UNIVERSIDADE Ð COIMBRA

Rui Pedro de Matos Fernandes

ROBOTIC TELEMANIPULATION FOR SURFACE POLISHING

Master's Dissertation in MIEEC, supervised by Prof. Dr. Rui Pedro Duarte Cortesão
and presented to the Faculty of Science and Technology of the University of Coimbra

February 2019

Faculty of Science and Technology of the University of Coimbra

ROBOTIC TELEMANIPULATION FOR SURFACE POLISHING

Rui Pedro de Matos Fernandes

Master's Dissertation in MIEEC, in the Automation specialization, supervised by Prof. Dr. Rui Pedro Duarte Cortesão and presented to the Faculty of Science and Technology of the University of Coimbra.

February 2019

UNIVERSIDADE Đ COIMBRA

Acknowledgments

Primeiro, quero agradecer aos meus pais, António e Nazaré, por me terem criado e sempre acreditado em mim ao longo deste percurso, também quero agradecer ao meu irmão, João, pela paciência em certas situações.

Segundo, agradeço ao meu orientador, o Professor Doutor Rui Cortesão, por me ter orientado e partilha de experiência ao longo do trabalho.

Terceiro, quero agradecer à Leticia Simon pela infinita compreensão que teve comigo em muitas situações e por me apoiar até ao fim.

Quarto, agradeço aos meus colegas de laboratório, Hélio Ochoa, Luís Santos e João Pereira, pelas ideias que me deram, a ajuda, a partilha de experiência e amizade.

Finalmente, quero agradecer aos meus colegas de curso, nomeadamente ao Manel Abrantes, Ricardo Lopes, João Palhinha, Rui Baptista e Miguel Maranha, pela amizade e apoio ao longo deste trabalho.

Abstract

The polishing process of a mold is still applied manually by skilled workers who are capable of adjusting the polishing motion and force according to the workpiece conditions based on their own experience. This work is going to focus in the development of a control architecture and an algorithm to execute a desired pattern to perform a polishing task of a surface via telemanipulation.

For that, an extensive study is going to be made to find a feasible control architecture and to develop an intuitive telemanipulation system that can complement the operator's work and the Kinova JACO^2, the robot that is going to be used to execute the polishing task and where all the control architectures are tested beforehand.

The control architectures are going to be implemented in simulation and real environment and correspond to computed torque control in the joint and task space. Then, the different impedance controls are implemented and tested in both environments. The results taken in free-space with the impedance control were not satisfactory because of coupling problems between the position and orientation control, since in this robot the actuator's friction in the last 3 joints is dominant as opposed to the first 3 where the mass is.

To bypass this problem, two more controls are developed that decouples the position and orientation control in two different controllers. One of this controllers takes into consideration the null-space of the position control when computing the torque for orientation. This is the chosen control for the polishing task, which is also tested in simulation and real environment.

The results taken show that this control can be effectively used for executing the polishing task, even though the orientation still needs a little more work. Also, the developed algorithm gives the possibility of an operator to execute the task once and then the robot replicates as many times as the operator desires.

KEYWORDS: polishing, control, algorithm, telemanipulation, JACO^2, joint space, task space, impedance control, null-space, SpaceMouse

Resumo

O processo de polimento de um molde ainda é realizado manualmente por trabalhadores experientes que são capazes de ajustar o movimento e força do polimento dependendo das condições da peça de trabalho e baseando-se na sua própria experiência. Este trabalho vai-se focar no desenvolvimento de uma arquitetura de controlo e num algoritmo para executar um padrão desejado para realizar a tarefa de polimento de uma superfície via telemanipulação.

Para isso, um estudo intensivo vai ser realizado para encontrar uma arquitetura de controlo viável e desenvolver um sistema de telemanipulação intuitivo que consegue complementar o trabalho do operador e o Kinova JACO^2, o robô que vai ser utilizado para executar a tarefa de polimento e onde todas as arquiteturas de controlo são testadas antecipadamente.

As arquiteturas de controlo vão ser implementadas em ambiente de simulação e em ambiente real e correspondem a um controlo do torque computado no espaço das juntas e de tarefa. De seguida, os diferentes controlos de impedância são implementados e testados em ambos os ambientes. Os resultados retirados em espaço-livre com o controlo de impedância não foram satisfatórios devido a problemas de acoplação entre o controlo de posição e orientação, visto que, a fricção dos atuadores do robô nas últimas 3 juntas é dominante, ao contrário ao que acontece nas primeiras 3 onde a massa é que é a componente dominante.

Para ultrapassar este problema, mais dois controlos foram desenvolvidos para desacoplar o controlo de posição e orientação em dois controladores diferentes. Um destes controladores tem em consideração o espaço-nulo da posição quando é computado o torque para a orientação. Este controlo é o escolhido para a tarefa de polimento, que também é testada em ambiente de simulação e real.

Os resultados obtidos mostram que este controlo pode ser usado, efetivamente, para executar uma tarefa de polimento, apesar de a orientação ainda precisar um pouco mais de trabalho. O algoritmo desenvolvido dá a possibilidade do operador executar a tarefa uma vez e de seguida o robô replica as vezes que o operador desejar.

PALAVRAS-CHAVE: polimento, controlo, algoritmo, telemanipulação, JACO^2, espaço das juntas, espaço de tarefa, controlo de impedância, espaço-nulo, SpaceMouse

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface.
CAD	Computer-aided design.
COM	Center of Mass.
DK	Dynamic Kinematics.
DOF	Degree of Freedom.
DSP	Digital Signal Processing.
FK	Forward Kinematics.
GUI	Graphical User Interface.
KDL	Kinematics and Dynamics Library.
PD	Proportional-Derivative.
PID	Proportional-Integrative-Derivative.
ROS	Robot Operating System.
SDK	Software Development Kit.
URDF	Unified Robot Description Format.
USB	Universal Serial Bus.
XML	eXtensible Markup Language.

Chapter 1. Introduction

1.1 Background

Polishing is a kind of finishing process that can effectively eliminate or reduce the processing defects caused by manufacturing procedures and improve surface quality and form accuracy, which in turn makes it a vital role to ensure the product quality and the service life [1]. The polishing process of a mold not only affects the final appearance and its quality, but it also occupies the total production time and can cost up to 40% of it [2]. This process is still applied manually by skilled workers who are capable of adjusting the polishing motion and force according to the workpiece conditions based on their own experience.

Nowadays, the number of skilled workers is decreasing as the current workforce ages. In addition, the polishing process can be harmful to the operator's health because of the scattering of abrasives and buffed materials. Furthermore the surface quality is dependent on the proficiency level of the worker [3]. Therefore, it has become a hot topic in the field of engineering and academics. In the research of polishing automation, a critical and difficult problem is how to precisely plan the tool-path, namely the trajectory planning for the polishing process, in order to achieve the quality requirements [1].

A bilateral telemanipulation system allows a human operator to interact without direct physical contact with the environment using a master-slave pair of manipulators. A scaled bilateral telemanipulation system scales the amount of power transferred between the human and environment that has length, force and power scales that are very different from those of the human. Ideally, such a system may scale but would otherwise preserve the ”feel” of the environment with which the operator interacts. In reality, however, the system cannot completely preserve this information, but rather will filter and, thus, alter the perceived dynamic character of the environment.

The extent to which the manipulator system preserves the feel of the environment is characterized by the ”transparency” of the system. In addition, a bilateral telemanipulator system should relay the feel of the environment to the operator in a robustly stable manner. As such, the control system should be such that the human-telemanipulator-environment loop remains stable when subjected to significant perturbations in the environmental or human dynamics [4].

1.2 Objectives

This work is going to focus in the development of a control architecture and an algorithm to execute a desired pattern to perform a polishing task of a surface via telemanipulation.

