceleration vector (3×1), $r_c(q)$ is the links center of gravity matrix $(3 \times n)$ and m_c is the links mass vector $(n \times 1)$.

[2]The term augmented links refers to a system composed of a link + actuator.

Then, the Lagrangian mechanics is represented in vector form by

$$\tau = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L(q,\dot{q})}{\partial \dot{q}} - \frac{\partial L(q,\dot{q})}{\partial q} \tag{3.19}$$

where τ is the generalized torque acting on q.

From equations 3.16, 3.17 and 3.19,

$$\frac{\mathrm{d}}{\mathrm{d}t}(M(q)\dot{q}) - \frac{\partial E(q,\dot{q})}{\partial q} + \frac{\partial U(q)}{\partial q} = \tau \tag{3.20}$$

Therefore, equation 3.20 can be written as

$$M(q)\ddot{q} + v(q,\dot{q}) + g(q) = \tau \tag{3.21}$$

where $M(q)$ is the mass matrix, $v(q,\dot{q})$ is the Coriolis and centripetal forces term and $g(q)$ is the term relating to the gravity forces.

The mass (or inertia) matrix $M(q)$ is given by

$$M(q) = \sum_{i=1}^{n}\left[m_i J_{v_i}(q)^T J_{v_i}(q) + J_{w_i}(q)^T R_i(q) I_i R_i(q)^T J_{w_i}(q)\right] \tag{3.22}$$

where m_i, $J_{v_{i}(q)}$, $J_{w_{i}(q)}$, $R_i(q)$ and I_i are respectively the mass of each link, the Jacobians associated with linear and angular velocities, the rotation matrix in base coordinates and the inertia tensor in link coordinates. It is important to refer that all these quantities are referred to the center of mass of link i.

The Coriolis and centripetal forces term $v(q,\dot{q})$ is given by

$$v(q,\dot{q}) = C(q,\dot{q})\dot{q} = \dot{M}(q)\dot{q} - \frac{\partial E(q,\dot{q})}{\partial q} \tag{3.23}$$

where $C(q,\dot{q})$ is the Coriolis matrix which has dimensions $(n \times 1)$ and is given by

$$C(q,\dot{q}) = \dot{M}(q) - \frac{1}{2}\dot{q}^T\frac{\partial M(q)}{\partial q} \tag{3.24}$$

Lastly, the gravity term $g(q)$ $(n \times 1)$ is represented by

$$g(q) = \frac{\partial U(q)}{\partial q} \tag{3.25}$$

4 Control Architectures

To achieve the goal of this work it is necessary first analyse the kinematic and dynamic model calculated or estimated in the previous chapter. For that it is needed a iterative process, where different control architectures from the simplest to the most complex are tested in order to validate the estimated model. But first it is necessary understand how this control architectures could be implemented either in a simulation environment (Gazebo simulator) or with the real robot since this is to be carried out in ROS environment.

To start, an explanation of how the different controllers are connected to ROS and the control architectures to be implemented are detailed.

4.1 ROS Control

Before starting to work with ROS it is necessary an in-depth study of how the interface between the robot and ROS is made. As mentioned in section 2.3 ROS offers a set of packages, called **ros_control** [18], which includes controller interfaces, controller managers, transmissions and hardware interfaces. Figure 4.1 briefly explains how ROS control works.

Regarding the real robot, once the kinova-ros stack is created on top of the kinova API, there is already a hardware interface, as well as a way to communicate with the robot, and there is also available examples of implemented controllers, which simplifies the work to be developed. With this, when the **kinova_robot.launch** file from the package **kinova_bringup** is initialized, this one sends to the server the drivers and settings needed to control the real robot. This makes available a topics and services list. This allows accessing for instance the joint states, such as the topic "**/j2n6s300_driver/out/joint_states**" which publishes information regarding the position, velocity and effort of each joint. Concerning the services, they allow for example to command the robot to the home position ("**rosservice call /j2n6s300_driver/in/home_arm**"), permit to changing from position to torque control ("**rosservice call j2n6s300_driver/in/set_torque_control_mode**").

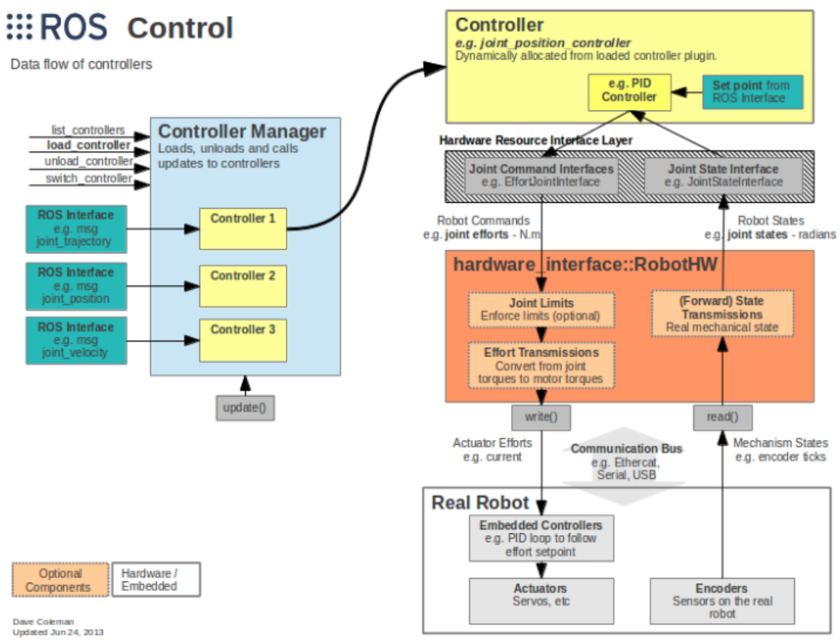

ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
(Forward) State Transmissions Real mechanical state
Effort Transmissions Convert from joint torques to motor torques
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 24, 2013

Figure 4.1: Diagram of the relationship between controller interfaces, controller managers, transmissions and hardware interfaces in ros_control [18].

Regarding the real robot, the **controller_manager** box as well as the **controller** box in figure 4.1 are substituted by the controllers detailed from section 4.2 to 4.5.

Comparatively to the Gazebo simulator, it is necessary first understand how the torques could be sent to the virtual robot. By analysing the list of controllers available in ros_control, it is concluded that is necessary to create a new package, named **thesis_control**, with the appropriate controllers to interface with Gazebo.

As a result it is needed to create a ".yaml" configuration file inside the new package, called **j2n6s300_control.yaml**, with the torque controllers for each joint, position controllers for each of the end-effector fingers, trajectory controller to the joints and fingers, and finally a controller to publish the joint states at a given frequency. A "roslaunch" file is also created to initialize the mentioned controllers and a "roslaunch" file to load the URDF model in the ROS parameter server and send the virtual robot to the Gazebo simulator. It is also necessary to mention that a node is created to command the robot to the home position when it is initialized in the simulator. Therefore, the control architectures described are introduced in the **controller** box in the figure 4.1.

The commands necessary for the user to launch the real robot and the virtual robot in the simulator running the controllers are available in the appendix D.

4.2 Computed torque control in the joint space

Like is mentioned in chapter 3, the robot dynamics is given by (see equation 3.21)

$$M(q)\ddot{q} + v(\dot{q}, q) + g(q) = \tau \tag{4.1}$$

- $M(q)$ is the mass matrix $(n \times n)$;
- n is the number of DOF;
- $v(\dot{q}, q)$ represents Coriolis and centripetal forces $(n \times 1)$;
- $g(q)$ is the gravity term $(n \times 1)$;
- and τ is the generalized torque acting on q, including friction (τ_f), computer commanded (τ_c) and external torques (τ_e).

$$\tau = \tau_c + \tau_f + \tau_e \tag{4.2}$$

Throughout this work, external and friction torques are not considered, i.e.,

$$\tau \simeq \tau_c \tag{4.3}$$

For this reason, equation 4.1 is denoted as

$$\tau_c = \hat{M}(q)\ddot{q} + \hat{v}(\dot{q}, q) + \hat{g}(q) \tag{4.4}$$

where $(\hat{.})$ means estimations of the dynamic model obtained from the KDL library referenced in the beginning of the chapter 3 and the theoretical background is mentioned in section 3.3.

Equation 4.4 illustrates a system non-linearly dependent upon the joint positions and velocities. Due to this linearisation and decoupled control are applied in order to cancel non-linear effects. As soon as the system is linearised, (see equation 4.5), using a non-linear feedback law the robot dynamics becomes

$$\tau_c = \hat{M}(q)w + \hat{g}(q) \tag{4.5}$$

where $w = \ddot{q}$ is the new control variable and the term $\hat{v}(\dot{q}, q)$ is ignored, because the experimental tests pursued with the JACO^2 robot are done with low speed motions, making this term less contributive. Equation 4.5 is known as computed torque control or inverse dynamics control. As can be easily seen, the direct application of this control law can be used for joint space control techniques.

For this reason, a simple dynamic model based on PID controller is developed, since it presents a good starting point to infer the viability of the estimated dynamic model.

Figure 4.2 shows a PID computed torque control in the joint space, based on a real-time dynamic model computation.

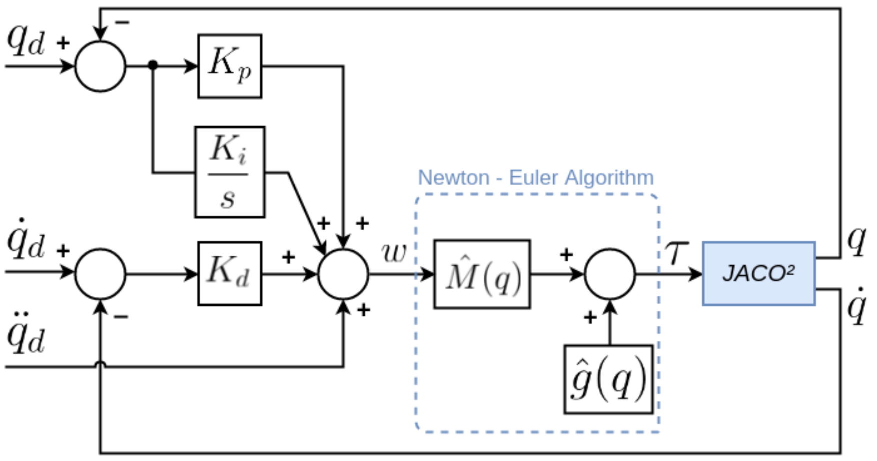


Figure 4.2: PID computed torque control in the joint space.

The inertia matrix, $\hat{M}(q)$ and the gravity term $\hat{g}(q)$ are computed in real-time with the control variable w being

$$w = \ddot{q}_d + K_d(\dot{q}_d - \dot{q}) + K_p(q_d - q) + K_i \int_{t_0}^{t} (q_d - q)d\lambda \tag{4.6}$$

- q_d, $\dot{q}_d$ and $\ddot{q}_d$ represent desired joint positions, velocities and accelerations;

- K_p, K_d and K_i are $(n \times n)$ positive definite diagonal matrices with diagonal gains K_{pj}, K_{dj} and K_{ij}, respectively.

Therefore, the following decoupled error dynamics is obtained

$$\ddot{e} + K_d\dot{e} + K_p e + K_i \int_{t_0}^{t} e d\lambda = 0 \tag{4.7}$$

with

$$e = q_d - q \tag{4.8}$$

Finally, in order to obtain the adequate gains for equation 4.6 a critically damped error dynamics with natural frequency w_j entails for each joint j,

$$K_{pj} = w_j^2 \tag{4.9}$$

$$K_{dj} = 2w_j \tag{4.10}$$

In relation to the K_{ij}, the gains are manually chosen to each joint j.

4.3 Computed torque control in the task space

Now, a dynamic control in task space is proposed. Knowing that the Jacobian J, (see section 3.2), relates task and joint velocities as

$$\dot{X} = J\dot{q} \tag{4.11}$$

The equation 4.1 can be written as

$$M(q)J^{-1}(\ddot{X} - \dot{J}\dot{q}) + v(\dot{q}, q) + g(q) = \tau \tag{4.12}$$

A control law that linearises and decouples the task space equations is given by

$$\hat{M}(q)J^{-1}w + \hat{g}(q) = \tau_c \tag{4.13}$$

Just as $v(\dot{q}, q)$ term, the derivative Jacobian $\dot{J}$ has a high computation cost and a small influence on the system, for this reason it is ignored over this work. Again, without estimation errors and neglecting or compensating friction and external torques a new variable is considered

$$w = \ddot{X} \tag{4.14}$$

With this for a PD control, w is given by

$$w = \begin{bmatrix} f_c \\ \mu_c \end{bmatrix} \tag{4.15}$$

where w is the concatenation of a 3D force vector f_c and a 3D torque vector μ_c. Figure 4.3

represents the real-time dynamic model in the task space.

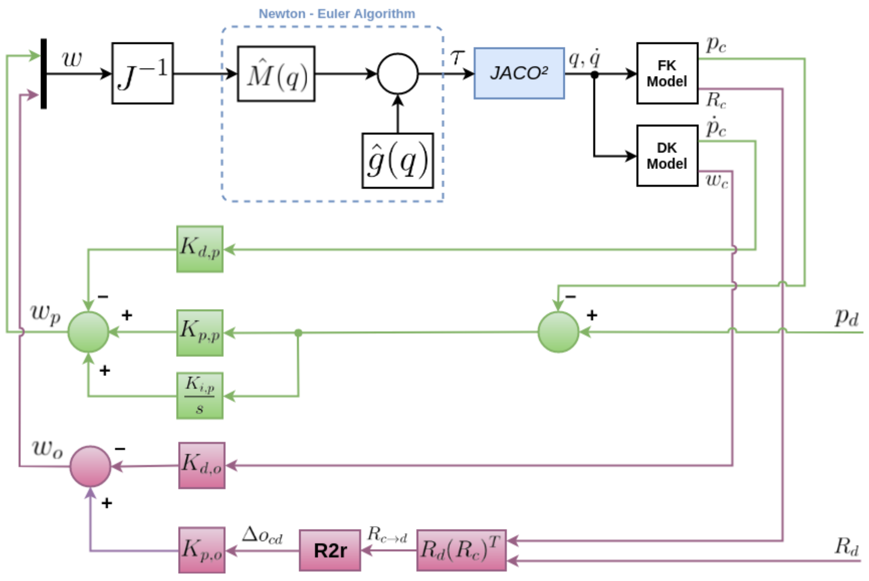


Figure 4.3: Position/Orientation PD control in the task space.

For a better understanding the position and orientation control are analysed separately. In figure 4.3, the green colour represents the position and the violet colour represents the orientation. And the boxes with **FK Model** and **DK Model** represent the Forward and Differential Kinematics, respectively.

Position Control

Being p_c and p_d respectively current and desired end-effector positions, the PD terms for the task space are computed by

$$f_c = K_{p,p} \Delta p_{cd} - K_{d,p} \dot{p}_c \tag{4.16}$$

with

$$\Delta p_{cd} = p_d - p_c \tag{4.17}$$

and $\dot{p}_c$ is the linear velocity in the task space, $K_{p,p}$ is the proportional gain and $K_{d,p}$ is the derivative gain, both related to position.

Orientation Control

In relation to the orientation error it requires additional formulation. Since rotation matrices are being used to represent orientation, the computation of the rotation from the current to the desired orientation, described in the current end-effector frame $R^c_{c\to d}$ [1] is given by

$$R^c_{c\to d} = R_c^{-1} R_d \tag{4.18}$$

where R_c is the current end-effector orientation and R_d the desired one. Equation 4.18 can be represented in the base frame using the following transformation [26]:

$$R_{c\to d} = R_c R^c_{c\to d} R_c^{-1} \tag{4.19}$$

There are various ways to represent the orientation of the end effector: Euler angles, angle-axis representation or quaternions. In this case it is chosen the angle-axis representation, which can be parametrized as

$$\begin{cases} \theta = cos^{-1}(\frac{R_{11}+R_{22}+R_{33}-1}{2}) \\ \nu = \frac{1}{2sin(\theta)} \begin{bmatrix} R_{32}-R_{23} \\ R_{13}-R_{31} \\ R_{21}-R_{12} \end{bmatrix} \end{cases} \tag{4.20}$$

where R refers to a rotation matrix like $R_{c\to d}$, θ is the angle, and ν is a unitary vector. Multiplying the vector by the axis, the rotation vector r is obtained

$$r = \theta\nu \tag{4.21}$$

Although the axis vector ν is not defined for the null rotation, it is possible to impose the rotation vector to be the null vector in such occasion, since in the vicinity of zero is

$$\lim_{\theta\to 0^+} \theta\nu = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.22}$$

The same applies for a rotation of an angle π, which is also not defined by 4.20. The function R2r(·) in figure 4.3 has been defined to implement 4.20 and 4.21 with zero values for $\theta = 0$ and $\theta = \pi$ [26]. To avoid this, in practice it is assumed a threshold of 0.03 rad.

$$r = R2r(R) \tag{4.23}$$

[1]The superscript c is related to the current frame. Matrices having no superscript are related to the base frame.

The function R2r(·) transforms a rotation matrix into a rotation vector. So, applying R2r(·) to $R_{c\to d}$ a rotation vector $r_{c\to d}$ can be obtained, which represents the orientation error Δo_{cd} between current end-effector frame and desired one,

$$\Delta o_{cd} \equiv r_{c\to d} = R2r(R_{c\to d}) \tag{4.24}$$

Finally, the orientation PD control is then defined as

$$\mu_c = K_{p,o}\Delta o_{cd} - K_{d,o}w_c \tag{4.25}$$

where w_c is the angular velocity in the task space, $K_{p,o}$ is the proportional gain corresponding to the orientation and $K_{d,o}$ is the derivative gain referring to the orientation.

4.4 Joint Space Control with Task Posture Reference

In previous sections (sections 4.2 and 4.3) different operational space control schemes are addressed. Acceptable results can be obtained using these kind of architectures, their control gain design is limited, however. In relation to the control in the joint space, each joint is controlled individually, which is not very useful for task operation purposes. On the other hand, the controller in the task space is in general more complex than controlling it in the joint space and the presence of singularities and/or redundancy influences the Jacobian.

Then, a new approach must be followed. The posture errors in the task space are converted into velocity references in the joint space. This approach merges task and joint space formalism into a single architecture.

Before starting, another orientation representation is introduced, the quaternion orientation.

Quaternion Orientation

Unit quaternions are a four parameter representation that comes with a complete quaternion algebra, enabling a better analysis and development of control algorithms [24]. The unit quaternion is defined as

$$Q = \left\{\eta, \epsilon\right\} \tag{4.26}$$

where η is called the scalar part

$$\eta = cos\left(\frac{\vartheta}{2}\right) \tag{4.27}$$

and $\epsilon = \begin{bmatrix} \epsilon_x & \epsilon_y & \epsilon_z \end{bmatrix}^T$ is the vector part

$$\epsilon = sin\left(\frac{\vartheta}{2}\right) r \tag{4.28}$$

ϑ is the angle of rotation and r is the unit vector of an equivalent angle-axis representation (see eq.4.20). Since unit quaternions are a four parameter representation for a sufficient three parameter representation, η and ϵ are not independent, being related to the following constrain

$$\eta^2 + \epsilon^T \epsilon = 1 \tag{4.29}$$

The Rotation matrix corresponding to a given quaternion is [24]

$$R = (\eta^2 - \epsilon^T \epsilon)I + 2\epsilon\epsilon^T + 2\eta S(\epsilon) \tag{4.30}$$

where I is the identity matrix and $S(\cdot)$ is the skew-symmetric matrix operator. The skew-symmetric operator can be written as

$$S(\epsilon) = \begin{bmatrix} 0 & -\epsilon_z & \epsilon_y \\ \epsilon_z & 0 & -\epsilon_x \\ -\epsilon_y & \epsilon_x & 0 \end{bmatrix} \tag{4.31}$$

The quaternion corresponding to a given rotation matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.32}$$

can be obtained by [24]

$$\eta = \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1}, \tag{4.33}$$

and

$$\epsilon = \frac{1}{2}\begin{bmatrix} sgn(r_{32}-r_{23})\sqrt{r_{11}-r_{22}-r_{33}+1} \\ sgn(r_{13}-r_{31})\sqrt{r_{22}-r_{33}-r_{11}+1} \\ sgn(r_{21}-r_{12})\sqrt{r_{33}-r_{11}-r_{22}+1} \end{bmatrix} \tag{4.34}$$

where

$$sgn(x) = \begin{cases} 1 \quad if \quad x \geq 0 \\ -1 \quad if \quad x < 0 \end{cases} \tag{4.35}$$

The quaternions $\left\{\eta, \epsilon\right\}$ and $\left\{-\eta, -\epsilon\right\}$ are both a representation of the same orientation. Since equation (4.33) implies that $\eta \geq 0$, the angle $\vartheta \in \left[-\pi, \pi\right]$, allowing the representation of any rotation and solving the ambiguity problem of two different quaternions representing the same orientation. The quaternion Q^{-1} represents R^{-1} and can be computed as

$$Q^{-1} = \left\{\eta, -\epsilon\right\} \tag{4.36}$$

where Q^{-1} is the conjugate of Q. Now, let $Q_a = \left\{\eta_a, \epsilon_a\right\}$ and $Q_b = \left\{\eta_b, \epsilon_b\right\}$ be the corresponding quaternions of rotation matrices R_a and R_b. In the quaternion framework, the product R_aR_b is

$$Q_a * Q_b = \left\{\eta_a\eta_b - \epsilon_a^T\epsilon_b, \eta_a\epsilon_b + \eta_b\epsilon_a + \epsilon_a \times \epsilon_b\right\} \tag{4.37}$$

where $*$ is the quaternion product operator. The product of a quaternion by its conjugate gives the identity element of the product of quaternions

$$Q * Q^{-1} = \left\{1, 0\right\} \tag{4.38}$$

Control Architecture

It is time to explain the controller suggested. With this, the equation 4.1 can be written as

$$\tau_c = \hat{M}(q)\alpha + \hat{g}(q) \tag{4.39}$$

where α is the new control variable and like in last sections the $\hat{v}(\dot{q}, q)$ term is ignored. The following linear and decoupled system is achieved

$$\ddot{q} = \alpha \tag{4.40}$$

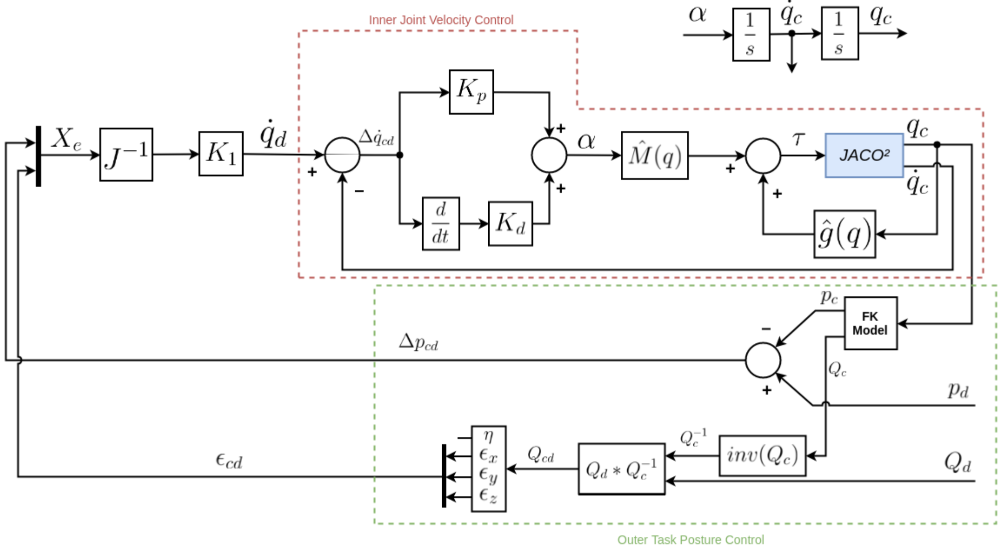


Figure 4.4: PD joint space torque control with task space posture reference.

Figure 4.4 shows the control architecture suggested. It is easy to realise that the inner joint velocity control depends on the outer task posture control. Then the task space error is given by

$$X_e = \begin{bmatrix} \Delta p_{cd} \\ \epsilon_{cd} \end{bmatrix} \tag{4.41}$$

where Δp_{cd} and ϵ_{cd} are the task space position and orientation errors, respectively. In this way a velocity vector is generated, which is collinear with the task space error vector. The task space position error Δp_{cd} is given by

$$\Delta p_{cd} = p_d - p_c \tag{4.42}$$

where p_d and p_c are the reference and current task space positions. The task space orientation error ϵ_{cd} is computed based on unit quaternion theory. The mutual orientation between the current Q_c and the desired orientation Q_d, expressed in the current frame $c_{Q_{cd}} = \left\{\eta_{cd}, c_{\epsilon_{cd}}\right\}$, is given by

$${}^{c}\mathrm{Q}_{cd} = Q_c^{-1} * Q_d \tag{4.43}$$

Equation 4.43 can be described in the base frame by

$$Q_{cd} = Q_c *^c Q_{cd} * Q_c^{-1} = Q_d * Q_c^{-1} \tag{4.44}$$

Knowing that $Q_{cd} = \left\{1, 0\right\}$ only if the desired frame is aligned with the current frame ($Q_c = Q_d$), the orientation error can be defined as [24]

$$\epsilon_{cd} = \eta_c \epsilon_d - \eta_d \epsilon_c - S(\epsilon_d)\epsilon_c \tag{4.45}$$

where $S(\cdot)$ is the skew-symmetric operator. Then, the joint velocity reference $\dot{q}_d$ can be computed by the following relation

$$\dot{q}_d = K_1 J^{-1} X_e \tag{4.46}$$

where K_1 is a $(n \times n)$ diagonal matrix and represents a proportional control gain whose elements are independently set for each joint. And J^{-1} is the inverse of the Jacobian matrix.

Finally, the resolved acceleration α is computed by a PD controller over the velocity error in the joint space.

$$\alpha = K_p \Delta\dot{q}_{cd} + K_d \Delta\ddot{q}_{cd} \tag{4.47}$$

with

$$\Delta\dot{q}_{cd} = \dot{q}_d - \dot{q}_c \tag{4.48}$$

and

$$\Delta\ddot{q}_{cd} = \frac{d}{dt}(\dot{q}_d - \dot{q}_c) \tag{4.49}$$

K_p and K_d are $(n \times n)$ diagonal matrices whose elements are the proportional and derivative gains for each joint controller.

4.5 Impedance Control

In the previous sections different control architectures were suggested, but none of them allow external interactions with the environment. As a result the impedance control is the next step. The idea of this control is to assign a prescribed robot dynamic behaviour in the presence of external interactions, matching the dynamics of mass-spring-damper systems [23].

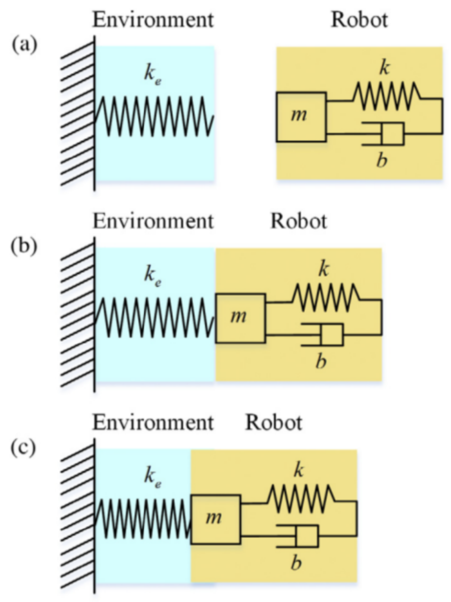
According to this, the end-effector velocity $\dot{X}$ and the robot applied force F_e are related to a mechanical impedance Z. In the Laplace domain,

$$-F_e(s) = Z(s)\dot{X}(s) \tag{4.50}$$

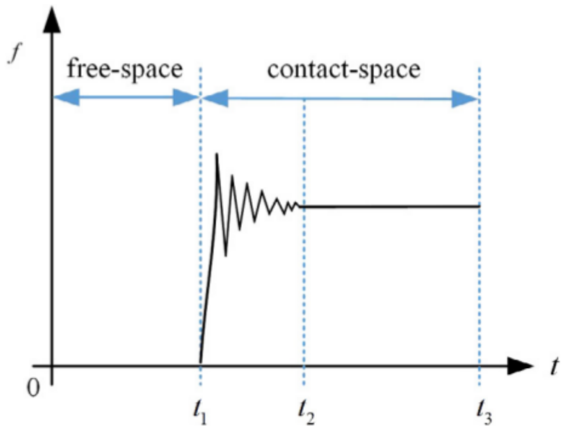
with

$$sZ(s) = As^2 + Ds + K \tag{4.51}$$

where A, K and D are the parameters of a mass-spring-damper system. Figure 4.5 shows the relation between the robot and the environment.



(a) System model of a robot and rigid environment: (a) without any contact between robot and environment, (b) critical point when contact occurs but $f = 0$ and (c) contact with $f \neq 0$.



(b) The diagram of robot and environment contact force.

Figure 4.5: The two pictures above shows the 3 different situations that occur between robot and environment. [3]

The figure 4.6 shows the principle of the impedance control, where the overall dynamics in contact can be written as

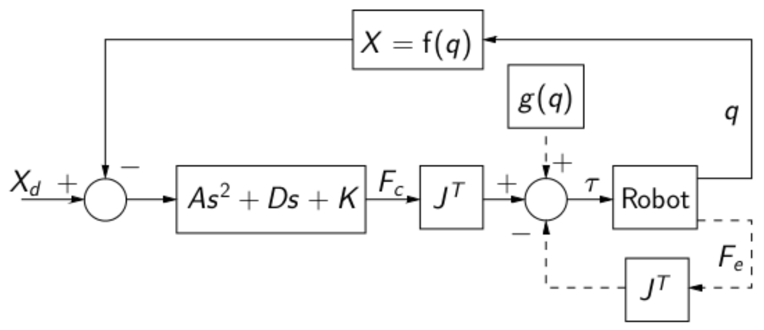


Figure 4.6: Impedance control scheme in the task space without force sensing. [23]

$$M(q)\ddot{q} + v(\dot{q}, q) + g(q) = \tau_c - J^T F_e \tag{4.52}$$

with

$$\tau_c = J^T \Big[A(\ddot{X}_d - \ddot{X}) + D(\dot{X}_d - \dot{X}) + K(X_d - X) \Big] + g(q) \tag{4.53}$$

Neglecting robot dynamic effects, 4.52 and 4.53 gives

$$A(\ddot{X}_d - \ddot{X}) + D(\dot{X}_d - \dot{X}) + K(X_d - X) \approx F_e \tag{4.54}$$

where F_e is the robot applied force in equation 4.50. Analysing the figure 4.6 in detail, if there is no contact the robot under impedance control generates a force F_c according to the mass-spring-damper system, based on X, $\dot{X}$ and $\ddot{X}$ displacements around the equilibrium point.

Impedance control scheme in the task space without force sensing and without inertia shaping

The mass term A is not present in the control scheme, figure 4.7, not only because this term is hard to implement in practice but also because it is important to go from a simpler to a more complex controller without tripping over the smallest details.

Similar procedure as equations 4.5, 4.13 and 4.39, corresponding to resolved accelera-

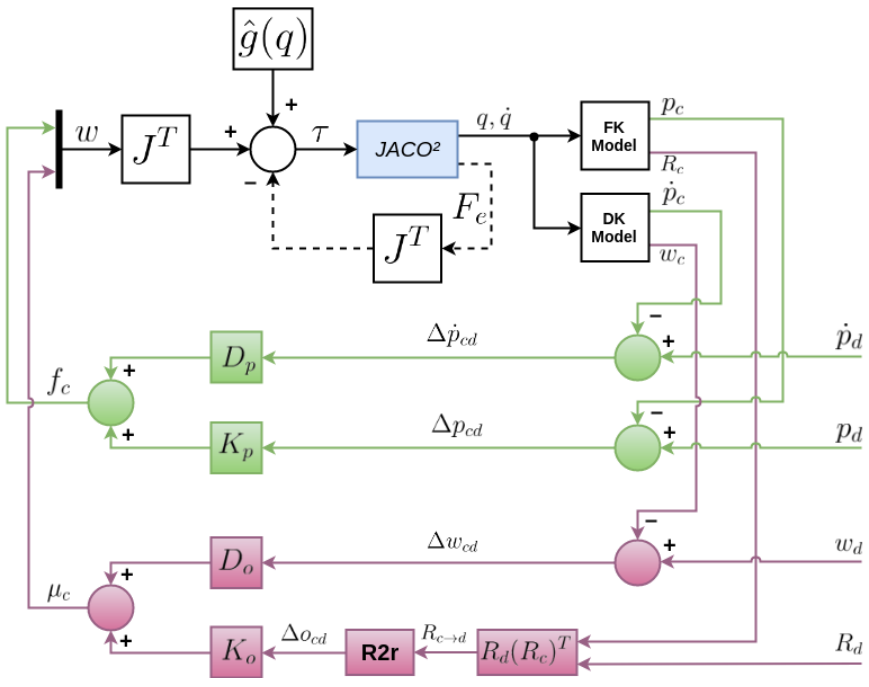


Figure 4.7: Impedance control architecture in the task space without force sensing and without inertia shaping.

tion control, another control variable w can be considered

$$\tau_c = J^T w + \hat{g}(q) \tag{4.55}$$

J^T is the transpose Jacobian, $\hat{g}(q)$ is the estimated gravity term and the variable $w = \ddot{X}$ is given by

$$w = \begin{bmatrix} f_c \\ \mu_c \end{bmatrix} \tag{4.56}$$

where w, like in equation 4.15, is the concatenation of a 3D force vector f_c and a 3D torque vector μ_c. So, analysing the position control first, (see the green blocks in figure 4.7), the vector f_c is given by

$$f_c = D_p \Delta\dot{p}_{cd} + K_p \Delta p_{cd} \tag{4.57}$$

$\Delta\dot{p}_{cd}$ is the linear task velocity error and can be written as

$$\Delta\dot{p}_{cd} = \dot{p}_d - \dot{p}_c \tag{4.58}$$

$\dot{p}_d$ is the desired linear velocity in the task space and $\dot{p}_c$ is the current linear velocity in the task space, which are obtained using the differential kinematics (see equation 3.4). Δp_{cd} is the position error in the task space (see equation 4.17). In relation to D_p and K_p, they are diagonal matrices that represent the position damping and spring terms in task space. In this specific control scheme they are similar to the derivative and proportional gains.

Now, the orientation control, (see the violet blocks in figure 4.7), the vector μ_c is given by

$$\mu_c = D_o \Delta w_{cd} + K_o \Delta o_{cd} \tag{4.59}$$

Δw_{cd} is the angular task velocity error and can be written as

$$\Delta w_{cd} = w_d - w_c \tag{4.60}$$

The current angular velocity w_c is similarly computed as the linear velocity (see equation 3.4) and w_d is the desired angular velocity in the task space. Again the orientation representation chosen is the angle-axis, thus Δo_{cd} is calculated in the same way as in equation 4.24. The D_o and K_o are diagonal matrices that represent orientation damper and spring terms in the task space, like derivative and proportional gains.

Last but not least, the F_e term in figure 4.7 is physic and it represents the force that the robot end-effector feels when it is in contact with the environment.

Impedance control scheme in the task space with force sensing

It is time to add the force sensing. Using force sensing and the full robot dynamics model, an impedance control scheme can be derived. Starting from the robot dynamics in contact,

$$M(q)\ddot{q} + v(\dot{q}, q) + g(q) + J^T F_e = \tau_c \tag{4.61}$$

where

$$A(\ddot{X}_d - \ddot{X}) + D(\dot{X}_d - \dot{X}) + K(X_d - X) = F_e \tag{4.62}$$

the previous equation 4.62 can be written as

$$\ddot{X} = \ddot{X}_d + A^{-1}\Big[D(\dot{X}_d - \dot{X}) + K(X_d - X) - F_e\Big] \tag{4.63}$$

Then, using a similar procedure to equation 4.55 corresponding to resolved acceleration control a new control variable w is considered

$$\tau_c = \hat{M}(q)J^{-1}w + \hat{g}(q) + J^T F_e \tag{4.64}$$

where

$$w = a_d + A^{-1}\begin{bmatrix} f_c - f_e \\ \mu_c - \mu_e \end{bmatrix} \tag{4.65}$$

- f_c is a 3D force vector;
- μ_c is a 3D torque vector;
- a_d is the desired acceleration in the task space;
- $\begin{bmatrix} f_e \\ \mu_e \end{bmatrix} = F_e$, is the end-effector force;
- A^{-1} is the inverse of the mass term. The mass term A is a $(n \times n)$ diagonal matrix.

During this work the inverse of the mass term A^{-1} is given by

$$A^{-1} = JM^{-1}J^T \tag{4.66}$$

Considering this equality, there is no inertia shaping in the task space. In this case, the F_e terms in 4.64 and 4.65 are eliminated, due to this no force feedback is needed [23].

Therefore, without inertia shaping the equation 4.65 can be re-written as

$$w = a_d + A^{-1}\begin{bmatrix} f_c \\ \mu_c \end{bmatrix} \tag{4.67}$$

where

$$f_c = D_p\Delta\dot{p}_{cd} + K_p\Delta p_{cd} + I_p\int_{t_0}^{t}\Delta p_{cd}d\lambda \tag{4.68}$$

I_p is a (3×3) diagonal matrix with position integral gains. This term and the sum of the position error are responsible for eliminating the magnitude and duration of the error over time. Apart from this, the vector f_c is similar to equation 4.57. In relation to the 3D torque vector μ_c is given by

$$\mu_c = D_o\Delta w_{cd} + K_o\Delta o_{cd} \tag{4.69}$$

wish can be calculated likewise in equation 4.58, since the orientation representation chosen is the angle-axis.

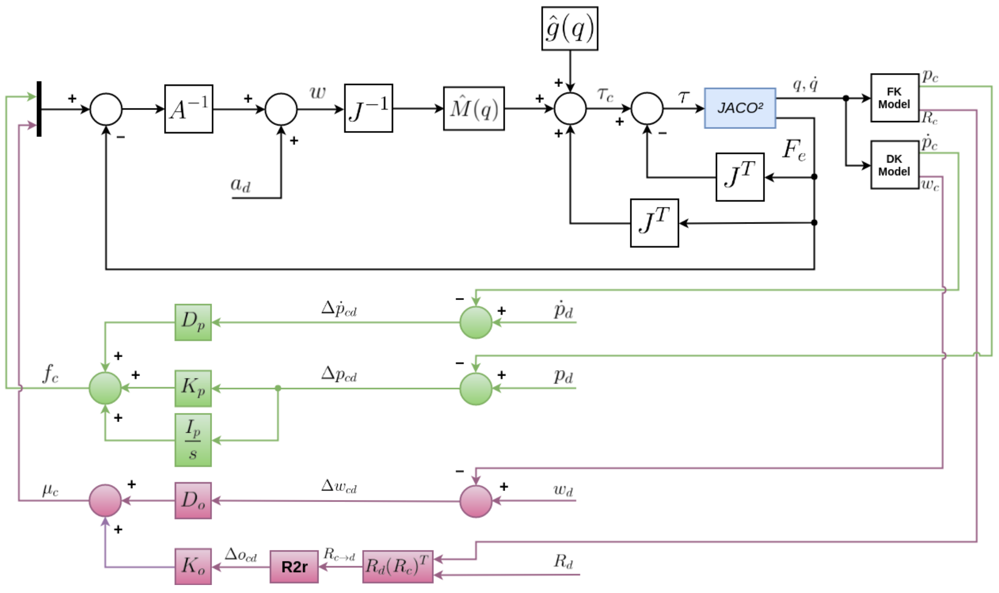


Figure 4.8: Impedance control scheme in the task space with force sensing.