In order to achieve that feat, an extensive study is going to be made to find a feasible control architecture and to develop an intuitive telemanipulation system that can complement the operator's work and the Kinova JACO^2's behavior, the robot that is going to be used to execute the polishing task and where all the control architectures are tested beforehand.

In the end, a demo that validates this concept is shown.

1.3 Contributions

With this work it is possible test the control architectures and the polishing task in Gazebo, the simulator used to test them which has Robot Operating System (ROS) support, and also in the real JACO^2 robot.

It also introduces telemanipulation to the polishing task and the developed algorithm can be used to for a multitude of surfaces and encourages the human-machine interaction.

1.4 Organization

- **Chapter 1:** Introduces the work, including background, objectives, contributions and organization (1);
- **Chapter 2:** Covers the JACO^2 robot developed by Kinova Robotics, including the robot's system, its communication modes and its preferred one and a general view of the robot's kinematics and dynamics (2);
- **Chapter 3:** Covers the SpaceMouse navigator developed by 3Dconnexion which was the peripheral used for telemanipulation during this work, it includes an overview of the system, the communication modes and how the motion control of the robot was developed with said peripheral (3);
- **Chapter 4:** Presents the control architectures studied and the chosen one for the polishing task (4);
- **Chapter 5:** Analyses the results from the control architectures presented in the previous chapter both in the Gazebo simulator and the real JACO^2 robot (5);
- **Chapter 6:** Explains the algorithm developed for the polishing task and shows the results in the simulator and the real robot (6);
- **Chapter 7:** Concludes the work developed and features future projects (7).

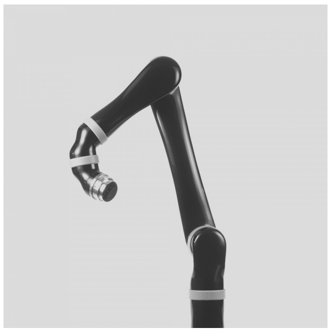
Chapter 2. State of the Art: The Kinova Jaco² Robotic Arm

In this chapter, it is given a general overview about the JACO² Robotic Arm, as well as its main characteristics and functionalities.

2.1 System Overview

The JACO² Robotic Arm was developed by Kinova Robotics[1], a Canadian company that develops assistive/rehabilitation devices. The first version, called JACO Rehab Edition, was launched in 2010 and was aimed to assist people with reduced mobility and upper limb impairments. An updated version was developed in 2012 for scientific purposes called the JACO².

In this work the version used is the JACO² Research Edition (figure 2.1a) which is a 6 degrees of freedom (DOF) manipulator with a Kinova Gripper KG-3 (figure 2.1b) coupled in the last joint. This gripper is composed of three fingers and it can be entirely removed from the robot.


(a) JACO² 6 DOF Robot without the gripper [5].

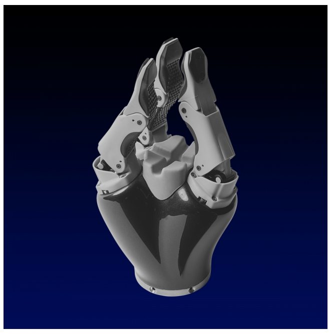

(b) Gripper KG-3 that is coupled to the JACO² robot [6].

Figure 2.1: The Kinova JACO² system.

The robot is composed of six carbon fiber links interconnected with each other, which are jointed together by six aluminum brushless direct current actuators. To note that the actuators do not possess any mechanical limitation, which means that it allows each actuator to have unlimited rotation around their axis. Due to the carbon filter, the main structure of the robot becomes very lightweight.

The Gripper KG-3 consists of three plastic fingers which can be controlled individually. Because of its material composition, the fingers can adjust to many different types of objects.

[1]https://www.kinovarobotics.com/

In table 2.1 there are some general specifications about the JACO² robot that are taken directly from the robot's specifications user guide. In appendix A some more of the robot's specifications are presented [7].

Table 2.1: JACO² General Specifications

General Specifications	
Total Weight	4,4 Kg
Payload capabilities	2.6 Kg (mid-range continuous) 2.2 Kg (full-reach peak/temporary)
Materials	Carbon fiber (links) Aluminum (actuators)
Maximum Reach	90 cm
Joint Range after start-up (Software Limitation)	$\pm$ 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	25 W (Operating Mode) 5 W (Standby Mode)
Communication Protocol	RS-485
Communication Cables	20 pins flat flex cable
Expansion pins	2 (on communication bus)
Water Resistance	IPX2
Operating Temperature	-10 ºC to 40 ºC

The robot comes with a controller (figure 2.2) which can be used to manipulate the arm. It is composed by a three-axis joystick, five independent push-buttons and four external auxiliary inputs placed at its back side. The buttons on the front side are the power button, the HOME button and the other five buttons are for switching between different operation modes. The blue light shows the current operation mode while the green light displays that the robot is powered and ready to be used.

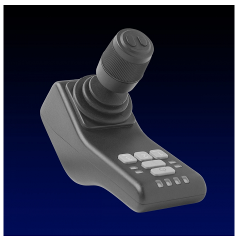
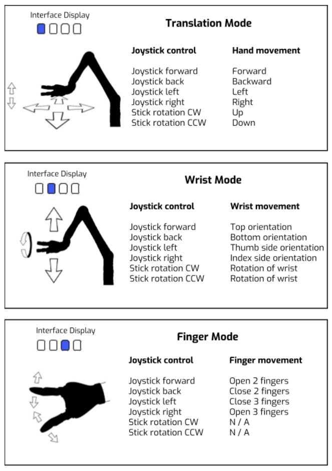


Figure 2.2: Kinova Controller that came with the robot arm [8].

The HOME buttons moves the robot to a preprogrammed pose. The other buttons are for switching between three-axis and two-axis mode. There are a number of modes that are included in the robot, which are: translation mode, wrist mode, "drinking" mode and finger mode. While using the joystick, the user can control the Cartesian position or orientation of the gripper.

As mentioned before, in translation mode the robot hand can be controlled in the three-axis Cartesian coordinate system. In wrist mode, the arm is controlled around a reference point which is set in the middle of the end-effector. While in "drinking" mode, it allows the wrist of the robot to execute a rotation around another point in the space with a set offset in height and length from the reference point. Finally, in finger mode the user can open and close the gripper's fingers. In figure 2.3a it is shown the joystick control when the robot is in three-axis mode, while in figure 2.3b shows the joystick in it is in two-axis mode.

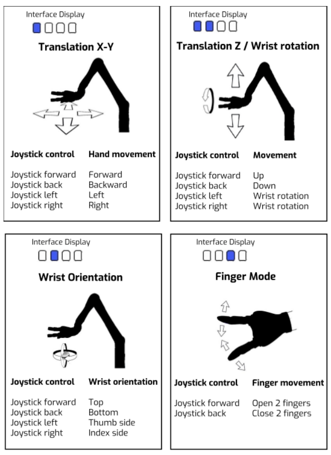


Joystick control	Hand movement
Joystick forward	Forward
Joystick back	Backward
Joystick left	Left
Joystick right	Right
Stick rotation CW	Up
Stick rotation CCW	Down

Joystick control	Wrist movement
Joystick forward	Top orientation
Joystick back	Bottom orientation
Joystick left	Thumb side orientation
Joystick right	Index side orientation
Stick rotation CW	Rotation of wrist
Stick rotation CCW	Rotation of wrist

Joystick control	Finger movement
Joystick forward	Open 2 fingers
Joystick back	Close 2 fingers
Joystick left	Close 3 fingers
Joystick right	Open 3 fingers
Stick rotation CW	N / A
Stick rotation CCW	N / A

(a) Joystick controls in three-axis mode.



Joystick control	Hand movement
Joystick forward	Forward
Joystick back	Backward
Joystick left	Left
Joystick right	Right

Joystick control	Movement
Joystick forward	Up
Joystick back	Down
Joystick left	Wrist rotation
Joystick right	Wrist rotation

Joystick control	Wrist orientation
Joystick forward	Top
Joystick back	Bottom
Joystick left	Thumb side
Joystick right	Index side

Joystick control	Finger movement
Joystick forward	Open 2 fingers
Joystick back	Close 2 fingers

(b) Joystick controls in two-axis mode.

Figure 2.3: Joystick control modes [9].

The arm has temperature, voltage, current and torque sensors to monitor its condition. In terms of hardware, the robot has inputs for the power supply, joystick and ethernet and universal serial bus (USB) ports in order to connect the robot to a personal computer (PC) or laptop for using it with the software development kit (SDK) [10] [11]. In appendix B specifications about the actuators and controller are presented that are taken directly from the actuator's specifications user guide [12].

2.2 Communication Modes

There are three alternatives to communicate with the JACO² robotic arm. The first one is through the SDK user interface (figure 2.4) [13] that enables the user to control the robot in trajectory and torque control without the need to use the joystick.

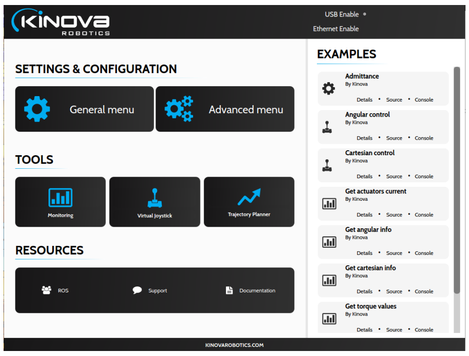


Figure 2.4: SDK main menu.

The second form of communication is the Kinova Application Programming Interface (API). The API has many built-in functions programmed in C++ that allows the user create specific software programs. The internal protocol communication used to control de actuators is the RS485 [10].

The third and final form of communication is the Kinova Robot Operating System (Kinova-ROS) [14]. This is the preferred communication mode for this work, because ROS is very flexible in terms of programming robot software and it simplifies the task of creating a desired robot behaviour. The Kinova-ROS stack provides a ROS interface for the JACO, JACO^2 and MICO robotic manipulator arms which were all developed by Kinova Robotics. This stack was developed with the Kinova C++ API functions, which communicates with the digital signal processor (DSP) which is located inside the robot base.

The recommended configuration for this communication mode is by using ROS Indigo paired with Ubuntu 14.04 64-bit operating system, however for this work it is utilized the ROS Kinetic with Ubuntu 16.04 64-bit because it is the more recent version. The Kinova-ROS supports two types of programming languages, C++ and python. For this work it is used C++, despite being more complex, it is thought that it would lead to better results.

The control system frequency varies depending on what communication method is used. If it is high level, like via USB, the rate is between 100 Hz and 500 Hz, but the refresh rate of the DSP controller is 100 Hz. If the communication is made directly with the actuators, in other words, the low level approach, the communication rate is 500 Hz. For this work, it is made using high level, which means that all the control architectures are developed with the refresh rate of the DSP controller [11].

2.3 Kinova-ROS

During this section there is a brief explanation about the contents of the Kinova-ROS stack [14], since it was the communication mode used throughout this work (section 2.2).

File System

This stack has many files that were divided into several packages:

- *kinova_bringup*: This package contains the launch file for the *kinova_driver* and also applies some other configurations
- *kinova_driver*: In this package it is included the most essential files in order to run the kinova-ros stack. In the **include** folder, it is defined both the Kinova C++ API headers and the ROS package header files, the first one in the *kinova* folder and the former in the *kinova_driver* folder. The *kinova_api* source file is a wrap of the Kinova C++ API, while the *kinova_comm* builds up the fundamental functions. To access some advanced settings regarded to force/torque control are only provided by *kinova_api*. Most parameters and topics are created in *kinova_arm*. In figure 2.5 it is

shown a diagram that summarizes the general architecture of the *kinova_driver* from low level up.

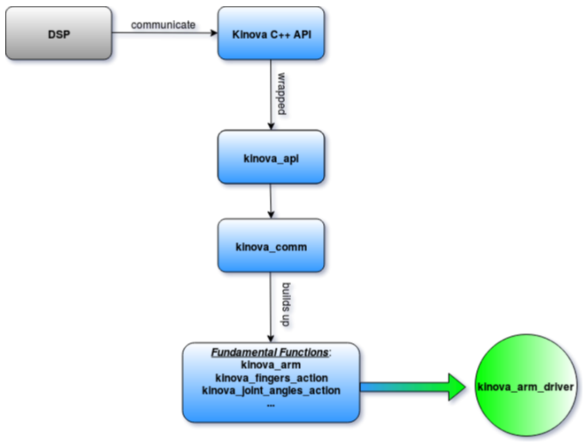


Figure 2.5: Diagram of the *kinova_driver* package general architecture from low level up.

- *kinova_demo*: This package contains the python scripts for the *actionlibs* in joint space and task space.
- *kinova_messages*: In this package all the *messages*, *servers* and *actionlib* format are defined here.
- *kinova_description*: This package provides the Unified Robot Description Format (URDF) models for the robots and their respective meshes.
- *kinova_docs*: This final package has html files for the *kinova_comm* reference which were generated by doxygen.

Controllers

This stack came with a **Joint Position Control** and a **Cartesian Position Control**. There is also a **Finger Position Control** in order to open and close the robot's fingers and a **Velocity Control** for both joint space and task space.

It is also possible to switch between **position control** and **torque control**, and publish torque commands, which is important during the development of this work. Because of the nature of the actuators, sometimes the robot needs to be re-calibrated, since switching between **torque** and **position control** and sending torque commands creates an offset in its torque sensors.

Gazebo

The Gazebo simulator was the simulator used to develop the robot controllers and to analyze its behavior before executing them on the robot. The stack came with a Gazebo package [15], *kinova_gazebo*. In figure 2.6 it is represented the Kinova JACO² 6 DOF robot in the Gazebo simulator environment.

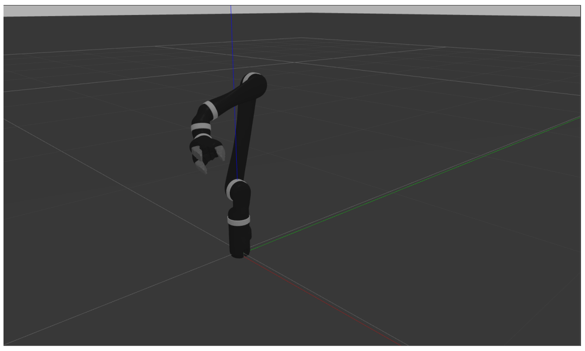


Figure 2.6: The Kinova JACO² Robot in Gazebo simulator.

This package uses *ros_control* to control the robot, as such there exists three types of controllers that can be used: effort, position and velocity. All the configuration files needed to control the robot using *ros_control* are located in the *kinova_control* package. In figure 2.7 there is a diagram that gives a general overview simulation, hardware, controllers and transmission.