Impedance control scheme in the task space with force sensing for redundant robots

The control scheme in figure 4.8 requires the computation of J^{-1}, which is not possible for redundant robots. From the control point of view, a redundant robot means that it has more DOF than the task space. Even though JACO² is a non-redundant robot, another approach is proposed, because this new control scheme, figure 4.9, can be used in all kind of robot manipulators.

Again, the robot dynamics in contact can be written as

$$M(q)\ddot{q} + v(\dot{q}, q) + g(q) + J^T F_e = \tau_c \tag{4.70}$$

and pre-compensating $v(\dot{q}, q)$, $g(q)$ and $J^T F_e$,

$$M(q)\ddot{q} = \tau' \tag{4.71}$$

with

$$\tau_c = \tau' + v(\dot{q}, q) + g(q) + J^T F_e \tag{4.72}$$

Shifting the inertia matrix $M(q)$ to the right side of equation 4.71, and multiplying by Jacobian J,

$$J\ddot{q} = JM(q)^{-1}\tau' \tag{4.73}$$

Since

$$\dot{X} = J\dot{q} \rightarrow J\ddot{q} = \ddot{X} - \dot{J}\dot{q} \tag{4.74}$$

and applying an additional Cartesian force F_c through τ', the equation 4.73 becomes

$$\ddot{X} - \dot{J}\dot{q} = JM(q)^{-1}J^T F_c \tag{4.75}$$

Therefore, from equation 4.75, the dynamic equation in the task space can be written as

$$\Lambda(q)\ddot{X} - \Lambda(q)\dot{J}\dot{q} = F_c \tag{4.76}$$

where $\Lambda(q)$ is the inertia matrix of the task space, and has the following inverse

$$\Lambda(q)^{-1} = JM(q)^{-1}J^T \tag{4.77}$$

which always exists, even if J is non-square ($M(q)^{-1}$ always exists) [23].

Considering $w = \ddot{X}$ the new control variable, ignoring the terms $\dot{J}\dot{q}$ and $v(\dot{q}, q)$ for the same reasons as in previous sections, the resolved acceleration control architecture can be implemented without the computation of J^{-1}. As a result the expression 4.72 can be re-written as

$$\tau_c = J^T F_c + \hat{g}(q) + J^T F_e \tag{4.78}$$

with

$$F_c = \Lambda(q)w \tag{4.79}$$

Figure 4.9 shows the control scheme for redundant robots.

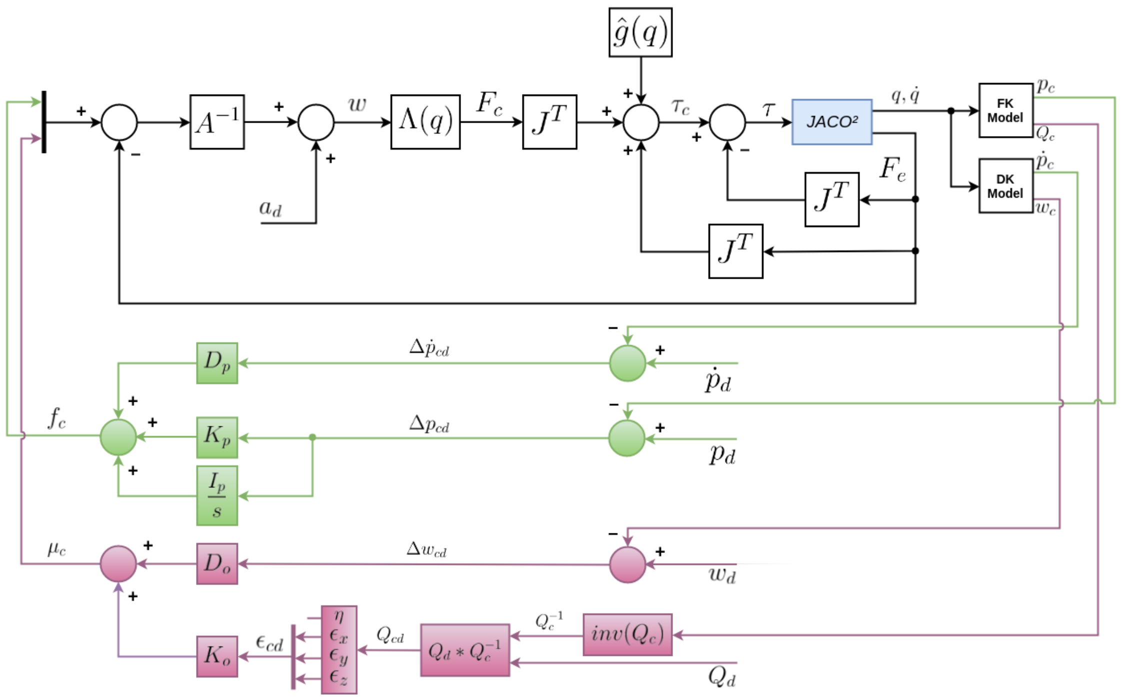


Figure 4.9: Impedance control scheme in the task space with force sensing for redundant robots and with quaternion orientation.

Since there is no inertia shaping ($A = \Lambda(q)$) no force feedback is needed, so the terms with F_e in 4.78 are cancelled. Seeing that, w is given by

$$w = a_d + A^{-1} \begin{bmatrix} f_c \\ \mu_c \end{bmatrix} \tag{4.80}$$

The vector f_c is computed as the same way as in equation 4.68 and the vector μ_c can be written as

$$\mu_c = D_o \Delta w_{cd} + K_o \epsilon_{cd} \tag{4.81}$$

where Δw_{cd} is equal to 4.59 and the orientation error ϵ_{cd} is equal to 4.45.

The orientation chosen in figure 4.9 is the quaternions, because comparing with the angle-axis representation, (see equation 4.20), no singularity occurs.

5 Experimental Results

In this chapter the control architectures proposed are evaluated. Firstly, this control schemes are tested in the Gazebo simulator and after in the real JACO^2 robot.

Also, it is necessary to mention that it is created a package inside the kinova-ros stack, named "**thesis_demo**". This package contains all the files or algorithms necessary to test the controllers mentioned in previous chapter, and also the files necessary to calculate and estimate the robot kinematics, differential kinematics and dynamics.

5.1 Gazebo Simulator Results

As stated in section 4.1, the **thesis_control** package is created in order to send binaries or torques to the robot in the simulator. This package allows to perform a switch controller, in other words, stop a controller and start another. Therefore, when the robot is launched in the simulator it is initialized with a trajectory controller in each of the robot joints. In order to send torques to the robot joints a ROS node is created. Then a switch controller from trajectory control to torque control is started. This is possible because **ros_control** provides a **controller_manager** which allows to interact with the different controllers. Without this switch the robot would start dead or dropped on the ground in the simulator, because the torques at each joint would be zero.

After the switch starts the control loop where the torques are generated depending on the control architecture.

5.1.1 Computed torque control in the joint space (simulator)

The first control scheme suggested it is a PID computed torque control in the joint space, figure 4.2. The estimated dynamic model and the robot performance are analysed for this control architecture. So as to perform it a desired trajectory is defined to infer the capabilities of the robotic arm. The trajectory defined is a sinusoidal wave for each joint.

Before starting the robot joints needs to be in a more favourable position due to robot geometry. So a third degree polynomial is denoted as

$$q_d(t) = q_i + \frac{3(q_f - q_i)(t - t_i)^2}{(t_f - t_i)^2} - \frac{2(q_f - q_i)(t - t_i)^3}{(t_f - t_i)^3} \tag{5.1}$$

Then, for each joint a sinusoidal wave is generated with the form

$$q_d(t) = q_i + Asin(wt) \tag{5.2}$$

where the angular frequency $w = \frac{2\pi}{T}$ with T being the period of the trajectory.

Figure 5.1 shows the performance of each joint of the robot during the defined trajectory. From $t = 10s$ to $t = 25s$ the trajectory is defined according to equation 5.1, whose parameters are

$$\begin{cases} q_i = \begin{bmatrix} 4.8046852 & 2.92482 & 1.002 & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T (rad) \quad \rightarrow \quad "Home" \\ q_f = \begin{bmatrix} \pi & \pi & \pi & \pi & \pi & \pi \end{bmatrix}^T (rad) \\ t_i = 10s \quad ; \quad t_f = 25s \end{cases} \tag{5.3}$$

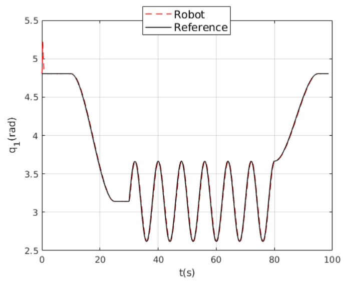
Then, from $t = 30s$ to $t = 80s$, the trajectory is defined according to equation 5.2, with the following parameters:

$$\begin{cases} q_i = \begin{bmatrix} \pi & \pi & \pi & \pi & \pi & \pi \end{bmatrix}^T (rad) \\ A = \frac{\pi}{6}(rad) \quad ; \quad T = 8s \end{cases} \tag{5.4}$$

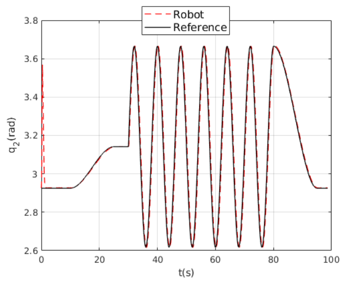
Finaly, from $t = 80s$ to $t = 95s$ the robot returns to his "Home" position following again a third degree polynomial (eq. 5.1), with the given conditions:

$$\begin{cases} q_i = q_{d_{last}} \\ q_f = "Home" \\ t_i = 80s \quad ; \quad t_f = 95s \end{cases} \tag{5.5}$$

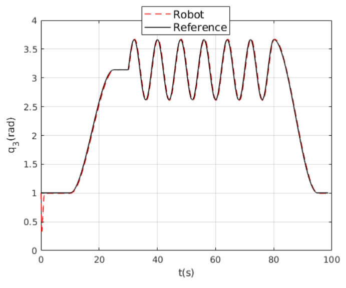
It is important, that during the first milliseconds the switch controller occurs, which is notorious analysing the plots of the figure 5.1. Although the switch occurs, the controller corrects well the position of the robot joints and along the defined trajectory achieves a good position tracking with stable and smooth movements.



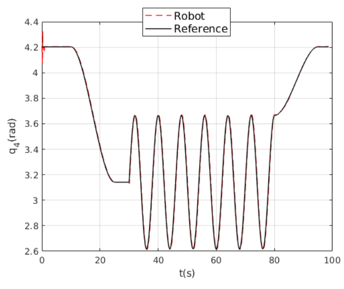
(a) Joint 1.



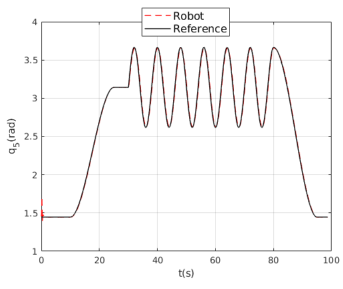
(b) Joint 2.



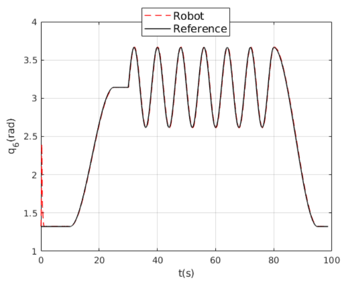
(c) Joint 3.



(d) Joint 4.



(e) Joint 5.



(f) Joint 6.

Figure 5.1: PID computed torque control in the joint space (Gazebo Simulator).

In relation to table 5.1 it is verified that the gains of the last three joints are higher compared to the first three joints, which can cause problems in the real robot since the movement of each joint is influenced by the movement of the other joints. This problem can be associated to the fact that the inertia matrix corresponds to an approximation assuming uniform cylinders for the robot links (see section 2.3). This issue is further discussed in the real robot.

Table 5.1: The control gains were done by the following expressions: $K_{p_j} = w_{n_j}^2$ and $K_{d_j} = 2w_{n_j}$.

Joint j	w_{n_j}	K_{i_j}
1	16	0.005
2	10	0.005
3	10	0.005
4	20	0.005
5	20	0.005
6	20	0.005

5.1.2 Computed torque control in the task space (simulator)

The controller suggested in the previous chapter is a PD computed torque control in the task space, figure 4.3. A desired trajectory is defined again to infer the robot performance. The desired trajectory is a spline (see equation 5.1) both in position and orientation and then a circular trajectory is proposed, based on the following expression

$$p_d = p_i + \begin{bmatrix} 0 \\ a \cdot cos(wt) - a \\ b \cdot sin(wt) \end{bmatrix} \tag{5.6}$$

where $w = \dfrac{2\pi}{T}$, with T being the period of the trajectory, and a and b correspond to the radius of the circle, so $a = b$. The equation 5.6 could be used also to define a ellipse trajectory, and in this case $a \neq b$.

In figure 5.2, from $t = 10s$ to $t = 18s$ the spline parameters are

$$\begin{cases} \begin{bmatrix} p_i(m) \\ r_i(rad) \end{bmatrix} = \begin{bmatrix} 0.2127 & -0.2566 & 0.5069 & 2.0292 & -0.6806 & 2.0476 \end{bmatrix}^T \\ \begin{bmatrix} p_f(m) \\ r_f(rad) \end{bmatrix} = \begin{bmatrix} 0.3127 & -0.3566 & 0.6069 & 1.5249 & 0.7589 & 1.0022 \end{bmatrix}^T \\ \qquad t_i = 10s \quad ; \quad t_f = 18s \end{cases} \tag{5.7}$$

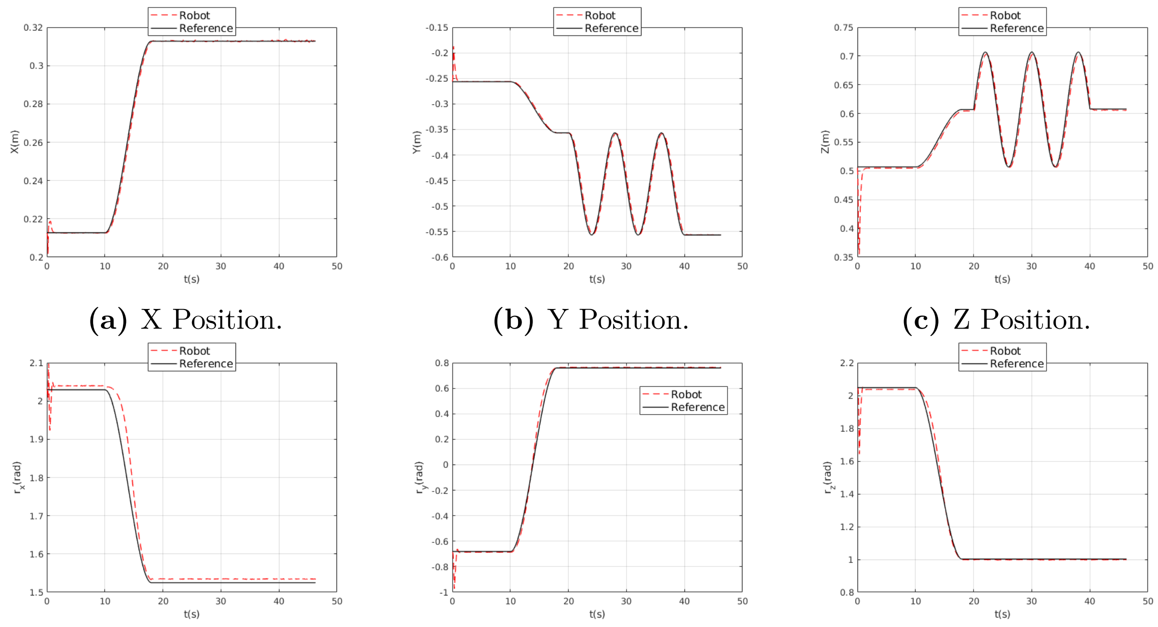
Then, from $t = 20s$ to $t = 40s$ a circular position trajectory (see eq. 5.6) in y and z axis is defined, whose parameters are

$$\begin{cases} p_i = \begin{bmatrix} 0.3127 & -0.3566 & 0.6069 \end{bmatrix}^T \\ \quad a = b = 0.1(m) \quad ; \quad T = 8s \end{cases} \tag{5.8}$$

As can be seen in figure 5.2 the steady state error is good, both in position and orientation. The orientation is based on angle-axis representation and the figures 5.2d, 5.2e and 5.2f correspond to the end-effector rotation vector, (see eq. 4.21). The tracking response is good, with smooth end-effector movements. Here the underestimated dynamic model does not seem to have much influence on the controller gains, table 5.2.

Table 5.2: Position and Orientation Gains.

Position	Gain	Orientation	Gain
$\boldsymbol{K_{p,p}}$	200	$\boldsymbol{K_{p,o}}$	100
$\boldsymbol{K_{d,p}}$	40	$\boldsymbol{K_{d,o}}$	10



(a) X Position. **(b)** Y Position. **(c)** Z Position.

(d) X Orientation. **(e)** Y Orientation. **(f)** Z Orientation.

Figure 5.2: Position/Orientation PD computer torque control in the task space (Gazebo Simulator).

5.1.3 Joint space control with task posture reference (simulator)

In the control scheme presented in figure 4.4 a task posture reference is designed similarly to the task space control in section 4.3, with the task space errors being converted into joint velocity references, by virtue of the known differential kinematic relation. This conduces to a control of each joint, just like the control of the section 4.2, but having the advantage of possessing a task space reference.

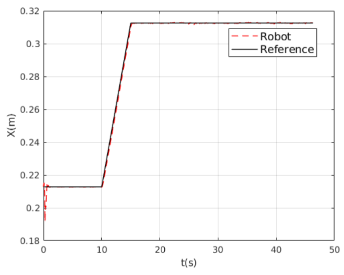
The desired trajectory for this controller it is almost the same of the previous controller, but instead of a spline it is suggested a rectilinear movement. The equation of a line can be given by

$$p_d = p_i(1-\alpha) + p_f\alpha, \quad \alpha \in [0,1] \tag{5.9}$$

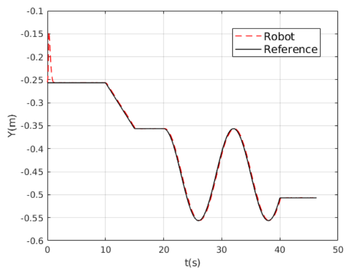
where p_d is the desired position, p_i is the line start point, p_f is the line final point and α is the step time.

In figure 5.3, from $t = 10s$ to $t = 15s$ is represented a position control for a line, with the following parameters:

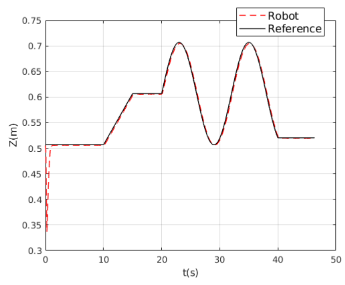
$$\begin{cases} p_i = \begin{bmatrix} 0.2127 & -0.2566 & 0.5069 \end{bmatrix}^T \\ p_f = \begin{bmatrix} 0.3127 & -0.3566 & 0.6069 \end{bmatrix}^T \end{cases} \tag{5.10}$$



(a) X Position.



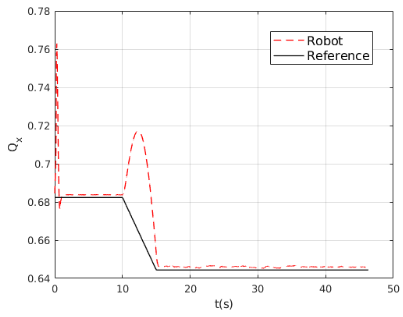
(b) Y Position.



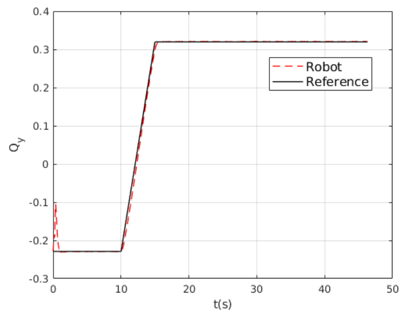
(c) Z Position.

Figure 5.3: Position PD joint space torque control with task space posture reference (Gazebo Simulator).

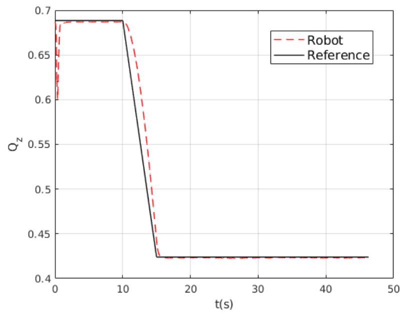
Then, between $t = 20s$ to $t = 40s$ a circular path is defined with the same trajectory conditions of 5.8, with the exception that the trajectory period is $T = 12s$. In relation to the results, the control algorithm performs a good position tracking and the robot follows the reference with smooth movements. Concerning the orientation control, figure 5.4.



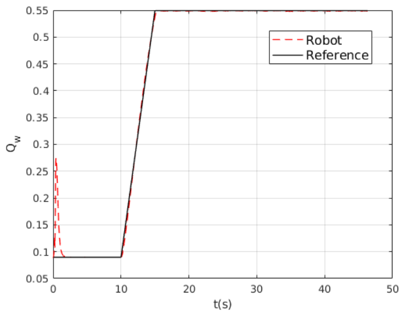
(a) ϵ_x Orientation.



(b) ϵ_y Orientation.



(c) ϵ_z Orientation.



(d) η Orientation.

Figure 5.4: Orientation PD joint space torque control with task space posture reference (Gazebo Simulator).

The desired orientation trajectory is also a line, whose parameters are:

$$\begin{cases} Q_i = \begin{bmatrix} 0.6823 & -0.2288 & 0.6884 & 0.0896 \end{bmatrix}^T \\ Q_f = \begin{bmatrix} 0.6443 & 0.3206 & 0.4234 & 0.5502 \end{bmatrix}^T \end{cases} \tag{5.11}$$

Analysing the figure 5.4, the steady state error is good but the orientation associated to the quaternion Q_x, (figure 5.4a), the robot has difficulties in following the reference. However, the robot has a good orientation tracking in the other axis.

Table 5.3: Joint control architecture gains.

Joints	K_1	K_p	K_d
Joint 1	7	50	0.02
Joint 2	7	50	0.02
Joint 3	7	50	0.02
Joint 4	10	50	0.5
Joint 5	7	50	0.5
Joint 6	7	50	0.5

5.1.4 Impedance control in the task space without force sensing (simulator)

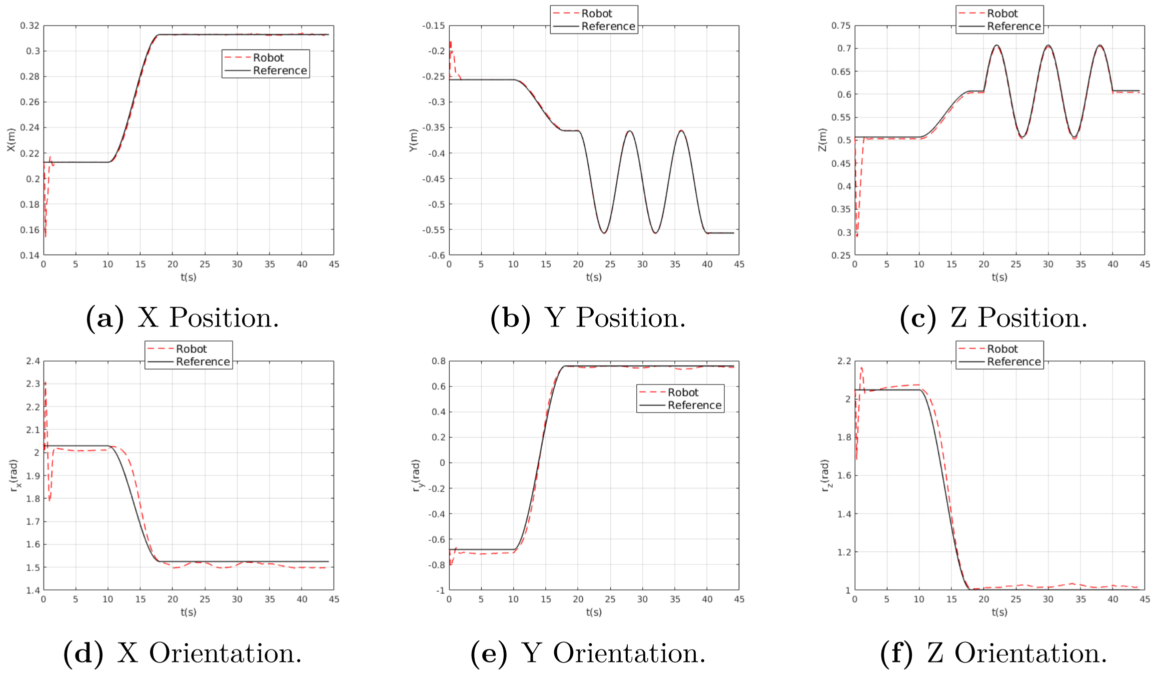
It is time to test the control scheme shown in figure 4.7, this architecture corresponds to the basis of the impedance control. Since the term A associated to the mass and the inertia matrix M is not present in this scheme, this control scheme corresponds to a compliance control. To analyse the performance of the robot in this control algorithm is defined the same trajectory as in subsection 5.1.2.

It is important to mention that besides position/orientation tracking is also done the tracking of the linear velocity associated to the movement of the robot in the task space. The linear velocity corresponds in this case to the derivative of the position during the circular trajectory, whose expression is given by

$$p_d = p_i + \begin{bmatrix} 0 \\ a \cdot cos(wt) - a \\ b \cdot sin(wt) \end{bmatrix} \; ; \quad \dot{p}_d = \dot{p}_i + \begin{bmatrix} 0 \\ -w \cdot a \cdot sin(wt) \\ w \cdot b \cdot cos(wt) \end{bmatrix} \tag{5.12}$$

where the parameters a, b and w are the same of equation 5.8 and the initial linear velocity is $\dot{p}_i = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

The results obtained in figure 5.5 are practically the same as the results obtained in the figure 5.2. Although the inertia matrix M has not been used in this control scheme, the robot has a good tracking during the defined trajectory. However, this is a simulated environment where the inertia and friction of the actuators are different in the case of the real robot.



(a) X Position. **(b)** Y Position. **(c)** Z Position.

(d) X Orientation. **(e)** Y Orientation. **(f)** Z Orientation.

Figure 5.5: Position/Orientation impedance control without force sensing (Gazebo Simulator).

Table 5.4: Position and Orientation Gains.

Position	Gain	Orientation	Gain
$\boldsymbol{K_p}$	150	$\boldsymbol{K_o}$	8
$\boldsymbol{D_p}$	24	$\boldsymbol{D_o}$	2

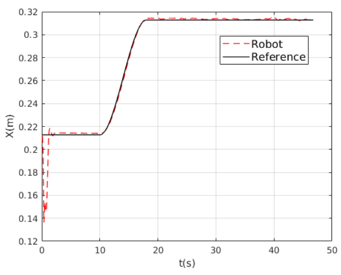
5.1.5 Impedance control in the task space with force sensing (simulator)

This architecture allows the interaction between the robot and the environment (figure 4.8). It can be used to interact with a human or any kind of material, like a mould. In this work it is used the JACO^2 robot in a polishing task as main goal. Firstly it is necessary to analyse the performance of the robot in free space, and later a polishing task is demonstrated, which represents the interaction between the robot and a flat surface.

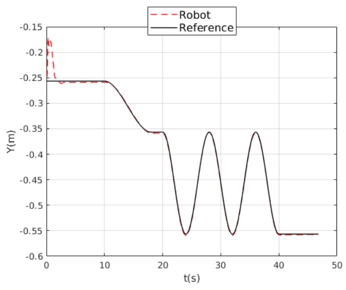
For a movement in free space, the defined trajectory is the same as the trajectory defined in the previous subsection 5.1.4. Besides position tracking it is also made the velocity tracking and the acceleration tracking during the circular trajectory. So, the robot starts to move from a point to another, (see equation 5.1), and then begins a circular trajectory, defined from the following expression

$$p_d = p_i + \begin{bmatrix} 0 \\ a \cdot cos(wt) - a \\ b \cdot sin(wt) \end{bmatrix} \; ; \quad \dot{p}_d = \dot{p}_i + \begin{bmatrix} 0 \\ -w \cdot a \cdot sin(wt) \\ w \cdot b \cdot cos(wt) \end{bmatrix} \; ; \quad a_d = \begin{bmatrix} 0 \\ -w^2 \cdot a \cdot cos(wt) \\ -w^2 \cdot b \cdot sin(wt) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.13}$$

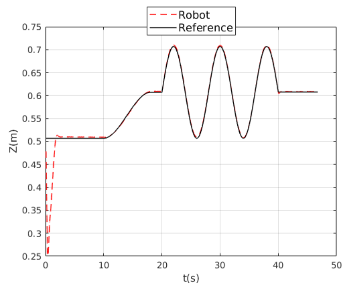
Figure (5.6) shows that the robot has good position and orientation tracking in free space. Analysing the plots referred to orientation you can see that the position motion has a small influence in the orientation motion.



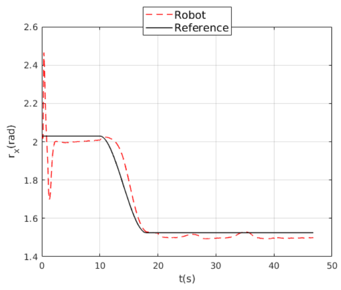
(a) X Position.



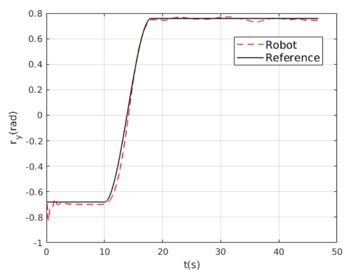
(b) Y Position.



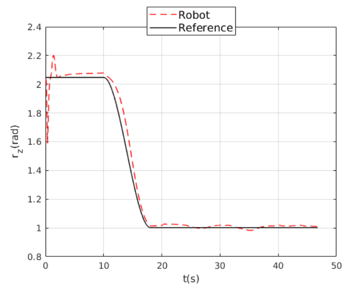
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

Figure 5.6: Position/Orientation impedance control with force sensing (Gazebo Simulator).

Table 5.5 represents the controller gains used to move the robot in free space.

Table 5.5: Impedance task space control gains, where the variable I in 5.5a and 5.5b represents the identity matrix.

(a) Position Gains.

Gain	Position
K_p	$150I_{3\times3}$
D_p	$30I_{3\times3}$
I_p	$5I_{3\times3}$

(b) Orientation Gains.

Gain	Position
K_o	$5I_{3\times3}$
D_o	$2I_{3\times3}$

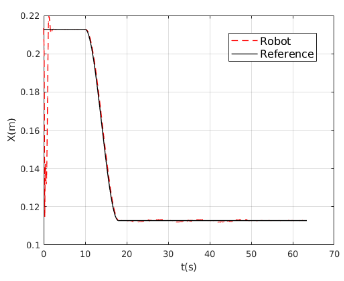
5.1.6 Impedance control in the task space with force sensing for redundant robots (simulator)

The last results to be analysed in the simulator refers to the control architecture present in figure 4.9. This algorithm is an alternative to the previously control, since it does not need to calculate the inverse of the Jacobian, and this control can be adapted or implemented in any type of manipulator. Besides this advantage it is also implemented the orientation with the quaternions, thus solving the problems of singularities associated to angle-axis representation.

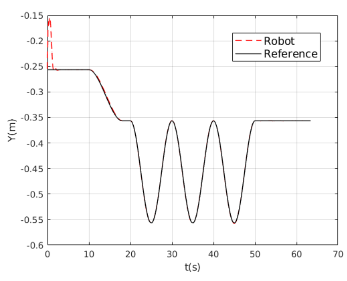
Figure 5.7 shows the good position tracking where the robot end-effector performs smooth movements in the task space. From $t = 10s$ to $t = 18s$ the position trajectory is defined according to equation 5.1, whose parameters are

$$\begin{cases} p_i = \begin{bmatrix} 0.2127 & -0.2566 & 0.5069 \end{bmatrix}^T \\ p_f = \begin{bmatrix} 0.1127 & -0.3566 & 0.6069 \end{bmatrix}^T \\ \qquad t_i = 10s \quad ; \quad t_f = 18s \end{cases} \tag{5.14}$$

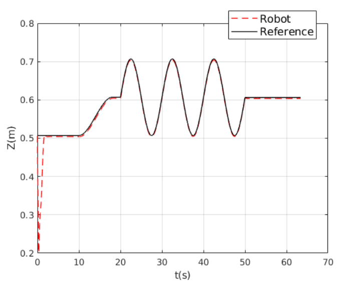
In relation to the circular path is described according to expression 5.13.



(a) X Position.



(b) Y Position.



(c) Z Position.

Figure 5.7: Position impedance control with force sensing for redundant robots (Gazebo Simulator).

It is important to refer, table 5.6 refers to the movement in free space, so in contact the gains can differ from axis to axis, in other words, the controller can be more compliant in one axis than in the others.

Table 5.6: Impedance task space control gains, where the variable I in 5.6a and 5.6b represents the identity matrix.

(a) Position Gains.

Gain	Position
K_p	$200I_{3\times3}$
D_p	$30I_{3\times3}$
I_p	$25I_{3\times3}$

(b) Orientation Gains.

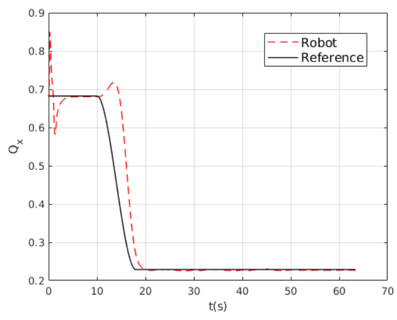
Gain	Position
K_o	$5I_{3\times3}$
D_o	$2I_{3\times3}$

In relation to the orientation control, the desired trajectory is a third degree polynomial, whose parameters are:

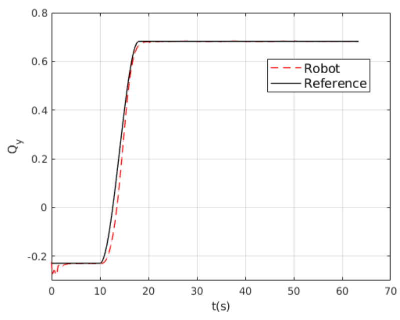
$$\begin{cases} Q_i = \begin{bmatrix} 0.6823 & -0.2288 & 0.6884 & 0.0896 \end{bmatrix}^T \\ Q_f = \begin{bmatrix} 0.2288 & 0.6823 & -0.0896 & 0.6884 \end{bmatrix}^T \\ \qquad t_i = 10s \quad ; \quad t_f = 18s \end{cases} \tag{5.15}$$

This orientation trajectory (eq. 5.15) corresponds to a rotation of $-\pi$ radians about the z axis of the robot end-effector.

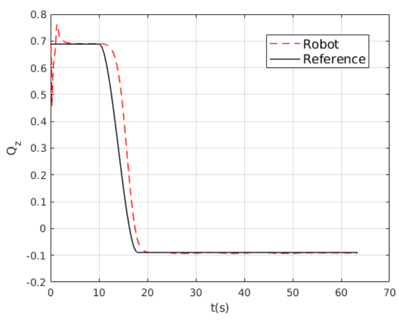
The results showed in figure 5.8 shows a good steady state error, a good orientation tracking and a smooth rotation.



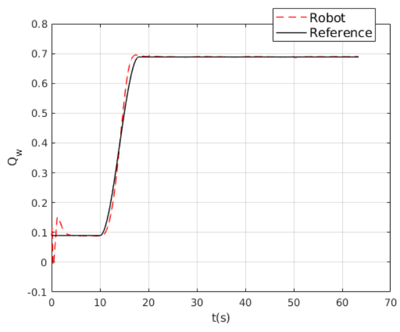
(a) ϵ_x Orientation.



(b) ϵ_y Orientation.



(c) ϵ_z Orientation.



(d) η Orientation.

Figure 5.8: Quaternion orientation impedance control with force sensing for redundant robots(Gazebo Simulator).

5.2 Real JACO² Results

The time has come to evaluate and compare the experimental results in the real robot with the results obtained in the Gazebo simulator. As in the simulator, the experimental tests are done with the robot performing a defined trajectory. But before discussing the results obtained it is necessary to change some internal parameters of the robot, in order to be able to integrate the controllers studied in chapter 4.

Internally in the robot are active different types of controllers. In order to minimize the influence of these internal factors it is necessary to use some functions available in the Kinova API. The functions used and the changed parameters are:

Table 5.7: Internal parameters that need to be changed.

Function	Description	Value
MySetTorqueSafetyFactor	If the velocity of an actuator gets to a specific threshold the robot stops and changes back to trajectory control. Feature that prevents the robot from taking high speed motion. Setting to 1 disables the feature.	1
MySetTorqueVibrationController	Vibration observer/controller to eliminate vibrations during contact with stiffness environments. Adjust from 0 to 1, to enable or disable vibrations.	1
MysetTorqueActuatorDamping	Set actuators damping gain.	0
MySetTorqueActuatorGain	Set the actuators feedback gain.	0

It is important to refer that after reboot the robot the parameters presented in the table 5.7 return to their default values. Moreover, before initiating the experimental tests, it is required to calibrate the torque sensors by setting them to zero, so it is necessary first place the robot in a configuration where gravity does not influence the joint torques. In the advanced specification guide [7] is suggested to use the following joint position

$$q_{calibration} = \begin{bmatrix} * & 180 & 180 & 0 & 0 & 180 \end{bmatrix} (degrees) \tag{5.16}$$

with '$*$' being any joint value. Similarly, the position of the actuators is also calibrated due to the common displacement occurred when the actuators are suddenly stopped. So as to calibrate the robot a node in the package "**thesis_demo**" denominated by "**calibration_robot.py**" is created. This node uses the function "**joint_position_client()**" to send the robot to calibrate position, $q_{calibration}$, and uses function "**ZeroTorque()**" to set the joint torques to zero. This functions are available in the "kinova_demo" package, (see section 2.3).