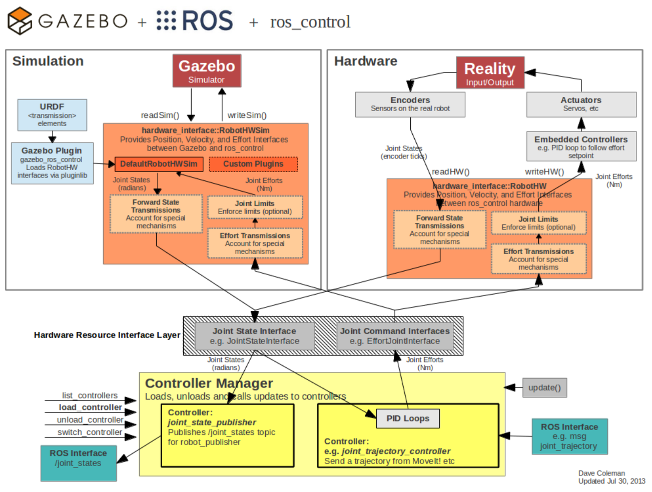

GAZEBO + ROS + ros_control
Simulation
Gazebo Simulator
URDF <transmission> elements
Gazebo Plugin gazebo_ros_control Loads RobotHW interfaces via pluginlib
readSim()
writeSim()
hardware_interface::RobotHWSim Provides Position, Velocity, and Effort Interfaces between Gazebo and ros_control
DefaultRobotHWSim
Custom Plugins
Joint States (radians)
Joint Efforts (Nm)
Forward State Transmissions Account for special mechanisms
Joint Limits Enforce limits (optional)
Effort Transmissions Account for special mechanisms
Hardware
Reality Input/Output
Encoders Sensors on the real robot
Actuators Servos, etc
Embedded Controllers e.g. PID loop to follow effort setpoint
Joint States (encoder ticks)
readHW()
writeHW()
Joint Efforts (Nm)
hardware_interface::RobotHW Provides Position, Velocity, and Effort Interfaces between ros_control hardware
Forward State Transmissions Account for special mechanisms
Joint Limits Enforce limits (optional)
Effort Transmissions Account for special mechanisms
Hardware Resource Interface Layer
Joint State Interface e.g. JointStateInterface
Joint Command Interfaces e.g. EffortJointInterface
Joint States (radians)
Joint Efforts (Nm)
Controller Manager Loads, unloads and calls updates to controllers
update()
list_controllers
load_controller
unload_controller
switch_controller
Controller: joint_state_publisher Publishes /joint_states topic for robot_publisher
PID Loops
Controller: e.g. joint_trajectory_controller Send a trajectory from MoveIt! etc
ROS Interface /joint_states
ROS Interface e.g. msg joint_trajectory
Dave Coleman Updated Jul 30, 2013

Figure 2.7: Data flow of *ros_control* and Gazebo [16].

The inertial parameters have been added to all mesh models. Each link has the inertial model for the link and half of the actuator on one end of the link and half on the other end of it. The Center Of Mass (COM) position mass of the links are accurate in order to get correct torque readings. However, the inertia matrix is an approximation to uniform cylinders for the links. The joint dynamics, like damping, friction and stiffness do not accurately represent the hardware [11] [15].

2.4 JACO² Kinematics and Dynamics

In this last section, it is explained how the JACO²'s kinematics and dynamics models are obtained. It is not gonna be given any theoretical background about kinematics and dynamics in general, since other master thesis [10] [11] were made using this robot that focus on this theoretical background.

To get the kinematics and dynamics of this robot it was used the Orocos Kinematics and Dynamics Library (KDL)[2]. With this library it is possible to use solvers in order to compute the forward, inverse and differential kinematics and the dynamics, like the Inertia Matrix and the Gravity Vector. It is also possible to create a KDL chain from an eXtensible Markup Language (XML) URDF file. As previously mentioned, the Kinova-ROS stack provides these files (section 2.3).

To obtain said kinematics and dynamics, it is developed a class in C++ called **jaco_kdl**. In this class it is developed its constructor and many other functions. In the constructor is where it is created the chain based on the URDF file.

The functions developed are in order to obtain the kinematics and dynamics of the robot based on its chain. These are: **forward_kinematics**, **jacobian**, **dynamic** and **quaternion**. It is possible to find each of these parameters based on current joint positions and velocities or by giving arbitrary values. In the **quaternion** function it is used another ROS library called *tf*, which can be used to execute operations between quaternions, since they are more complex than the usual operations.

From the **forward_kinematics** function the following 4x4 homogeneous matrix can be extracted:

$$ {}^{b}_{e}T(q) = \begin{bmatrix} {}^{b}_{e}R(q)_{3x3} & {}^{b}_{e}t(q)_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.1} $$

where:

- ${}^{b}_{e}T(q)$ is the end-effector transformation relatively to the robot's base in reference to its joint positions;
- ${}^{b}_{e}R(q)$ is the end-effector rotation relatively to the robot's base in reference to its joint positions;
- ${}^{b}_{e}t(q)$ is the end-effector translation relatively to the robot's base in reference to its joint positions.

The **jacobian** function retrieves the following 6x6 matrix:

$$ J(q) = \begin{bmatrix} J_{vn} \\ J_{wn} \end{bmatrix} \tag{2.2} $$

where:

- $J(q)$ is the robot's jacobian in reference to its joint positions;
- J_{vn} is the robot's linear jacobian for joint n ($n = 1 \dots 6$), each of these values is 3x1 vector with each line referring to x, y and z.
- J_{wn} is the robot's angular jacobian for joint n ($n = 1 \dots 6$), each of these values is 3x1 vector with each line referring to x, y and z.

[2]`http://www.orocos.org/kdl`

The **dynamic** function retrieves the Mass, or Inertia, and Coriolis matrix, both with dimensions 6x6, and the gravity term with dimension 6x1. These three parameters are calculated in the following way:

$$M(q) = \sum_{i=1}^{6} [m_i J_{vi}^T(q) J_{vi}(q) + J_{wi}^T(q) R_i(q) I_i R_i^T(q) J_{wi}(q)] \tag{2.3}$$

where:

- $M(q)$ is the mass matrix of the robot in reference to its joint positions;
- m_i is the mass of link i;
- $J_{vi}(q)$ is the linear jacobian of link i in reference to its joint positions;
- $J_{wi}(q)$ is the angular jacobian of link i in reference to its joint positions;
- $R_i(q)$ is the rotation matrix of link i relatively to the robot's base in reference to its joint positions;
- I_i is the inertia tensor of link i;

$$C(q,\dot{q}) = \dot{M}(q) - \frac{1}{2}\dot{q}^T \frac{\partial M(q)}{\partial q} \tag{2.4}$$

where:

- $\dot{M}(q)$ is the mass matrix derivative in reference to its joint positions;
- $\dot{q}$ is the joints velocities;
- $\frac{\partial M(q)}{\partial q}$ is the partial derivative of the mass matrix with respect to its joint positions;

$$g(q) = \frac{\partial U(q)}{\partial q} \tag{2.5}$$

where:

- $\frac{\partial U(q)}{\partial q}$ is the partial derivative of the robot's potential energy with respect to its joint positions.

Chapter 3. State of the Art: The 3Dconnexion SpaceMouse Compact

This next chapter is going to focus in the SpaceMouse navigator, which is the chosen peripheral in order to manipulate the JACO² robotic arm, via telemanipulation.

3.1 System Overview

The SpaceMouse Compact (figure 3.1) was developed by 3Dconnexion[1], which is an American company that develops peripherals for manipulation in Computer-aided design (CAD) applications and their respective software.

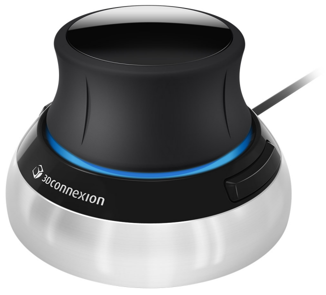


Figure 3.1: The 3Dconnexion SpaceMouse Compact [17].

The SpaceMouse has a 6 DOF sensor that allows the user to manipulate an environment in the X, Y and Z axis and also in their orientation. It is also has two programmable buttons on each side that allows the user to do a multitude of features. In table 3.1 it is shown some technical specifications of the SpaceMouse device that are taken from its website [17].