Last but not least, before sending the commanded torques, τ_c, switching to torque control from position control is required, this means that is necessary to use the service "**SetTorqueControlMode**" available in package "kinova_msgs". Then, to publish torque commands it is available the ros topic "**/j2n6s300_driver/in/joint_torque**" that use the "**JointTorque**" message, also available in package "kinova_msgs".

5.2.1 Computed torque control in the joint space

Initially it is used the same algorithm implemented in the Gazebo simulator, but the results are far from being the best, since to move the last three joints together high gains is required, which cause disturbances in the other joints leading to the instability of the system. This problem is due to the fact that the inertia matrix is underestimated in the last three joints, analytically this corresponds to a very low values in the last three values of the diagonal of the inertia matrix.

In order to solve this problem a solution has been proposed, use only the inertia matrix in the first three joints and in the other three only use a PID controller.

Before the sinusoidal wave trajectory it is needed first to put the robot joints in a more favourable position due to the robot geometry. It is suggested a third degree polynomial, (see eq. 5.1), with the following parameters

$$\begin{cases} q_i = \begin{bmatrix} 4.8046852 & 2.92482 & 1.002 & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T \quad (rad) \\ q_f = \begin{bmatrix} 4.8046852 & \pi & \pi/2 & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T \quad (rad) \\ t_i = 5s \quad ; \quad t_f = 15s \end{cases} \tag{5.17}$$

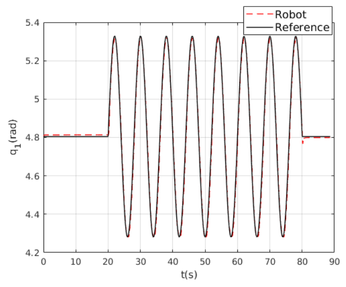
Then, from $t = 20s$ to $t = 80s$ begins the sinusoidal motion (see eq. 5.2) and the parameters are given by

$$\begin{cases} q_i = \begin{bmatrix} 4.8046852 & \pi & \pi/2 & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T \quad (rad) \\ A = \begin{bmatrix} \pi/6 & \pi/20 & \pi/20 & \pi/20 & \pi/20 & \pi/6 \end{bmatrix}^T \quad (rad) \quad ; \quad T = 8s \end{cases} \tag{5.18}$$

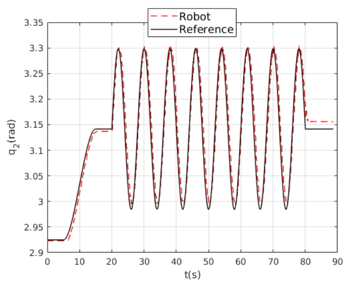
The achievements in the figure (5.9) concern to the control architecture of the figure 4.2, where is done not only the position tracking, but also the velocity and the acceleration tracking during the sinusoidal wave trajectory. So, the desired joint velocity $\dot{q}_d$ corresponds to the derivative of the desired joint positions q_d and the desired joint acceleration $\ddot{q}_d$ corresponds to the derivative of the joint velocities.

$$q_d = q_i + A \cdot sin(wt); \quad \dot{q}_d = w \cdot A \cdot cos(wt); \quad \ddot{q}_d = -w^2 \cdot A \cdot sin(wt) \tag{5.19}$$

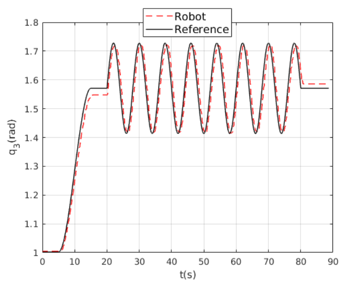
For this first solution the results presented in figure 5.9 with the PID gains detailed in table 5.8, have relatively good tracking response, performing stable movements during the



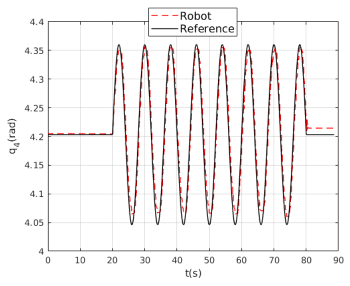
(a) Joint 1.



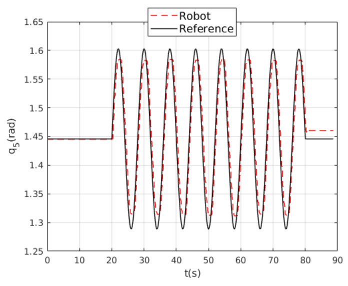
(b) Joint 2.



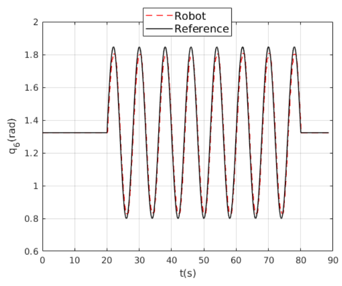
(c) Joint 3.



(d) Joint 4.



(e) Joint 5.



(f) Joint 5.

Figure 5.9: PID computed torque control in the joint space, with respect to the first solution.

defined trajectory. With this first approach the PID control gains in the last three joints are low comparatively to the gains of the simulated results (see tab. 5.1). This proves that is possible to achieve good results in the real JACO^2. Lastly, all the controller gains were manually tuned.

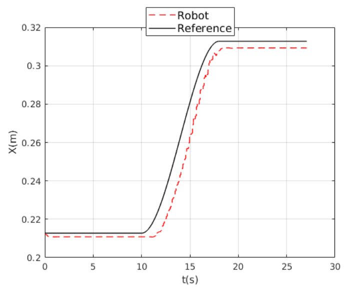
Table 5.8: Control Gains.

Joint j	K_{p_j}	K_{d_j}	K_{i_j}
1	200	20	0.1
2	180	34	0.3
3	200	34	0.3
4	22	6.4	0.3
5	24	4	0.1
6	20	2.6	0.1

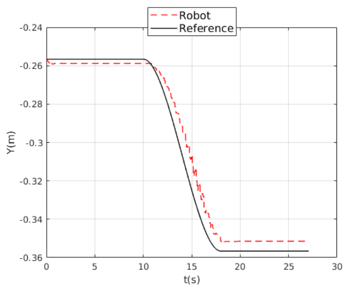
5.2.2 Computed torque control in the task space

Now, it is presented the practical results of the control scheme of the figure 4.3. In this case, the defined trajectory in task space is a third degree polynomial, (see eq. 5.1), where the trajectory parameters are the same of the simulated results in equation 5.7.

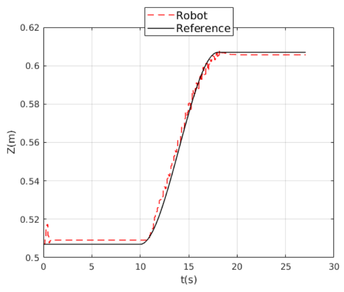
Observing the experimental results exposed in the figure 5.10, the robot is able to reach its final pose (position, orientation) with overall acceptable steady state error. Even though the dynamic model is underestimated the position and orientation tracking are acceptable and closely to the simulated results.



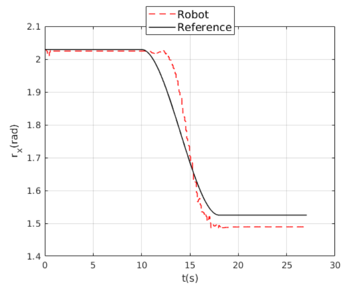
(a) X Position.



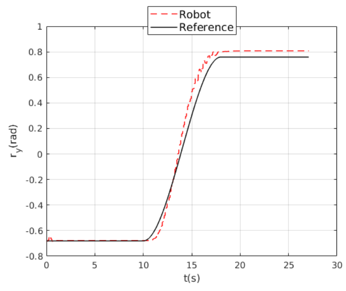
(b) Y Position.



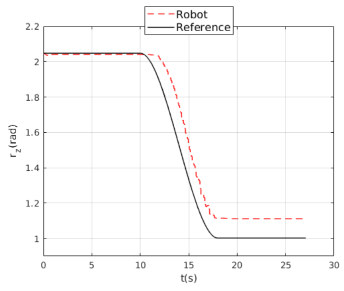
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

Figure 5.10: Position/Orientation PD computer torque control in the task space.

In relation to the controller gains, (tab. 5.8), they are similar to the simulated results. For obtain a better orientation tracking of this controller and on the following controllers, the inertia tensor with respect to "z" is artificially increased in the last joint. The multiplicative factor required to obtain good results is "200".

Table 5.9: Position and Orientation Gains.

Position	Gain	Orientation	Gain
$\boldsymbol{K_{p,p}}$	200	$\boldsymbol{K_{p,o}}$	100
$\boldsymbol{K_{d,p}}$	22	$\boldsymbol{K_{d,o}}$	4

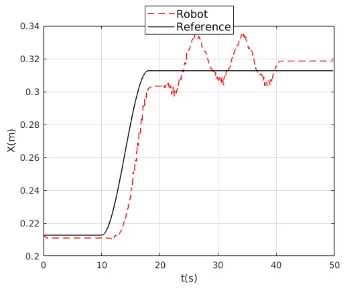
5.2.3 Impedance control in the task space without force sensing

The following experimental results are done in the same conditions of the simulated results, in other words, the same trajectory is defined, (see subsection 5.1.4) for the control scheme of the figure 4.7.

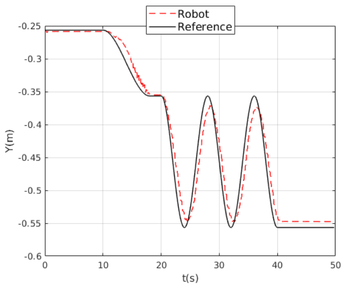
The results in figure 5.11, referring to the real robot, are reasonable considering that in the control scheme used the matrix associated to the robot dynamics, the inertia matrix M, is not considered. In addition, there is a great perturbation in the orientation of the end-effector, figures 5.11d, 5.11e and 5.11f, and also in the component "x" of the end-effector position, since a circular trajectory is defined in the components (y,z) of the end-effector position.

There is also some oscillation in the "z" component, figure 5.11c, making the movement of end-effector slightly smooth.

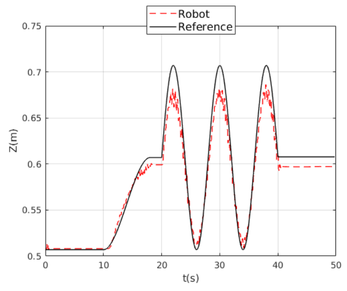
In the tests made in the simulated environment there is not such problems because the dynamic parameters, like damping, friction, stiffness, do not accurately represent the robot hardware.



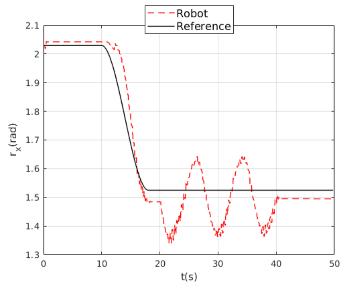
(a) X Position.



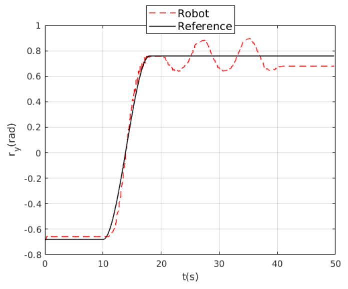
(b) Y Position.



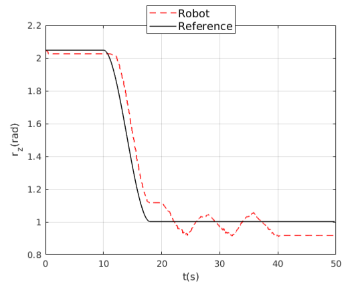
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

Figure 5.11: Position/Orientation impedance control without force sensing.

Table 5.10: Position and Orientation Gains.

Position	Gain	Orientation	Gain
$\boldsymbol{K_p}$	200	$\boldsymbol{K_o}$	10
$\boldsymbol{D_p}$	26	$\boldsymbol{D_o}$	1.2

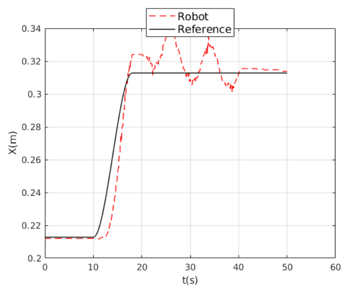
5.2.4 Impedance control in the task space with force sensing

Although the robot dynamics is considered, the results do not improve significantly in comparison to the controller aforementioned, figure 5.11.

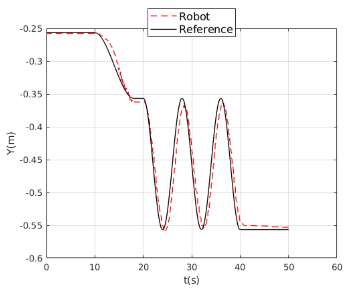
The tests are done for the same parameters as the algorithm presented in the simulator, (see subsection 5.1.5). These results are only made for movements in free space, even thought this controller allows external interactions.

Again it is possible to observe the noise that the circular path causes in the orientation of the end-effector, figures 5.12d, 5.12e and 5.12f. With this, it is verified that the friction of actuators becomes dominant causing noise and oscillations in the movement of the robot end-effector.

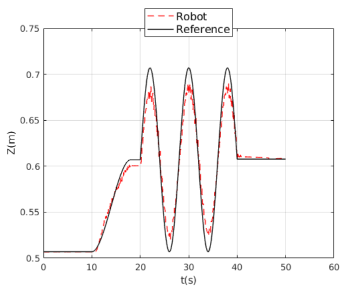
Furthermore, it is important to emphasize that the illustrated planned trajectory is not particularly simple, involving a complex movement, which naturally can cause even more friction.



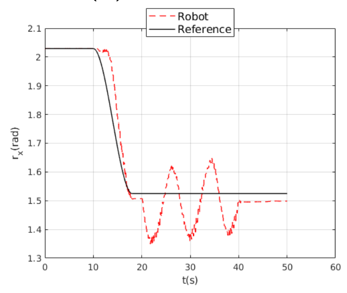
(a) X Position.



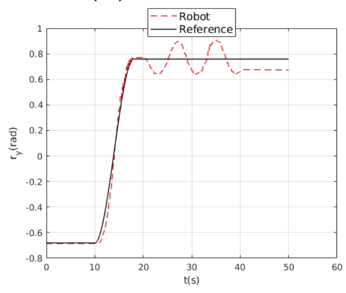
(b) Y Position.



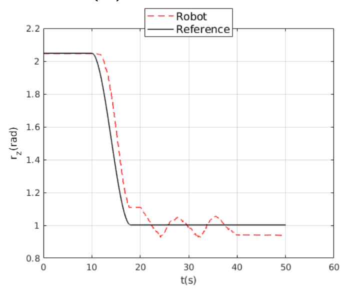
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

Figure 5.12: Position/Orientation impedance control with force sensing.

Table 5.11 represents the spring and damper terms with respect to end-effector position and orientation in free-space.

Table 5.11: Impedance task space control gains, where the variable I in 5.11a and 5.11b represents the identity matrix.

(a) Position Gains.

Gain	Position
K_p	$220I_{3\times3}$
D_p	$24I_{3\times3}$
I_p	$30I_{3\times3}$

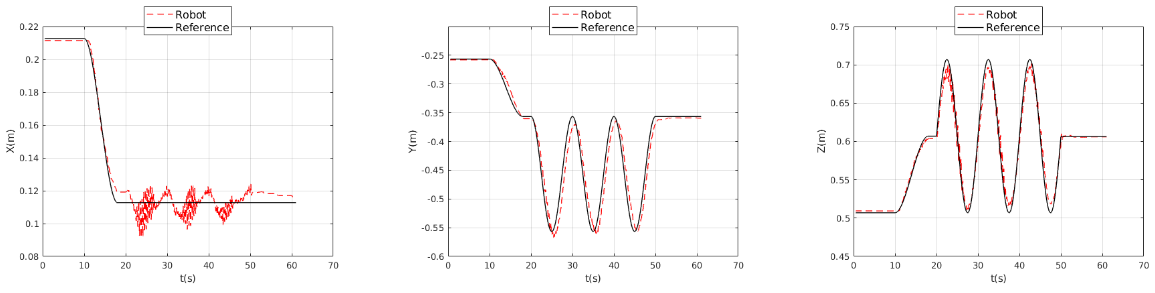
(b) Orientation Gains.

Gain	Orientation
K_o	$10I_{3\times3}$
D_o	$2I_{3\times3}$

5.2.5 Impedance control in the task space with force sensing for redundant robots

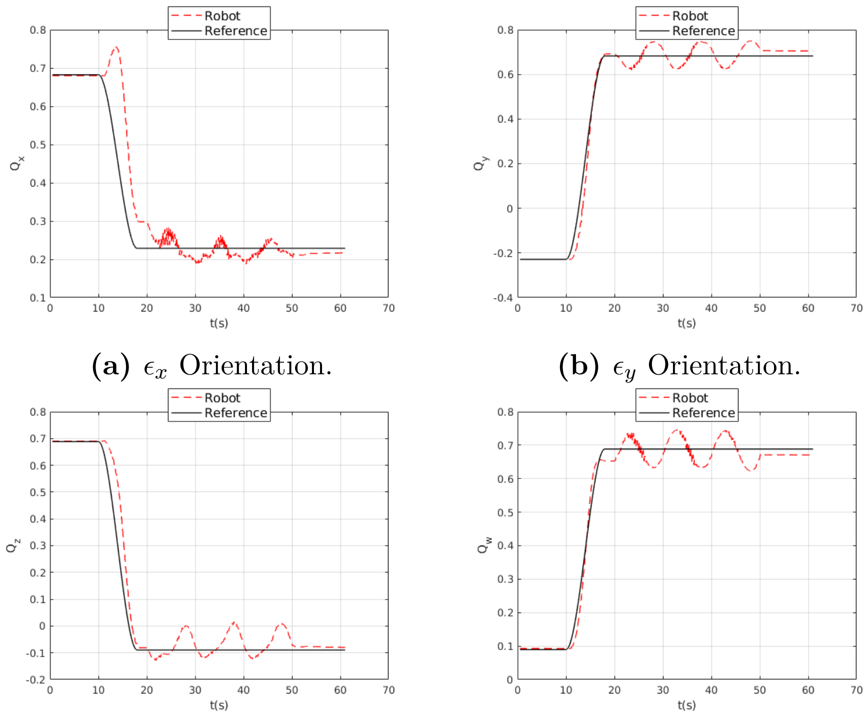
The last controller to be analysed corresponds to the architecture of the figure 4.9. Analysing the figure 5.13 referring to the end-effector position, it is possible to observe the significant noise in the "x" component that was verified in the architecture analysed previously.

In relation to the orientation of the end-effector, figure 5.14, there is a significant improvement over the previous results. This is because the chosen orientation is the quaternions, which helps the robot in the orientation tracking.



(a) X Position. **(b)** Y Position. **(c)** Z Position.

Figure 5.13: Position impedance control with force sensing for redundant robots.



(a) ϵ_x Orientation. **(b)** ϵ_y Orientation.

(c) ϵ_z Orientation. **(d)** η Orientation.

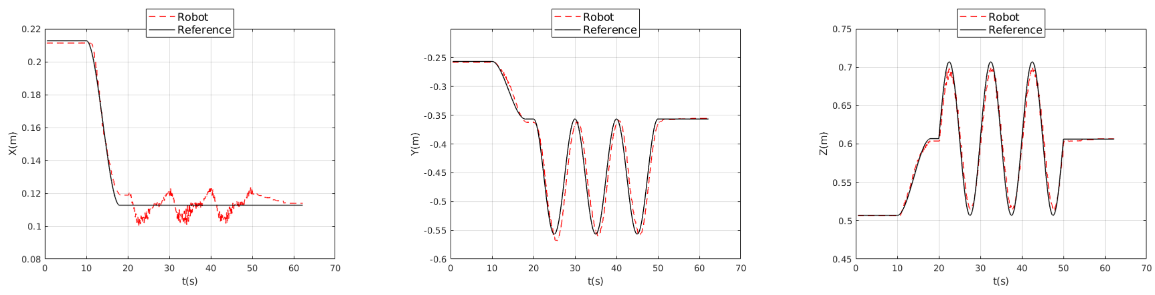
Figure 5.14: Quaternion orientation impedance control with force sensing for redundant robots.

In order to reduce the noise verified previously a low pass filter is applied to the joint position and also to the joint velocity.

$$q_{filtered}[k] = \frac{\alpha h q[k] + q_{filtered}[k-1]}{1+\alpha h}; \quad \dot{q}_{filtered}[k] = \frac{\alpha h \dot{q}[k] + \dot{q}_{filtered}[k-1]}{1+\alpha h} \tag{5.20}$$

where q is the joint angular position, $\dot{q}$ is the joint angular velocity, α is the cut-off frequency constant and h is the sample time. Therefore, the filtered signals do not solve the end-effector position problem, but helps to stabilize the controller.

Comparing the figures 5.13 and 5.15 referring to the end-effector position, there is an improvement comparing to the results analysed previously. Although the improvement, the robot continues having difficulties in following the reference during the circular trajectory, and the perturbation in the "x" component continues to be significant. However, the influence of the low pass filter makes the movement slightly softer. It is important to refer that these results are made for same conditions of the simulated experiments, see subsection 5.1.6.



(a) X Position. **(b)** Y Position. **(c)** Z Position.

Figure 5.15: Position impedance control with force sensing for redundant robots using a low pass filter with a cut-off frequency of 100 Hz.

Table 5.12: Impedance task space control gains, where the variable I in 5.12a and 5.12b represents the identity matrix.

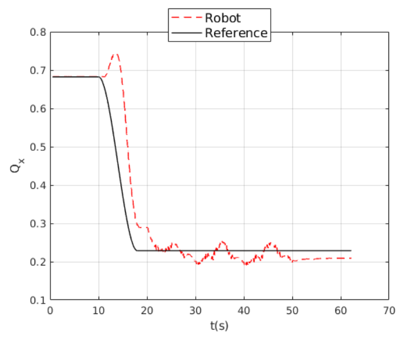
(a) Position Gains.

Gain	Position
K_p	$244I_{3\times3}$
D_p	$24I_{3\times3}$
I_p	$30I_{3\times3}$

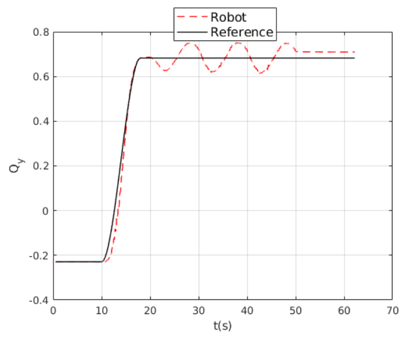
(b) Orientation Gains.

Gain	Orientation
K_o	$13I_{3\times3}$
D_o	$2I_{3\times3}$

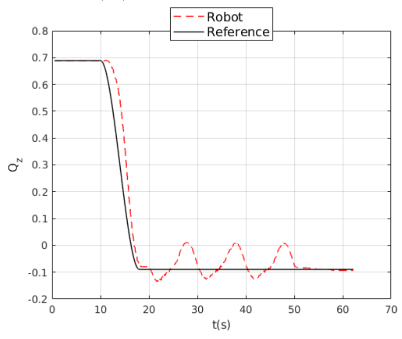
Regarding the end-effector orientation, figure 5.16, there is also perceptible that the low pass filter helps to decrease the perturbation of the end-effector position in the orientation.



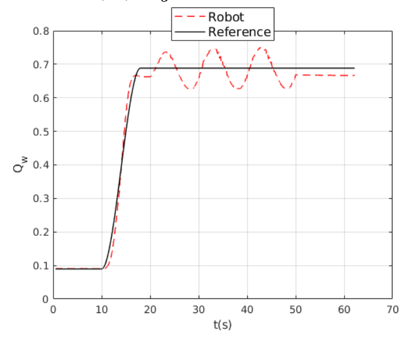
(a) ϵ_x Orientation.



(b) ϵ_y Orientation.



(c) ϵ_z Orientation.



(d) η Orientation.

Figure 5.16: Quaternion orientation impedance control with force sensing for redundant robots using a low pass filter with a cut-off frequency of 100 Hz.

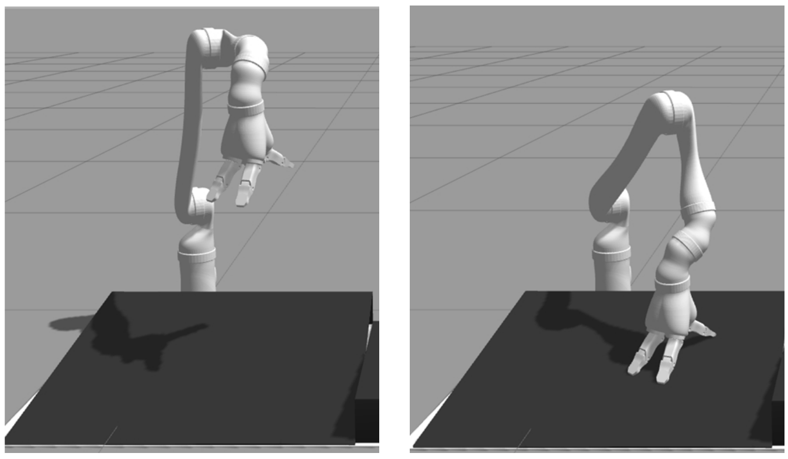
In the next chapter, the impedance controller presented in figure 4.9 is tested in a polishing task in a simulated environment and the in the real JACO^2 robot.

6 Surface Polishing

6.1 Polishing task in Gazebo simulator

As previously stated, the impedance control allows interactions between the robot and the surrounding environment. Therefore, to demonstrate the concept it is used a ramp shape surface, this object is obtained from the Gazebo database. At the beginning of the section 4.5 is shown three different situations that occur between the robot and the environment. The first is before contact, the second is when the contact occurs and the third is when the robot is in contact with the ramp.

Figure 6.1 shows the robot before contact and when the robot comes into contact with the ramp.



(a) Without any contact between robot and environment. **(b)** Critical point when contact occurs.

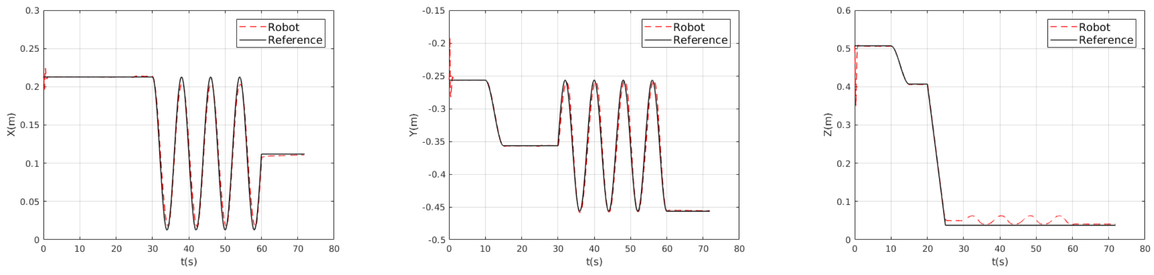
Figure 6.1: Polishing task demonstration in simulated environment.

In simulation a specific trajectory is defined. Firstly, a $-\frac{\pi}{2}$ radians rotation of the end-effector in "y" axis and a movement of the end-effector position is applied, the trajectory is based on a third degree polynomial. Secondly, a linear movement of the end-effector position in z axis is made until the robot comes into contact with the ramp. Finally, a circular movement is effectuated in contact with the ramp. It is important to mention that the control architecture used in this demonstration is the controller of the figure 4.9.

From $t = 10s$ to $t = 15$ the position trajectory, figure 6.2, is defined according to equation 5.1, then from $t = 20s$ to $t = 25s$ according to equation 5.9 and finally from $t = 30s$ to $t = 60s$ the circular trajectory is defined according to the following equation

$$p_d = p_i + \begin{bmatrix} a \cdot cos(wt) - a \\ b \cdot sin(wt) \\ 0 \end{bmatrix} ; \quad \dot{p}_d = \dot{p}_i + \begin{bmatrix} -w \cdot a \cdot sin(wt) \\ w \cdot b \cdot cos(wt) \\ 0 \end{bmatrix} ; \quad a_d = \begin{bmatrix} -w^2 \cdot a \cdot cos(wt) \\ -w^2 \cdot b \cdot sin(wt) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{6.1}$$

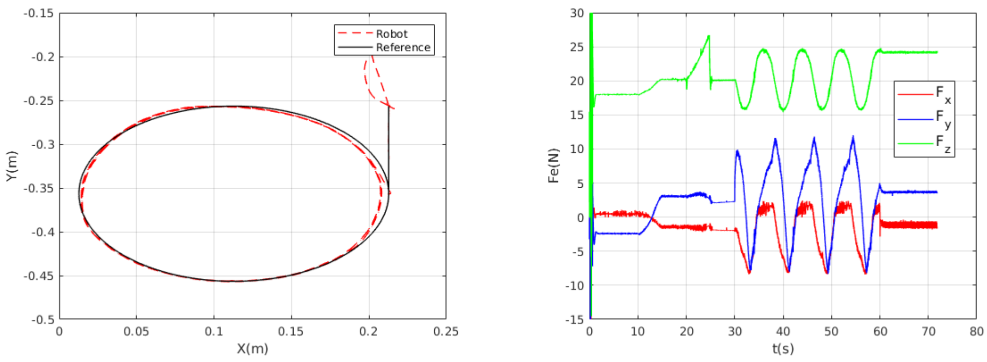
where the radius is $a = b = 0.1m$ and the angular frequency is $w = \dfrac{2\pi}{T}$ with the period $T = 8s$.



(a) X Position. **(b)** Y Position. **(c)** Z Position.

Figure 6.2: End-effector position during the polishing task (Gazebo Simulator).

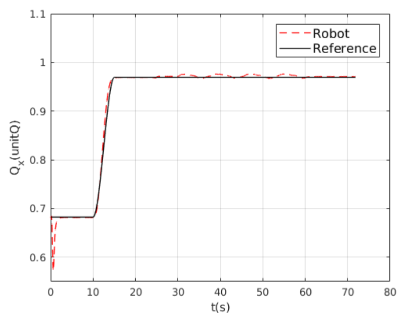
Figure 6.3 shows a different perspective of the circular trajectory and the measured end-effector force in contact with the ramp.



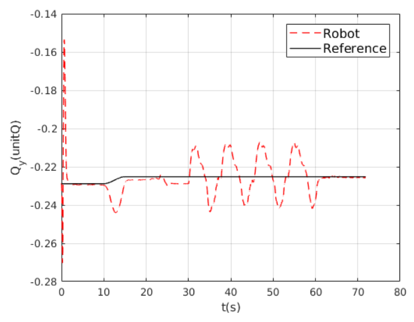
(a) Circular trajectory in contact. **(b)** End-Effector force F_e.

Figure 6.3: The circular defined trajectory and end-effector force measured (Gazebo Simulator).

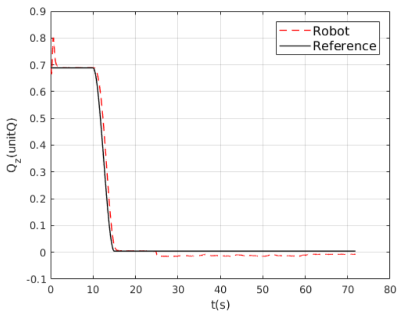
The orientation trajectory is defined according to the equation 5.1. In relation to the results of the figures 6.2, 6.3 and 6.4, it is verified a good position tracking without contact and with contact. In order to ensure the contact between the end-effector and the ramp, the reference of the robot in the component "z" is placed in a point that the robot can not reach. Analysing the figures 6.4b and 6.4d it is verified a perturbation associated to the circular trajectory and to the contact between the end-effector and the surface. Also, the robot fingers can cause friction between the robot and the surface, since they are not ready to polishing tasks.



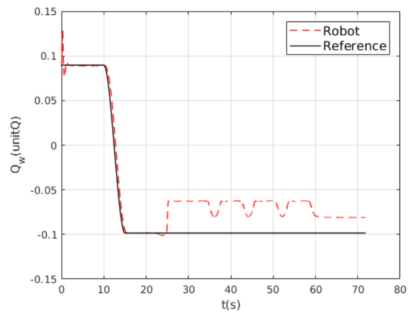
(a) ϵ_x Orientation.



(b) ϵ_y Orientation.



(c) ϵ_z Orientation.



(d) η Orientation.

Figure 6.4: End-effector orientation during the polishing task (Gazebo Simulator).

Table 6.1 represents the impedance control gains in free space and in contact with the surface.

Nevertheless, the polishing task in the simulator is successfully completed and validating the proposed impedance controller.

6.2 Polishing task in the real JACO²

The last step of this work corresponds to a demonstration of polishing in the real robot. The robot end-effector consists of three fingers, which is very useful for pick and place tasks, but is not adequate for the polishing task. Thus, for the polishing task accomplishment a

Table 6.1: Impedance task space control gains in Gazebo Simulator.

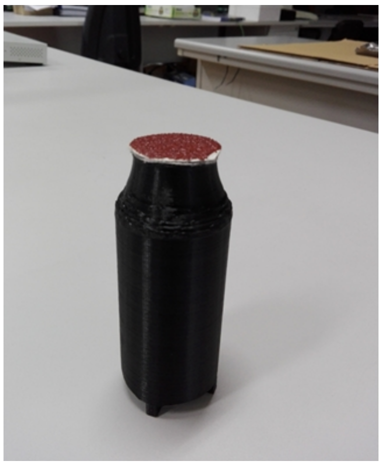
(a) Position Gains.

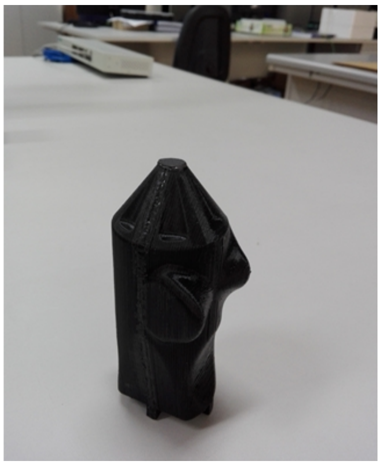
Gain	Position
K_p	$\begin{bmatrix} 400 & 0 & 0 \\ 0 & 400 & 0 \\ 0 & 0 & 500 \end{bmatrix}$
D_p	$\begin{bmatrix} 35 & 0 & 0 \\ 0 & 35 & 0 \\ 0 & 0 & 35 \end{bmatrix}$
I_p	$\begin{bmatrix} 30 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

(b) Orientation Gains.

Gain	Orientation
K_o	$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$
D_o	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

new tool is required. The tool is designed in the Fusion 360 and printed on a 3D printer, see figure 6.5b.


(a) Initial version.


(b) Final version.

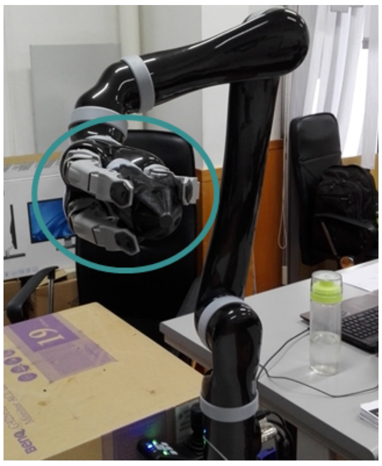

(c) Polishing tool.

Figure 6.5: Polishing tool with a sandpaper embedded in the robot end-effector.

To understand how the polishing task is accomplished, a state machine is constructed, figure 6.6. This process is divided as follows:

- **(0)**: The polishing node starts and the robot switch from position mode to torque mode;
- **(1)**: Then, a menu is displayed in the terminal, where the user can choose to start the polishing task by pressing the <enter> key or exit the program by pressing the <e> key;
- **(2)**: After the <enter> key is pressed the following message is displayed to the user: "Please choose the end point, after that select the number (1) ". With this, the user can pick up the end-effector and put it on the line endpoint. If the user does not

press the <1> key, the following message is displayed: "Please select the number (1) to continue ...";

- **(3)**: As soon as user selects the <1> key, the following message appears: "Please choose the initial point, after that select the number (2) ". Again the user can place the end-effector in the desired position. If after the procedure the user does not click the <2> key, the following message is displayed: "Please select the number (2) to continue ...";

- **(4)**: Then it is suggested to the user to press the <3> key to start the polishing;

- **(5)**: Once the <3> key is selected, the control loop is started, where the robot moves from the star point to the end point and when the end-effector reaches the end point performs a circular trajectory in contact with the flat surface. This control loop is active for a pre-set time, but can be interrupted by pressing the <enter> key. If the pre-set time has finished or the <enter> key is selected, the initial menu is shown again and so on.

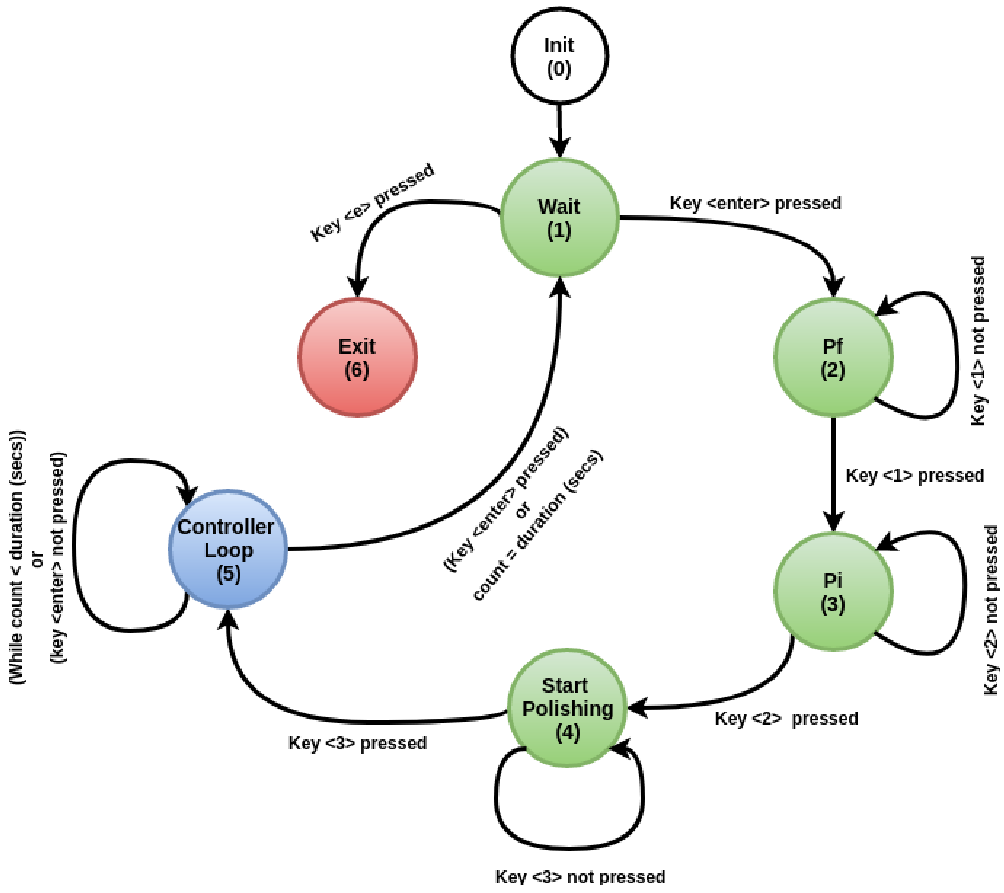
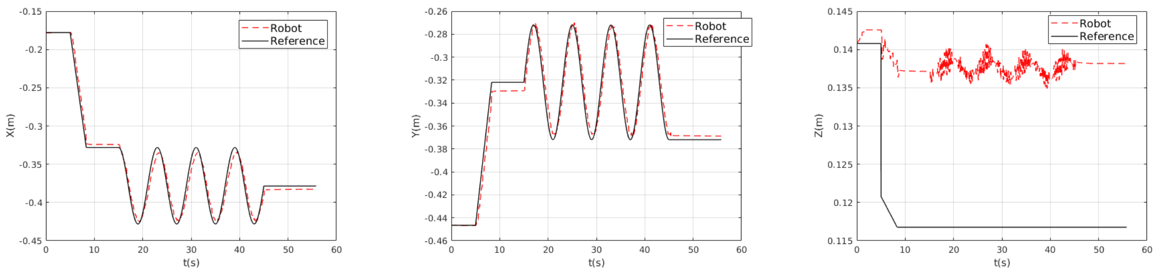


Figure 6.6: State-machine for the polishing task.