Table 3.1: SpaceMouse Compact Technical Specifications

Technical Specifications	
Main Features	3Dconnexion Six-Degrees-Of-Freedom (6DOF) sensor 2 Programmable buttons
Dimensions & Weight	Dimensions (LxWxH): 77 x 77 x 54 mm (3.03 x 3.03 x 2.13 in.) Weight: 480 g (16.93 oz)
Supported Operating Systems	Microsoft Windows, MacOS, Linux, SUN Solaris, AIX, HP-UX

[1]`https://www.3dconnexion.com/`

This is the selected device for the robot manipulation because, based on its description and specifications, it is a very lightweight device and with enough practice and the right software development it can be very easy to use and achieve the required task.

3.2 Communication Modes

One of the communication modes for the SpaceMouse is the 3DxWare 10 software that is compatible to almost all the devices developed by 3Dconnexion. This form of communication is not explored during the duration of this work so it is not going to be covered in any detail.

The preferred method of communication with the SpaceMouse is using the *spacenav_node* package [18] which has a ROS interface with the device. This package comes with three launcher files, but during this work only one of them is relevant: the *classic_launcher*. With this file the following ROS topics are created:

- *spacenav/offset* ⇒ publishes the linear component of the joystick's position (approximately normalized to a range of -1 to 1).
- *spacenav/rot_offset* ⇒ publishes the angular component of the joystick's position (approximately normalized to a range of -1 to 1).
- *spacenav/twist* ⇒ combines *offset* and *rot_offset* into a single message.
- *spacenav/joy* ⇒ outputs the spacenav's six degrees of freedom and its buttons as a joystick message.

During this work only the *spacenav/joy* topic is relevant since it is the one that reads the six degrees of freedom and the button presses from the SpaceMouse.

3.3 Motion Control

In order to retrieve the values from the SpaceMouse and use them in the robot, the following functions are developed: **MotionControl** and **joy_callback** both are present in the **jaco_spacenav** class.

In the **joy_callback** is where the software related actions when the user utilizes the SpaceMouse are recognized. If the user uses the 6DOF sensor, depending on what degree it is pressed, a value is stored in a vector called *spacenav_motion*. It has the following structure:

$$spacenav\_motion = \begin{bmatrix} t_x & t_y & t_z & \alpha & \phi & \gamma \end{bmatrix}^T \tag{3.1}$$

where:

- t_x is the translation in the x axis;
- t_y is the translation in the y axis;
- t_z is the translation in the z axis;

- α is the rotation around the x axis (this value is also used to open and close fingers, however it is not the value that is injected in the SpaceMouse, instead there is a predefined threshold in order to do that);
- ϕ is the rotation around the y axis;
- γ is the rotation around the z axis.

If the user presses the left button of the SpaceMouse there is a flag (*flag_mode*) that gets incremented, from 0 to 3, which represents the mode that the SpaceMouse is working in reference to the robot. This are the following modes programmed in the SpaceMouse:

- $flag\_mode = 0 \Rightarrow$ operating in polishing mode, it only permits to control the robot in position in reference to the base axis which is the most important movements that the robot should execute in order to achieve the polishing task, hence the name given;
- $flag\_mode = 1 \Rightarrow$ operating in rotation in robot's end-effector and in translation in its base (this is a more intuitive mode for starters);
- $flag\_mode = 2 \Rightarrow$ operating in robot's base;
- $flag\_mode = 3 \Rightarrow$ operating in robot's end-effector;
- $flag\_mode = 4 \Rightarrow$ operating robot's fingers (this mode was not implemented for the polishing task because the fingers were removed in favor of a proper polishing tool).

In the *MotionControl* function is where the desired pose of the end-effector is calculated in order for the robot to try to achieve it, depending on its developed controller. In this function, there are all the necessary transformations to obtain the desired movement that the user effected in the SpaceMouse, depending on the mode that is selected. The mode that the SpaceMouse is currently on can be viewed in a computer terminal.

Chapter 4. Control Architectures

During this chapter, it is given an overview of the control architectures that are developed to determine which one could achieve a better performance for the polishing task by telemanipulation, which is the objective of this work.

4.1 Computed torque control in the joint space

As a general rule in robotics, the robot's dynamics is given by the following equation:

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

where:

- $M(q)$ is the mass matrix ($n \times n$);
- n is the number of the robot's joints;
- $\upsilon(\dot{q},q)$ represents the Coriolis and centripetal forces ($n \times 1$);
- $g(q)$ is the gravity vector ($n \times 1$);
- τ is the generalized torque acting on q.

This generalized torque (τ) is calculated through the following equation:

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

where:

- τ_c is the computed commanded torque;
- τ_f is the friction torque;
- τ_e is the external torques.

Throughout this work τ_f and τ_e are going to be neglected, which means that :

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

with this approximation τ can be computed in through the following equation:

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

where $(\hat{\cdot})$ means that the dynamic values are an estimation computed through the KDL library (section 2.4).

Equation 4.4 illustrates a system nonlinearly dependent upon the joint velocities. Because of that, feedback linearization and decoupled must be applied in order to cancel nonlinear effects. For that, it needs to be linearized by using a nonlinear feedback law. This law can be written the following way:

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

where $w = \ddot{q}$ is the new control variable. For the purpose of this work, the estimated Coriolis term ($\hat{\upsilon}(\dot{q}, q)$) is going to be neglected since the robot is not going to perform any movements with high velocities. Through equation 4.5 it can be seen that the computed torque (τ_c) directly depends of the joint accelerations ($\ddot{q}$), which means that it can be used for joint space control techniques.

In figure 4.1 it is presented the block diagram for the joint space torque control. From the figure it can be seen that the joints will be controlled by a PID (Proportional-Integrative-Derivative) controller.

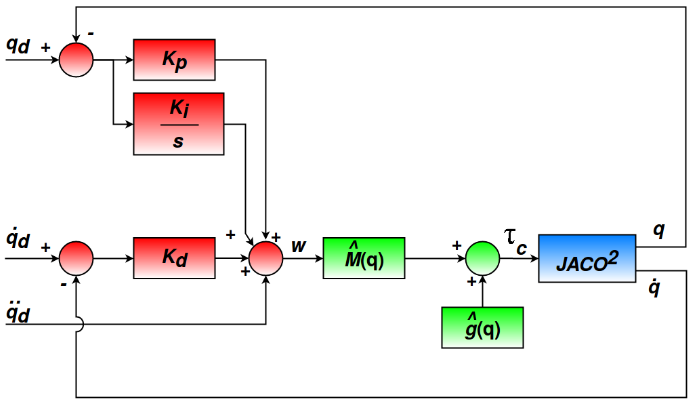


Figure 4.1: PID computed torque control in the joint space.

The computation of $\hat{M}(q)$ and $\hat{g}$ are executed in real time, while w is calculated through the following equation:

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

where:

- q_d, $\dot{q}_d$ and $\ddot{q}_d$ are vectors for the desired joint positions, velocities and accelerations, respectively, each of this vectors is of size n that corresponds to the number of joints;
- q, $\dot{q}$ and $\ddot{q}$ are vectors for the current joint positions, velocities and accelerations, respectively, this vectors are also of size n;
- K_p, K_d and K_i are $n \times n$ positive diagonal matrices with the proportional (K_{pj}), derivative (K_{dj}) and integrative (K_{ij}) gains, where j corresponds to the number of the joint.

Finally, in order to find the proper K_p and K_d values, the following rule is used which is based in the systems natural frequency (ω_n):

$$\begin{aligned} K_{pj} &= \omega_{nj}^2 \\ K_{dj} &= 2\omega_{nj} \end{aligned} \tag{4.7}$$

As for the K_i values, they are put manually as needed.

4.2 Computed torque control in the task space

The next proposed controller is a dynamic torque control in the task space. Through the KDL Library (section 2.4) the Jacobian J can be applied in the following equation in order to compute the task space velocities $\dot{X}$:

$$\dot{X} = J\dot{q} \tag{4.8}$$

With this, the robot's dynamics (4.1) can be written as follows:

$$M(q)J^{-1}(\ddot{X} - \dot{J}\dot{q}) + \upsilon(\dot{q}, q) + g(q) = \tau \tag{4.9}$$

The control law that linearizes and decouples the task space equations is given by:

$$\tau_c = \hat{M}(q)J^{-1}w + \hat{g}(q) \tag{4.10}$$

The Jacobian derivative ($\dot{J}$) has a small contribution to the control process, so it is omitted, as well as the Coriolis term ($\upsilon(q, \dot{q})$) for the same reasons as before. Once again, all the friction and external torques are neglected, so the control variable becomes:

$$w = \ddot{X} \tag{4.11}$$

In figure 4.2 it is shown the diagram of the PD (Proportional-Derivative) controller used to control the robot in the task space.

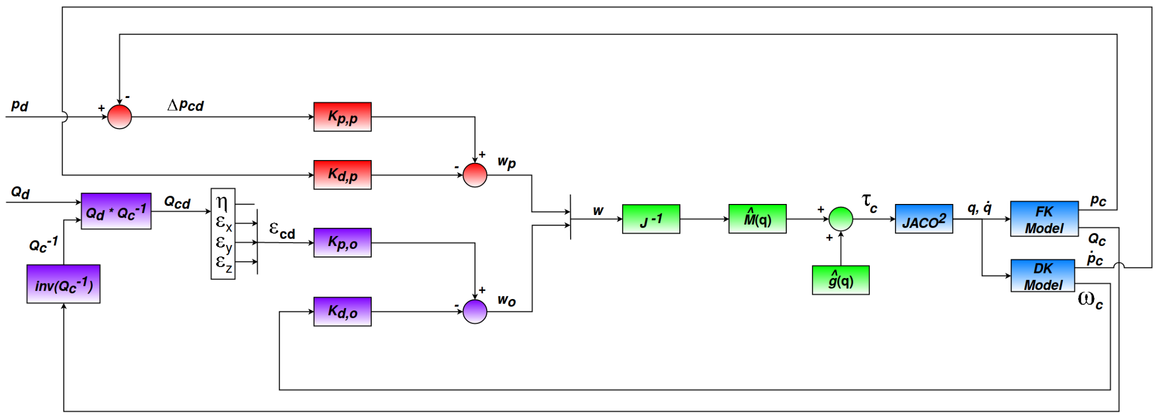


Figure 4.2: PD computed torque control in the task space (FK means forward kinematics and DK dynamic kinematics).

From the diagram, to note that the orientation control is computed using the quaternions, where η corresponds to its scalar part and ε_x, ε_y and ε_z corresponds to the vectorial part, and this is used in every controller throughout this work. In order to compute them it is used the *tf* library which is capable of making all the operations between them. Also, it can be seen that:

$$w = \begin{bmatrix} w_p \\ w_o \end{bmatrix} \tag{4.12}$$

with:

$$\begin{aligned} w_p &= K_{p,p}\Delta p_{cd} - K_{d,p}\dot{p}_c \\ w_o &= K_{p,o}\varepsilon_{cd} - K_{d,o}\omega_c \end{aligned} \tag{4.13}$$

where:

- $K_{p,p}$ and $K_{d,p}$ are 3×3 diagonal matrices for position control with proportional and derivative gains, respectively;
- $K_{p,o}$ and $K_{d,o}$ are 3×3 diagonal matrices for orientation control with proportional and derivative gains, respectively;
- Δp_{cd} is a 3×1 vector with the position error between the desired (p_d) and current (p_c) positions, each line corresponds to a different DOF (p_x, p_y, p_z);
- $\dot{p}_c$ is a 3×1 vector with the current linear velocities, each line corresponds to a different DOF ($\dot{p}_x$, $\dot{p}_y$, $\dot{p}_z$);
- ε_{cd} is a 3×1 vector with the vectorial part of the quaternion error, each line corresponds to a different DOF (o_x, o_y, o_z);
- ω_c is a 3×1 vector with the current angular velocities, each line corresponds to a different DOF (ω_x, ω_y, ω_z).

4.3 Impedance Control

The first two controllers were developed in order to test the SpaceMouse control and to get familiarized with the robot and its ROS interface. This section is going to focus on the controller that would become the basis for the robot's control during the polishing task. The idea of this control is to assign a prescribed robot dynamic behavior in the presence of external interactions, matching the dynamics of mass-spring-damper systems [19].

In order to assign this control to the robot's end-effector velocity ($\dot{X}$) and its force (F_e) are related by a mechanical impedance (Z). In the Laplace domain, it can be written as such:

$$-F_e(s) = Z(s)\dot{X}(s) \tag{4.14}$$

with:

$$sZ(s) = As^2 + Ds + K \tag{4.15}$$

where A, D and K are the mass-spring-damper system gains, respectively. In figure 4.3 it is presented a general impedance control scheme in the task space.

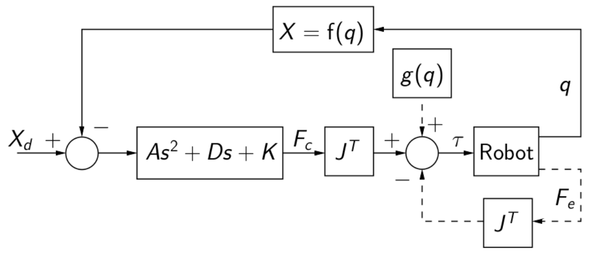


Figure 4.3: Impedance control scheme without force sensing [19].

The overall dynamics of this controller when in contact can be written as:

$$M(q)\ddot{q} + \upsilon(\dot{q}, q) + g(q) = \tau_c - J^T F_e \tag{4.16}$$

with

$$\tau_c = J^T[A(\ddot{X}_d - \ddot{X}) + D(\dot{X}_d - \dot{X}) + K(X_d - X)] + g(q) \tag{4.17}$$

Neglecting the robot dynamics effects, equations 4.16 and 4.17 gives:

$$A(\ddot{X}_d - \ddot{X}) + D(\dot{X}_d - \dot{X}) + K(X_d - X) \approx F_e \tag{4.18}$$

where F_e is the force applied by the robot. 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 [19].

4.3.1 Impedance Control without force sensing and inertia shaping

In figure 4.4 it is shown the impedance control scheme without force sensing and inertia shaping. This one is the most basic impedance control and it can be seen that the mass term A is not present because of its complex implementation. It can be seen that, unlike the previous task space controller (section 4.2), this system has velocity tracking, both for the linear and angular velocities.