With this, it is time to analyse the experimental results, where the control loop of the figure 6.6 corresponds to the control scheme of the figure 4.9.

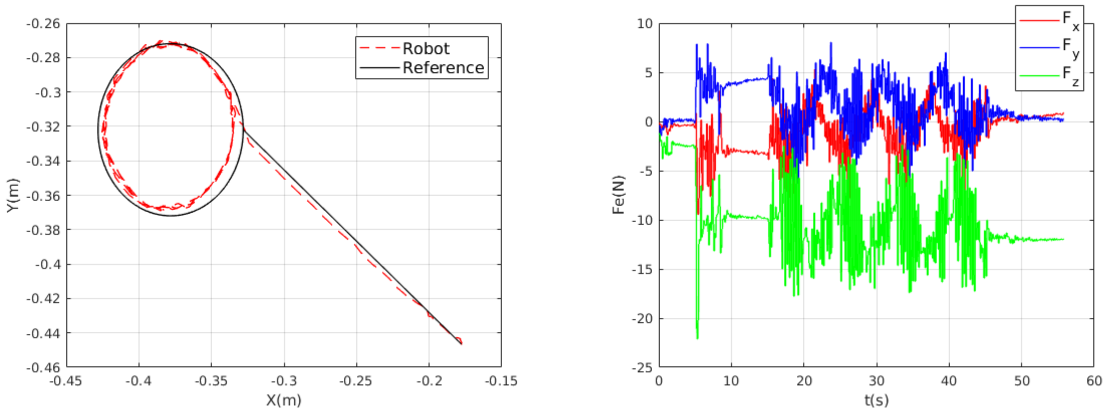
Looking at figure 6.7, it can be concluded that the position tracking is reasonable, but analysing the figure 6.7c is verified a lot of noise in the position component z. This noise causes a not smooth movement of the end-effector. The defined trajectory from $t = 5s$ to $t = 8s$ is according to equation 5.9, and then from $t = 15s$ to $t = 45s$ is defined according to equation 6.1, where the radius is $a = b = 0.05m$ and the angular frequency is $w = \frac{2\pi}{T}$ with the period being $T = 8s$. It is important to mention that before the movement from the initial point to the final point the position reference in "z" is placed two centimetres under surface to ensure the contact.



(a) X Position. **(b)** Y Position. **(c)** Z Position.

Figure 6.7: End-effector position during the polishing task.

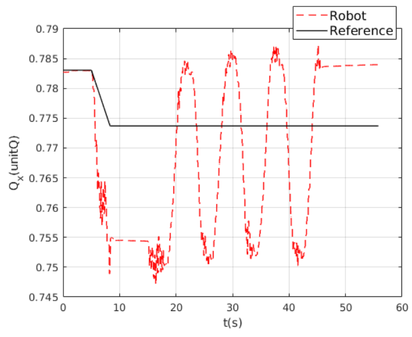
A different perspective of the circular trajectory and the measured end-effector force in contact with the ramp are shown in figure 6.8.



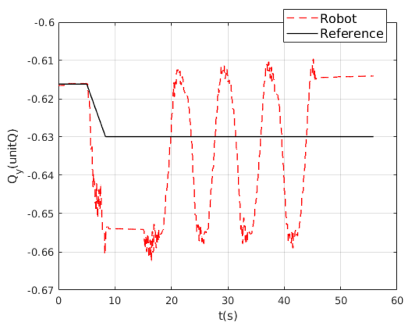
(a) Circular trajectory in contact. **(b)** End-Effector force F_e.

Figure 6.8: The circular defined trajectory and end-effector force measured.

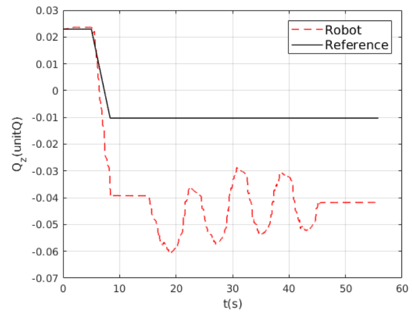
Now, analysing in detail the end-effector orientation during the contact between the polishing tool and the surface plane, figure 6.9, it can be seen a lot of noise in the end-effector orientation caused by the polishing movement. As result, the orientation tracking need improvements and again the end-effector movement is slightly smooth.



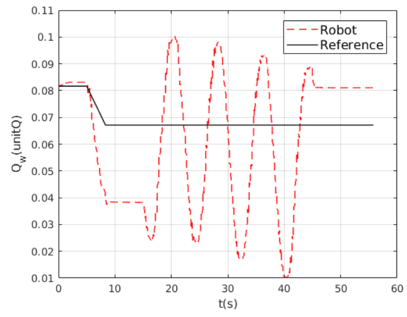
(a) ϵ_x Orientation.



(b) ϵ_y Orientation.



(c) ϵ_z Orientation.



(d) η Orientation.

Figure 6.9: End-effector orientation during the polishing task.

Table 6.2 represents the impedance control gains in contact with the flat surface. The integral gain in "z" is considered zero in order to avoid that the end-effector is forced to assume the reference in "z".

Table 6.2: Impedance task space control gains obtained in the real JACO^2.

(a) Position Gains.

Gain	Position
K_p	$\begin{bmatrix} 400 & 0 & 0 \\ 0 & 400 & 0 \\ 0 & 0 & 500 \end{bmatrix}$
D_p	$\begin{bmatrix} 24 & 0 & 0 \\ 0 & 24 & 0 \\ 0 & 0 & 24 \end{bmatrix}$
I_p	$\begin{bmatrix} 25 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

(b) Orientation Gains.

Gain	Orientation
K_o	$\begin{bmatrix} 13 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 13 \end{bmatrix}$
D_o	$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

Appendix F shows in detail the flat surface used to test the robot performance during the polishing task.

7 Conclusion

Despite the difficulties that appear during this work, the main objectives are achieved and the concluding remarks can be summarized in the following topics:

- The results obtained in Gazebo simulator are quite promising, allowing to prepare and test different control algorithms before being tested in the real robot. Because of this it is possible to establish a secure connection with the real robot;
- Although the results achieved in the real robot need improvements, due to the friction of the actuators and other external factors, this work demonstrates that it is possible to use the JACO^2 in polishing tasks and with an in-depth study of the impedance controller. Anyway, this work can be a big step in automated polishing tasks;
- It is also concluded that it is necessary to analyse and rectify the URDF model built for the robot in ROS, because the robot dynamics corresponds to an approximation to cylinders and this produce an underestimated dynamic model;
- Even though the internal controllers present in JACO^2 proved to be an obstacle, some effort has been made to overcome this problem, including turn off some internal parameters;
- Finally, this work opens doors for future researchers who want to work in the area of collaborative robotics and human-machine interaction.

7.1 Future Work

Many adaptations, tests and experiments have been left for the future. For this reason, here are some suggestions:

- Improve the impedance controller, reducing the friction and noise produced by the actuators, achieving more precise and smooth movements;

- Attach to the JACO^2 robot an end-effector suitable for polishing and test the impedance controller on different types of surfaces;
- Integrate external sensors, such as cameras to coordinate and supervise the robot movements during polishing tasks.

8 Bibliography

[1] Kinova Robotics. "http://www.kinovarobotics.com".

[2] ROS community. Higher-Level concepts of *ROS*. "http://wiki.ros.org/ROS/Higher-Level%20Concepts".

[3] Jinjun Duan, Yahui Gan, Ming Chen, and Xianzhong Dai. Adaptive variable impedance control for dynamic contact force tracking in uncertain environment. *Robotics and Autonomous Systems*, 102:54–65, 2018.

[4] Neville Hogan. Impedance control: An approach to manipulation: Part ii—implementation. *Journal of dynamic systems, measurement, and control*, 107(1):8–16, 1985.

[5] Kinova Robotics. Development center user guide. "http://www.kinovarobotics.com/wp-content/uploads/2017/10/Kinova-SDK-Development-Center-User-Guide.pdf".

[6] Kinova Robotics. Gazebo for Kinova robots. "https://github.com/Kinovarobotics/kinova-ros/wiki/Gazebo".

[7] Kinova Robotics. KINOVA Ultra lightweight robotic arm user guide. "https://beta.kinovarobotics.com/sites/default/files/KINOVA_JACO_Prosthetic_robotic_arm_USER_GUIDE_0.pdf".

[8] Kinova Robotics. MoveIt for Kinova robots. "https://github.com/Kinovarobotics/kinova-ros/wiki/MoveIt".

[9] Kinova Robotics. Official ROS packages for kinova robotic arms. "https://github.com/Kinovarobotics/kinova-ros".

[10] Lentin Joseph. *Mastering ROS for Robotics Programming.* Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK, first edition edition, December 2015.

[11] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Dynamics and Control*. Second edition edition, January 2004.

[12] Miguel Pereira Mendes. Computed torque-control of the Kinova JACO^2 arm. Master thesis, University of Coimbra, 2017.

[13] Abd El Khalick Mohammad, Jie Hong, and Danwei Wang. Design of a force-controlled end-effector with low-inertia effect for robotic polishing using macro-mini robot approach. *Robotics and Computer-Integrated Manufacturing*, 49:54–65, 2018.

[14] Morgan Quigley, Brian Gerkey and William D. Smart. *Programming Robots with ROS*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, first edition edition, December 2015.

[15] ROS community. Gazebo ROS plugin. "`http://wiki.ros.org/gazebo`".

[16] ROS community. Introduction to ROS. "`http://wiki.ros.org/ROS/Introduction`".

[17] ROS community. Main concepts of ROS. "`http://wiki.ros.org/ROS/Concepts`".

[18] ROS community. ROS Control. "`http://wiki.ros.org/ros_control`".

[19] ROS community. ROS Control and gazebo. "`http://wiki.ros.org/gazebo`".

[20] ROS community. ROS MoveIt. "`http://moveit.ros.org/`".

[21] ROS community. ROS Rviz. "`http://wiki.ros.org/rviz`".

[22] ROS community. ROS URDF. "`http://wiki.ros.org/urdf`".

[23] Rui Cortesão. *Medical Robotics*. University of Coimbra, 2016/17.

[24] Luis Santos and Rui Cortesao. Joint space torque control with task space posture reference for robotic-assisted tele-echography. In *RO-MAN, 2012 IEEE*, pages 126–131. IEEE, 2012.

[25] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Kinematics*. Springer, 2009.

[26] Cristóvão Sousa, Rui Cortesão, and Luís Santos. Computed torque posture control for robotic-assisted tele-echography. In *Control & Automation (MED), 2010 18th Mediterranean Conference on*, pages 1561–1566. IEEE, 2010.

[27] The Orocos Project. "`http://www.orocos.org/`".

[28] Fengjie Tian, Chong Lv, Zhenguo Li, and Guangbao Liu. Modeling and control of robotic automatic polishing for curved surfaces. *CIRP Journal of Manufacturing Science and Technology*, 14:55–64, 2016.

Appendix A

JACO² Product Specification

Specifications

JACO^2

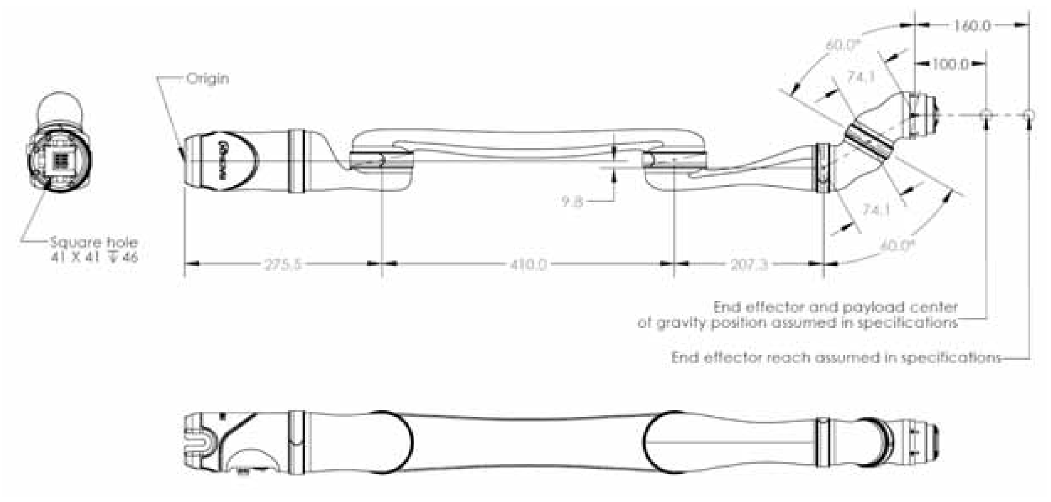
6 DOF

Tech Specs

JACO²

6 DOF

Version 1.1 - April 2017


Origin
Square hole
41 X 41 ⊽46
160.0
100.0
60.0°
74.1
74.1
60.0°
9.8
275.5
410.0
207.3
End effector and payload center
of gravity position assumed in specifications
End effector reach assumed in specifications

GENERAL

		NO GRIPPER	2 FINGERS (KG-2)	3 FINGERS (KG-3)
Total weight		4.4 kg	5.0 kg	5.2 kg
Payload capabilities	Mid-range continuous	2.6 kg	1.8 kg	1.6 kg
	Full-reach peak/temporary	2.2 kg	1.5 kg	1.3 kg

Materials	Links	Carbon fiber
	Actuators	Aluminum
Maximum reach		90 cm
Joint range after start-up *(sotware limitation)*		±27.7 turns
Maximum linear arm speed		20 cm/s
Power supply voltage		18 to 29 VDC, 24 VDC nominal
Peak power		100 W
Average power	Operating mode	25 W
	Standby mode	5 W
Communication protocol		RS-485
Communication cables		20 pins flat flex cable
Expansion pins		2 *(on communication bus)*
Water resistance		IPX2
Operating temperature		-10 °C to 40 °C

CONTROLLER

Ports	Joystick	1 Mbps Canbus
	Power supply	18 to 29 VDC, 24 VDC nominal
	USB 2.0 (API)	12 Mbps
	Ethernet (API)	100 Mbps
Control system frequency	High level (API)	100 Hz
	Low level (API)	500 Hz
CPU		360 MHz
SDK	APIs	High and low level
	Compatibility	Windows, Linux Ubuntu & ROS
	Port	USB 2.0, Ethernet
	Programming languages	C++
Control		Force, cartesian & angular

SPECIFICATIONS

Actuators #1, #2 & #3	K-75+
Actuators #4, #5 & #6	K-58

Appendix B

Actuators Product Specification

Specifications

ACTUATORS

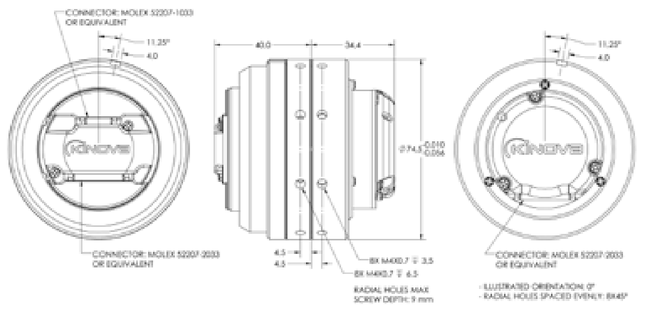
K-75+ K-58

Tech Specs

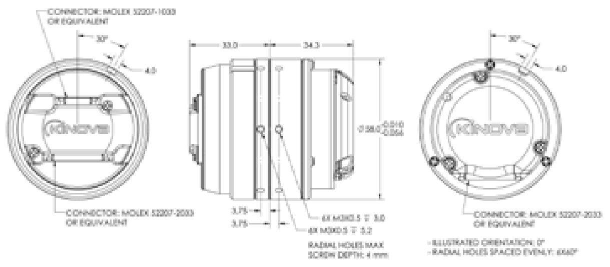
ACTUATORS

Version 1.1 – May 2017

K-75+ Ø74.5 mm, 12.0 Nm nominal, 37 Nm peak
Brushless DC motor, ratio 160 Harmonic Drive™


CONNECTOR: MOLEX 52207-1033 OR EQUIVALENT
11.25°
4.0
40.0
34.4
Ø74.5 0.010 0.056
CONNECTOR: MOLEX 52207-2033 OR EQUIVALENT
4.5
4.5
8X M4X0.7 ⊽ 3.5
8X M4X0.7 ⊽ 6.5
RADIAL HOLES MAX SCREW DEPTH: 9 mm
11.25°
4.0
CONNECTOR: MOLEX 52207-2033 OR EQUIVALENT
- ILLUSTRATED ORIENTATION: 0°
- RADIAL HOLES SPACED EVENLY: 8X45°

K-58 Ø58 mm, 3.6 Nm nominal, 7.7 Nm peak
Brushless DC motor, ratio 110 Harmonic Drive™


CONNECTOR: MOLEX 52207-1033 OR EQUIVALENT
30°
4.0
33.0
34.3
Ø58.0 0.010 0.056
CONNECTOR: MOLEX 52207-2033 OR EQUIVALENT
3.75
3.75
6X M3X0.5 ⊽ 3.0
6X M3X0.5 ⊽ 5.2
RADIAL HOLES MAX SCREW DEPTH: 4 mm
30°
4.0
CONNECTOR: MOLEX 52207-2033 OR EQUIVALENT
- ILLUSTRATED ORIENTATION: 0°
- RADIAL HOLES SPACED EVENLY: 6X60°

GEARED MOTOR (WITH 24V SUPPLY)

	K-75+	K-58
No load speed	12.2 rpm	20.3 rpm
Nominal torque	12.0 Nm	3.6 Nm
Nominal speed	9.4 rpm	15.0 rpm
Peak torque *(software limitation)*	30.5 Nm	6.8 Nm
Max motor efficiency	83%	81%
Max gearing efficiency	76%	69%
Torque gradient	13.8 Nm/A	7.8 Nm/A
Backdriving torque	0.8 to 7 Nm	1.7 to 5.2 Nm

SENSORS

	K-75+	K-58
Position sensor resolution	3,686,400/turn	2,534,400/turn
Motion before position indexation	±2.25°	±3.27°

Absolute position sensor precision at start-up *(before indexation)*	±1.5°
Torque sensor precision *(room temperature)*	±0.4 Nm
Torque sensor temperature drift *(-10 °C to 40 °C)*	±0.3 Nm
Torque sensor cross-axis torque sensitivity	0% to 8%
Accelerometers range and bandwidth *(x, y and z)*	±3g, 50 Hz
Motor current sensor range and bandwidth	±5 A, 140 Hz
Temperature sensor range and precision	-40 °C to 125 °C, ±2 °C

MECHANICAL

	K-75+	K-58
Weight	570 g	357 g
Motion range after start-up *(software limitation)*	±27.7 turns	±27.7 turns
Max axial, radial and flexion moment loads *(static)*	7.6 kN, 3.0 kN, 87 Nm	4.7 kN, 1.8 kN, 39 Nm
Dynamic axial, radial and flexion moment loads ratings of the main bearing	3.5 kN, 1.5 kN, 41 Nm	2.1 kN, 0.8 kN, 17 Nm

THERMAL

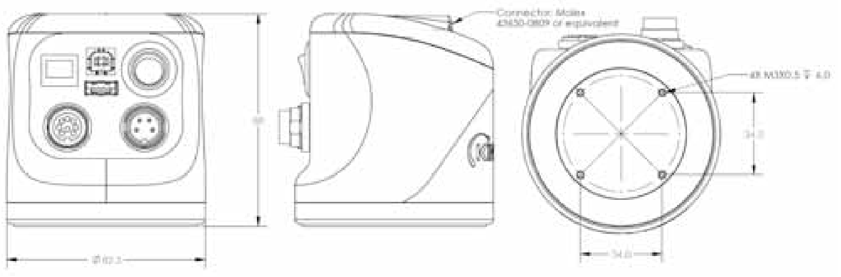
Operating temperature range	-10 °C to 40 °C
Max frame temperature *(overheat protection triggered)*	75 °C

	K-75+	K-58
Thermal time constant of the winding	22 s	16 s
Thermal time constant of the frame	39 min.	35 min.