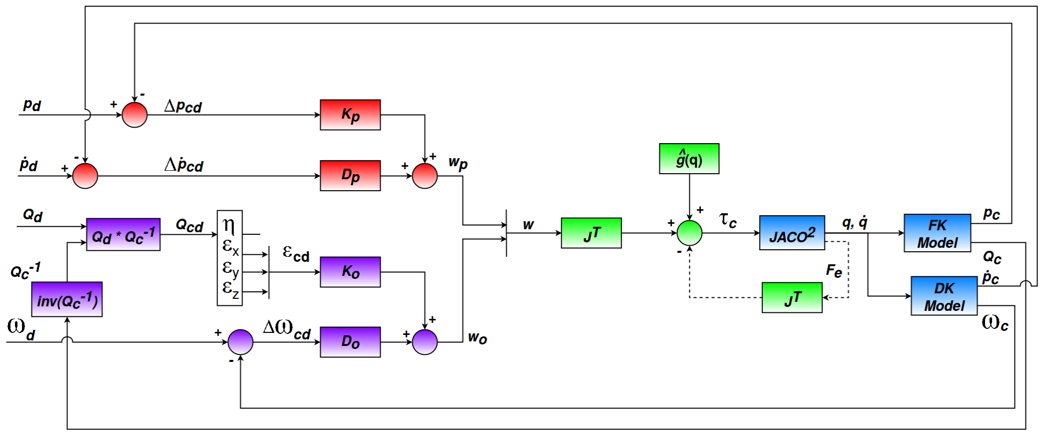


Figure 4.4: Impedance control scheme without force sensing and inertia shaping.

Similarly to the previous controllers a feedback control law for τ_c can be computed:

$$\tau_c = J^T w + \hat{g}(q) \tag{4.19}$$

with w being, once again, the control variable which has the following structure:

$$w = \begin{bmatrix} w_p \\ w_o \end{bmatrix} \tag{4.20}$$

where its components are calculated through the following expressions:

$$\begin{aligned} w_p &= K_p \Delta p_{cd} + D_p \Delta \dot{p}_{cd} \\ w_o &= K_o \varepsilon_{cd} + D_o \Delta \omega_{cd} \end{aligned} \tag{4.21}$$

where:

- K_p and K_o are 3×3 diagonal matrices with the spring gains for position (p) and orientation (o) control, each value corresponds to a DOF (x, y, z);
- D_p and D_o are 3×3 diagonal matrices with the damper gains for position (p) and orientation (o) control, each value corresponds to a DOF (x, y, z);
- $\Delta \dot{p}_{cd}$ is a 3×1 vector with the linear velocities error between the desired ($\dot{p}_d$) and the current ($\dot{p}_c$) linear velocities, each line corresponds to a different DOF ($\dot{p}_x$, $\dot{p}_y$, $\dot{p}_z$);
- $\Delta \omega_{cd}$ is a 3×1 vector with the angular velocities error between the desired (ω_d) and the current (ω_c) angular velocities, each line corresponds to a different DOF (ω_x, ω_y, ω_z).

4.3.2 Impedance Control with force sensing

This section introduces force sensing into the impedance controller of the previous section (section 4.3.1). With force sensing the robot's dynamics are considered when computing τ_c. Once again, starting with the robot dynamics in contact:

$$M(q)\ddot{q} + \upsilon(\dot{q}, q) + g(q) + J^T F_e = \tau_c \tag{4.22}$$

where:

$$A(\ddot{X}_d - \ddot{X}) + D(\dot{X}_d - \dot{X}) + K(X_d - X) = F_e \tag{4.23}$$

which can be rewritten as:

$$\ddot{X} = \ddot{X}_d + A^{-1}[D(\dot{X}_d - \dot{X}) + K(X_d - X) - F_e] \tag{4.24}$$

In figure 4.5 it is shown the control scheme with force sensing. From the scheme it can be deduced that this controller needs the implementation of the J^{-1} which is impossible for redundant robots, that means that for them a different type of controller in order to achieve force sensing has to be developed. To note, that this system has acceleration tracking unlike the previous one (section 4.3.1), as seen by the inclusion of the term a_d.

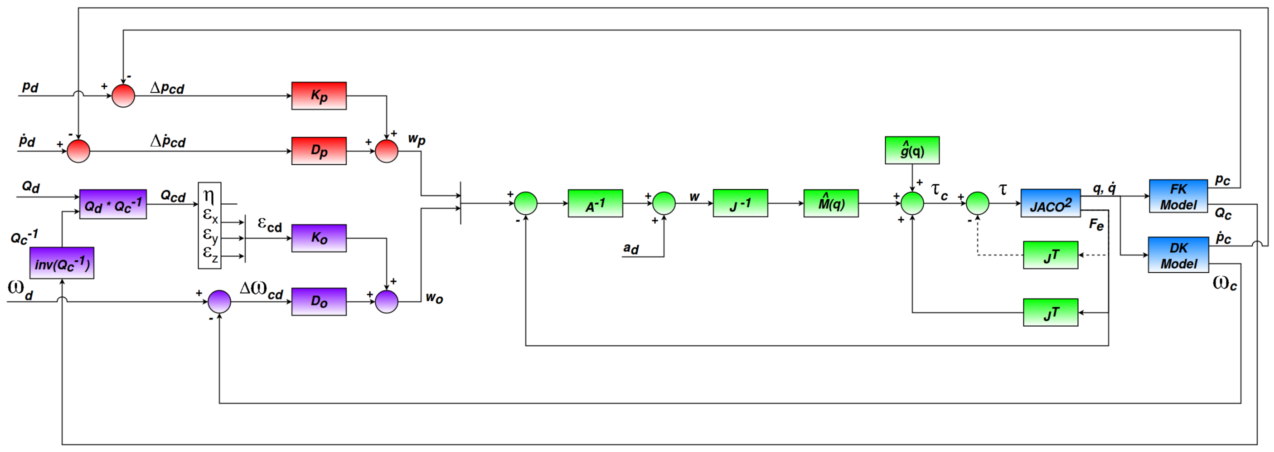


Figure 4.5: Impedance control scheme with force sensing.

For this system the control law can be computed as:

$$\tau_c = \hat{M}(q)J^{-1}w + \hat{g}(q) + J^T F_e \tag{4.25}$$

with the control variable w being, once again:

$$w = \ddot{X} \tag{4.26}$$

and it is computed the following way:

$$w = a_d + A^{-1} \begin{bmatrix} w_p - f_e \\ w_o - \mu_e \end{bmatrix} \tag{4.27}$$

where:

- a_d is a 6×1 vector with the desired linear and angular end-effector accelerations, each line corresponds a different DOF (a_{dx}, a_{dy}, a_{dz}, α_{dx}, α_{dy}, α_{dz});
- A^{-1} is a $n \times n$ matrix of the inverse mass term, with n being the number of joints;
- f_e and μ_e are 3×1 vectors corresponding the force and torque in the end-effector, respectively, each line represents a different DOF (f_{ex}, f_{ey}, f_{ez}, μ_{ex}, μ_{ey}, μ_{ez}).

To start off, the inverse mass term A^{-1} is going to be calculated the following way:

$$A^{-1} = JM^{-1}J^T \tag{4.28}$$

with this approximation it means that there is no inertia shaping in the task space. In this case, the F_e term in equations 4.25 and 4.27 are eliminated, which means that no force feedback is needed [19]. With this the control variable w can computed through the following expression:

$$w = a_d + A^{-1} \begin{bmatrix} K_p \Delta p_{cd} + D_p \Delta \dot{p}_{cd} \\ K_o \varepsilon_{cd} + D_o \Delta \omega_{cd} \end{bmatrix} \tag{4.29}$$

4.3.3 Impedance Control with force sensing for redundant robots

As it was mentioned in the previous section (section 4.3.2) the computation of J^{-1} is impossible for redundant robots, so a different approach must be implemented in order to achieve force sensing in the end-effector with an impedance control.

In robotics, a redundant robot is one that has a different number of joints than his DOF. This is not the case for the JACO² robot, but it is interesting to study its performance and this controller is going to be useful for the implementation of the final two controllers in this chapter (section 4.4).