ELECTRONIC

Power supply voltage	18 to 29 VDC, 24 VDC nominal
Communication protocol	RS-485
Communication cables	20 pins flat flex cable
Expansion pins	2 *(on communication bus)*

CONTROLLER



Ports	Joystick	1 Mbps Canbus
	Power supply	18 to 29 VDC
	USB 2.0 (API)	12 Mbps
	Ethernet (API)	100 Mbps
Control system frequency	High level (API)	100 Hz
	Low level (API)	500 Hz
CPU		360 MHz
SDK	APIs	High and low level
	Compatibility	Windows, Linux Ubuntu & ROS
	Port	USB 2.0, Ethernet
	Programming languages	C++
Control		Force, cartesian & angular

REFERENCE

A

Absolute position sensor precision at start-up (before indexation):
The absolute position measurement precision at power-up, before an index is detected (see Motion before indexation below).

Accelerometers range and bandwidth (x, y and z):
The range and bandwidth of the tri-axis accelerometer with signal conditioning.

B

Backdriving torque:
The load torque that causes an unpowered unit to backdrive. This value varies depending on of factors that include temperature and wear.

C

Communication cables:
The cables used to link each actuator in a daisy chain.

Communication protocol:
The communication protocol used between the actuators and controller.

D

Dynamic axial, radial and flexion moment loads ratings of the main bearing:
The actuator main bearing dynamic loads capacity.

E

Expansion pins (on communication bus):
The pins that are available to transmit signals through all the actuators to the controller with the output on the joystick port. 24V and ground pins are also available.

M

Max axial, radial and flexion moment loads (static):
The actuator main bearing static loads capacity.

Max frame temperature (overheat protection triggered):
The temperature measured at the frame at which a progressive current limitation starts to be applied by software. Torque loads above nominal should always be brief; this protection cannot guarantee the integrity of the motor under loads significantly higher than the nominal.

Max gearing efficiency:
An indicator of the gearing performance at input speed 500 rpm and temperature 30 °C. The efficiency of the gearing depends on factors including speed, load and temperature.

Max motor efficiency:
An indicator of the motor performance at its ideal operation torque and velocity. The efficiency of the motor depends on factors including friction and Joule power losses.

Motion before position indexation:
The max required output motion (after power-up) before an index is detected. When this precision index is detected, the position information is updated to the precise value.

Motion range after start-up (software limitation):
The motion range (software limitation).

Motor current sensor range and bandwidth:
The motor current measurement range and bandwidth.

N

No load speed:
The maximum speed (no payload, 24 VDC power supply).

Nominal speed:
The maximum speed under Nominal torque load.

Nominal torque:
The continuous torque output that causes the actuator frame to heat up to Max frame temperature (tested at 23 °C with the actuator enclosed in a plastic shell). Loadings above this value should always be brief.

O

Operating temperature range:
Actuator safe operating temperature range.

P

Peak torque (software limited):
The maximum torque output (in the direction of motion) with the motor current limited by software.

Position sensor resolution:
The position sensing resolution measured at the input and calculated for the output.

Power supply voltage:
The rated range of power supply tension of the actuator drive.

T

Temperature sensor range and precision:
The range and precision of the temperature sensor mounted on the actuator chassis.

Thermal time constant of the frame:
An indicator of the thermal response time (first order system approximation) of the frame. When a torque load is applied, the winding heats first and then start to heat the more massive frame (which has thus a slower response).

Thermal time constant of the winding:
An indicator of the thermal response time (first order system approximation) of the winding.

Torque gradient:
The ratio of torque output to motor current calculated without gearing losses. The actual torque applied on the load depends on motion direction and gearing efficiency.

Torque sensor cross-axis torque sensitivity:
The effect of torque applied perpendicularly to the actuator axis on the measured torque (torque measure bias / cross-axis torque).

Torque sensor precision (room temperature):
The precision of the sensor at 23 °C under a pure moment loads.

Torque sensor temperature drift (-10 °C to 40 °C):
The maximum effect of temperature on torque measurement precision.

W

Weight:
The weight of the actuator module.

Appendix C

Kinova-ROS: Installation

To make kinova-ros part of your workspace, follow these steps:

```
user@hostname$ mkdir -p ~/catkin_ws/src
user@hostname$ cd ~/catkin_ws/src
user@hostname$ git clone https://github.com/Kinovarobotics/kinova-ros.git kinova-ros
user@hostname$ cd ~/catkin_ws
user@hostname$ catkin_make
user@hostname$ source devel/setup.bash
```

To access the arm via usb copy the udev rule file "`10-kinova-arm.rules`" from "`~/catkin_ws/src/kinova-ros/kinova_driver/udev`" to "`/etc/udev/rules.d/`":

```
user@hostname$ sudo cp kinova_driver/udev/10-kinova-arm.rules /etc/udev/rules.d/
```

All functionalities available in USB are available in Ethernet. To use ethernet connection follow these steps:

1. Setup a static IP address for your ethernet network say - 192.168.100.100;
2. With the robot connected to your PC via USP open kinova's Development Center;
3. Open tab General/Ethernet - Set robot IP Address to something like - 192.168.100.xxx;
4. Make sure MAC address is not all zero;
5. Press 'Update' and restart robot;
6. In a terminal ping your robot's IP, then your robot is setup for ethernet.

To connect to robot via ethernet in ROS just set these parameters in *kinova_bringup*/launch/ *robot_parameters.yaml*

```
connection_type: ethernet
local_machine_IP: [your PC network IP]
subnet_mask: [your network subnet mask]
```

Appendix D

Kinova-ROS: How to use the stack

D.1 Gazebo Simulator

Launching Gazebo with ros_control:

```
roslaunch thesis_control j2n6s300_gazebo.launch
```

To run the nodes referred to the package thesis_demo in Gazebo:

```
rosrun thesis_demo joint_pos_control.py
rosrun thesis_demo surface_polishing_gazebo.py
```

Use RQT to send commands:

```
rosrun rqt_gui rqt_gui
```

Use RVIZ to visualize the robot:

```
rosrun rviz rviz
1) Fixed Frame: root
2) Add: RobotModel
```

To launch moveIt with Gazebo:

```
roslaunch j2n6s300_moveit_config j2n6s300_gazebo_demo.launch
```

To visualize a list of the available topics:

```
rostopic list
```

To get info about a specific topic:

```
rostopic info /'{...}'
```

To visualize a list of the available services:

```
rosservice list
```

To get info about a specific service:

```
rosservice info /'{...}'
```

D.2 Real JACO^2

Lauching the essential drivers and configurations for JACO^2:

```
roslaunch kinova_bringup kinova_robot.launch kinova_robotType:=j2n6s300 use_urdf:=true
```

use_urdf specifies whether the kinematic solution is provided by the URDF model. This is recommended and it is the default option.

Send the robot to HOME position:

```
rosservice call /j2n6s300_driver/in/home_arm
```

To launch moveIt with the real robot:

```
roslaunch j2n6s300_moveit_config j2n6s300_demo.launch
```

To switch from position to torque control and vice-versa:

```
rosservice call j2n6s300_driver/in/set_torque_control_parameters
rosservice call j2n6s300_driver/in/set_torque_control_mode 1
(Switch to Torque control: 1 | Position control: 0)
```

To calibrate the robot:

```
rosrun thesis_demo calibration_robot.py
```

To run the nodes referred to the package thesis_demo in JACO^2:

```
rosrun thesis_demo joint_space_robot.py
rosrun thesis_demo surface_polishing_Q.py
```

To acess API functions:

```
rosrun kinova_driver kinova_api_funcs
rosrun thesis_demo kinova_api_wrapper.py
```

More information is available in [9].

Appendix E

Robot Operating System

Introduction

The Robot Operating System (ROS) is an open-source, framework for create robot software. It is a collection of tools, libraries and conventions that simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.

Why ROS? Because creating truly robust robot software is really hard. For example, many problems and tasks that seem trivial to humans could be considered difficult from the robots perspective.

Dealing with real-world alterations in complex tasks and environments is so difficult that no single individual, laboratory or institution can imagine building a complete system from scratch. As result, ROS was built from the ground up to encourage collaborative robotics software development. [14]

Brief History

The ROS project was started in 2007 with the name *Switchyard* by Morgan Quigley as part of the Stanford STAIR (STanford AI Robot) project. In January, the main development of ROS happened at willow Garage, a robotics research institute/incubator. The effort was promoted by uncounted researchers who contributed their time and skills to the heart of ROS and its fundamental software packages.

From the beginning, ROS was being developed at different institutions and for multiple robots. At first, this appeared to be a mess, because it was far simpler for all contributors to place their code on the same servers. Ironically, over the years, this has emerged as one of the great strengths of the ROS ecosystem: any group can start their own repository on their own servers, and maintain full ownership and control of it. They don't need any-ones permission. If they choose to make their repository publicly, they can receive the recognition and credit they deserve and benefit from specific technical feedback and improvements like all open source software projects.

Nowadays, the ROS world consists of tens of thousands of users worldwide, working on projects ranging from hobbies to large industrial automation systems. [14]

Goals

The primary goal is to support code *reuse* in robotics research and development. ROS is a distributed framework of processes (also known as *Nodes*) that enables executables to be individually designed and loosely coupled at runtime. These processes can be organized into *Packages* and *Stacks*, which can be easily shared and distributed. ROS also supports a federated system of code *Repositories* that enable collaboration to be distributed as well. This design, from the file system level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools. [16]

In support of this primary goal there are several other goals of the ROS framework:

- Easy integration with other robot software frameworks: ROS has already been integrated with OpenRAVE, Orocos, and Player;
- Lightweight;
- Language independence: the ROS framework is easy to implement in any modern programming language (commonly used with Python, C++ and Lisp);
- Multi-platform: it is actually available for UNIX and MAC systems, but efforts are being made to make it fully compatible with Microsoft Windows OS;
- Scaling: ROS is appropriate for large runtime systems and for large development processes.

Concepts

ROS has three level of concepts: the File-System level, the Computation Graph level, and the Community level. [17]

File-System level

The following graph (fig.E.1) shows how ROS file-system and folder are organized on the disk:

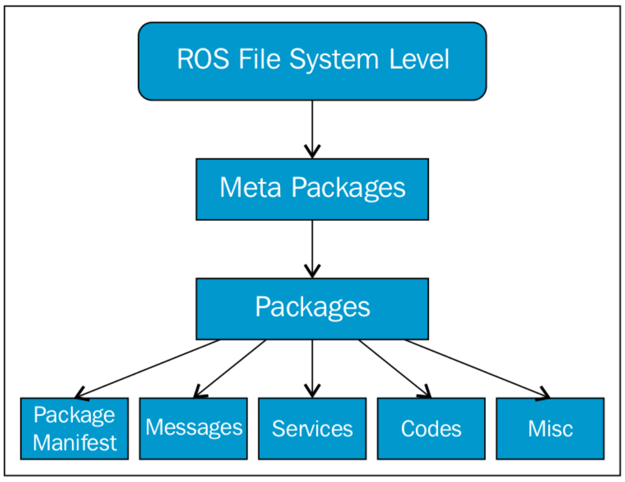


Figure E.1: ROS File-System level. [10]

The file-system level concepts mainly cover ROS resources that you find on disk, such as:

- **Packages:** The ROS packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in the ROS software.

- **Meta-packages:** The term meta-package is used for a group of packages for a special purpose. In an older version of ROS such as Electric and Fuerte, it was called stacks, but later it was removed, and meta-packages came to existence. One of the examples of meta-package is the ROS navigation stack.

- **Package Manifests:** The package manifest file is inside a package that contains information about the package, including its name, author, licence, dependencies, compilation flags, and other meta information like exported packages. The "*package.xml*" file inside the ROS package is the manifest file of that package.

- **Repositories:** Most of the ROS packages are maintained using a **Version Control System** (VCS) such as GitHub, subversion (svn), mercurial (hg), and so on. The collection of packages that share a common VCS can be called repositories. The package in the repositories can be released using a catkin release automation tool called *bloom*.

- **Message (msg) types:** The ROS messages are a type of information that is sent from one ROS process to another. We can define a custom message inside the *msg* folder inside a package (*my_package/msg/MyMessageType.msg*).

- **Services (srv) types:** The ROS service is a kind of request/reply interaction between processes. The reply and request data types can be defined inside the *srv* folder inside the package (*my_package/srv/MyServiceType.srv*).

Computation Graph Level

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The main concepts in the computation graph are *Nodes*, *Master*, *Parameter server*, *Messages*, *Topics*, *Services*, and *Bags*. All of which provide data to the Graph in different ways.

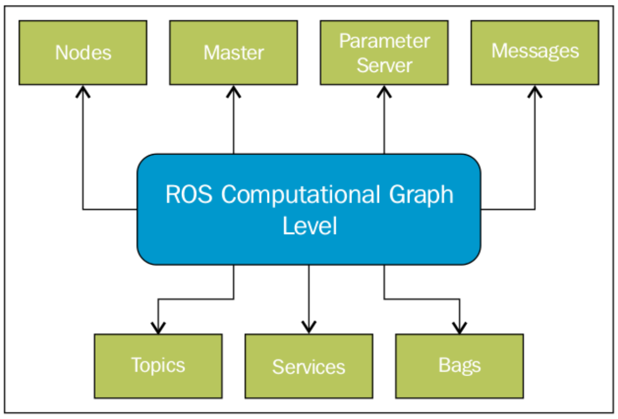


Figure E.2: Structure of the ROS Graph layer. [10]

These concepts are implemented in the *ros_comm* (`http://wiki.ros.org/ros_comm`) repository.

- **Nodes:** Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheels motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on. Each ROS node is written using ROS client libraries such as *roscpp* (`http://wiki.ros.org/roscpp`) or *rospy* (`http://wiki.ros.org/rospy`).

- **Master:** The ROS Master provides name registration and lookup to the rest of the nodes. Nodes will not be able to find each other, exchange messages, or invoke services without ROS Master. In a distributed system, we should run the master on one computer, and other remote nodes can find each other by communicating with this master.
- **Parameter Server:** The parameter server allows you to keep the data to be stored in a central location. All nodes can access and modify these values. Parameter server is currently part of the Master.
- **Messages:** Nodes communicate with each other by passing messages. Messages are simply a data structure containing the typed field, which can hold a set of data and that can be sent to another node. There are standard primitive types (integer, floating-point, boolean, and so on) and these are supported by ROS messages. We can also build our own message types following these standard types.
- **Topics:** Each message in ROS is transported using named buses called topics. When a node sends a message through a topic, we can say the node is publishing a topic. When a node receives a message through a topic, we can say the node is subscribing to a topic. The publishing node and subscribing node are not aware of each other existence. For example, we can subscribe a node that might not have any publisher. The idea is to decouple the production of information from its consumption. Logically, we can think of a topic as a strongly typed message bus. Each bus has a unique name, and any node can access this topic and send data through it as long as they have the right message type.
- **Services:** The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/reply interactions, which are often required in a distributed system. Request/reply is done via services, which are defined by a pair of message structures: one for the request and other for the reply. A providing node offers a service under a name, and when the client node sends a request message to this server, it will respond and send the result

to the client. The client might need to wait until the server responds. The ROS service interaction is like a remote procedure call.

- **Bags:** Bags are a format for saving and playing back ROS message data. They are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

The ROS Master acts as a name-service in the ROS Computation Graph. It stores topics and services registration for ROS nodes. Nodes communicate with the Master to report their registration information. As soon as these nodes communicate with the Master, they can receive information about other registered nodes and make connections. The Master will also make callbacks to these nodes when the registration information changes, which allows nodes to dynamically create connections with the new nodes that are running.

Nodes connect to other nodes directly; the Master only provides lookup information, like a DNS server. Nodes that subscribe to a topic will request connections from nodes that publish that topic, and will stablish connection over an agreed upon connection protocol. The most common protocol used in ROS is called *TCPROS* (`http://wiki.ros.org/ROS/TCPROS`), which uses standard TCP/IP sockets.

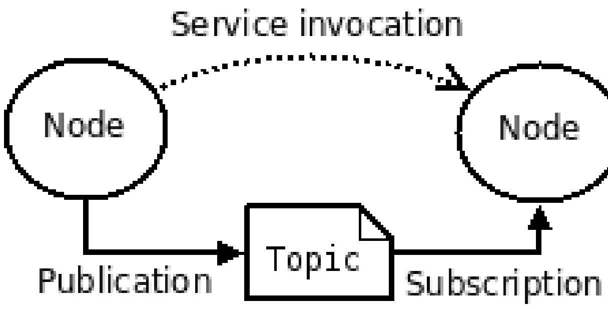


Figure E.3: ROS basic concepts. [10]

Community Level

The ROS community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include:

- **Distributions:** ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent across a set of software.

- **Repositories:** ROS on a federated network of code repositories, where different institutions can develop and release their own robot software components.
- **The ROS Wiki:** The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and much more.
- **Mailing Lists:** The ros-users mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about ROS software.
- **ROS Answers:** A site (`https://answers.ros.org/questions/`) for answering your ROS related questions.
- **Blog:**: The Blog (`http://www.ros.org/news/`) provides regular updates, including photos and videos.

Higher-Level Concepts

The higher-level concepts [2] are provided for helping the building of larger systems on top of ROS:

- **Coordinate Frames/Transforms:** The *tf* package provides a distributed ROS-based framework for calculating the positions of multiple coordinate frames over time.
- **Actions/Tasks:** The *actionlib* package provides tools to create servers that execute long-running goals that can be pre-empted. It also provides a client interface in order to send requests to the server.
- **Message Ontology:** The *common_msgs* stack provides a standard base message ontology for robotics systems, even if the definition of messages can be arbitrary.
- **Plugins:** The *pluginlib* package provides tools for writing and dynamically loading plugins using the ROS build infrastructure.
- **Filters:** The *filters* package provides a C++ library for processing data using a sequence of filters.
- **Robot Model:** The *urdf* package defines an XML format for representing a robot model and provides a C++ parser.

Appendix F

Polishing Task Example