Starting again from the robot dynamics in contact:

$$M(q)\ddot{q} + \upsilon(\dot{q}, q) + g(q) + J^T F_e = \tau_c \tag{4.30}$$

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

$$M(q)\ddot{q} = \tau' \tag{4.31}$$

with:

$$\tau_c = \tau' + \upsilon(\dot{q}, q) + g(q) + J^T F_e \tag{4.32}$$

By shifting $M(q)$ to the right side in equation 4.31 and multiplying both sides by J, the following expression can be written:

$$J\ddot{q} = JM(q)^{-1}\tau' \tag{4.33}$$

Since:

$$\dot{X} = J\dot{q} \rightarrow J\ddot{q} = \ddot{X} - \dot{J}\dot{q} \tag{4.34}$$

and by applying a Cartesian force F_c through τ', equation 4.33 becomes:

$$\ddot{X} - \dot{J}\dot{q} = JM(q)^{-1}J^T F_c \tag{4.35}$$

The inertia matrix in the task space $\Lambda(q)$ has the following inverse:

$$\Lambda(q)^{-1} = JM(q)^{-1}J^T \tag{4.36}$$

which always exists for any kind of robot, even if J is a non-square matrix, while $M(q)$ is always a square because is a $n \times n$ matrix, with n being the number of joints, so its inverse is always possible. Therefore, from equations 4.35 and 4.36, the dynamic equation in the task space can be written as:

$$\Lambda(q)\ddot{X} - \Lambda(q)\dot{J}\dot{q} = F_c \tag{4.37}$$

In figure 4.6 it is presented the impedance control scheme with force sensing for redundant robots. This figure demonstrates that this controller is similar to the previous one (section 4.3.2) with the exception of utilizing J^T instead of J^{-1} and $\Lambda(q)$ instead of $M(q)$.

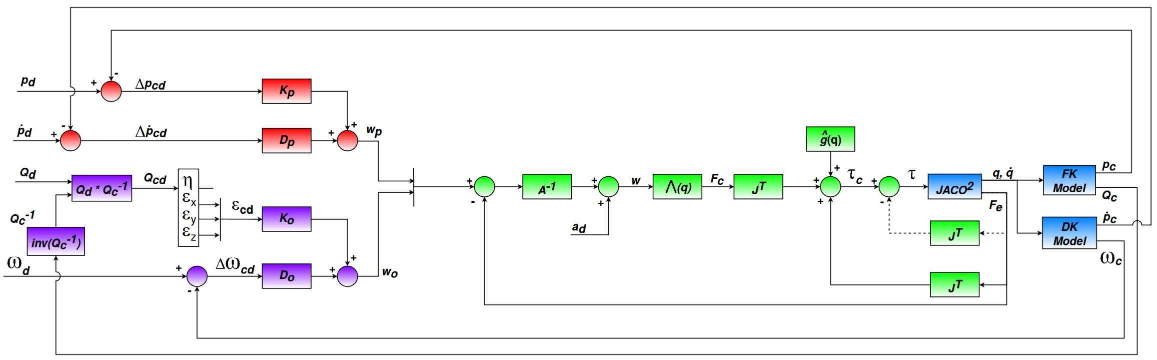


Figure 4.6: Impedance control scheme with force sensing for redundant robots.

The control law for this controller ends up being:

$$\tau_c = J^T \Lambda(q) w + g(q) + J^T F_e \tag{4.38}$$

with the control variable w being:

$$w = \ddot{X} \tag{4.39}$$

which is computed the following way:

$$w = a_d + A^{-1} \begin{bmatrix} w_p - f_e \\ w_o - \mu_e \end{bmatrix} \tag{4.40}$$

The A^{-1} is calculated the same way has equation 4.28 which means that, once again, there exists no inertia shaping, which in turn means that the F_e term is eliminated from equations 4.38 and 4.40, so no force feedback is needed [19]. In conclusion, the control variable w becomes:

$$w = a_d + A^{-1} \begin{bmatrix} K_p \Delta p_{cd} + D_p \Delta \dot{p}_{cd} \\ K_o \varepsilon_{cd} + D_o \Delta \omega_{cd} \end{bmatrix} \tag{4.41}$$

4.4 Hybrid Controllers

This next section is going to cover two ”hybrid” controllers. For reasons that are going to be explained later in chapter 5 it is necessary to separate the robot’s position and orientation control into two different controllers.

In order to do that, the robot’s position is going to use the impedance control with force sensing for redundant robots, since the robot’s DOF that are going to be controlled are only 3 (p_x, p_y, p_z). As for the robot’s orientation, the chosen controller is the joint space control with task space posture reference. With this, the orientation error in the task space are converted into velocity references for joint control [20].

In figure 4.7 it is shown the scheme for the ”hybrid” controller. During the development of this controller, special care must be taken so that the position parameters are not mixed with the orientation ones and vice versa. Also, the influence of each joint for the dynamics [$\hat{M}(q)$ and $\hat{g}(q)$] and forward kinematics calculation must be considered for both controls, so that its influence for each control is not neglected.

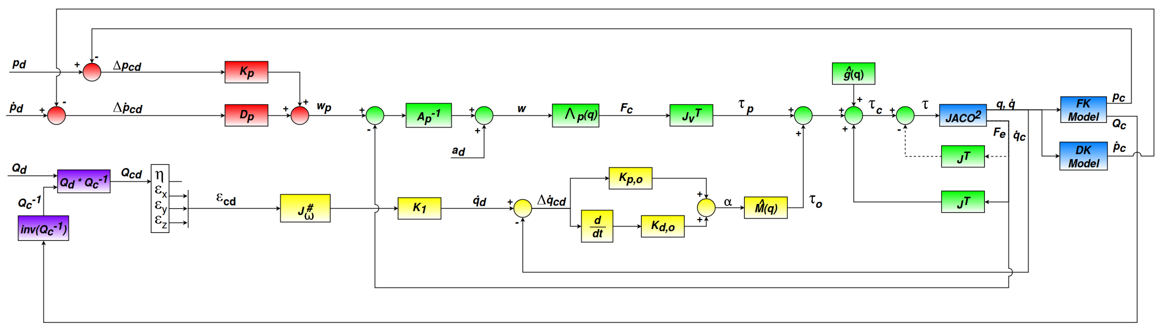


Figure 4.7: "Hybrid" controller scheme with impedance control with force sensing for redundant robots for position control and joint space torque control with task space posture reference for orientation control.

For this controller τ_c has the following expression:

$$\tau_c = \tau_p + \tau_o + \hat{g}(q) + J_v^T f_e \tag{4.42}$$

where:

- τ_p is a $n \times 1$ vector, n being the number of joints, with the computed torque for position control;
- τ_o is a $n \times 1$ vector, n being the number of joints, with the computed torque for orientation control;
- J_v is a $3 \times n$ matrix, n being the number of joints, representing the linear velocities Jacobian.

The following expression gives the control law for τ_p:

$$\tau_p = J_v^T \Lambda_p(q) w \tag{4.43}$$

where:

- $\Lambda_p(q)$ is 3×3 matrix, which represents the inertia in the task space for position control; this matrix is computed through the following equation:

$$\Lambda_p(q) = (J_v M(q)^{-1} J_v^T)^{-1} \tag{4.44}$$

The control variable w for this portion of the controller is, once again:

$$w = \ddot{X} \tag{4.45}$$

which it is computed through the following equation:

$$w = a_d + A_p^{-1}(w_p - f_e) \tag{4.46}$$

with:

$$w_p = K_p \Delta p_{cd} + D_p \Delta \dot{p}_{cd} \tag{4.47}$$

The A_p^{-1} represents the inverse mass gain parameter for position control and is calculated the following way:

$$A_p^{-1} = \Lambda_p(q)^{-1} \tag{4.48}$$

As before, there exists no inertia shaping, so the f_e term is eliminated from equations 4.42 and 4.46 which means that no force feedback is needed. Because of this, the control variable w becomes:

$$w = a_d + A_p^{-1}(K_p \Delta p_{cd} + D_p \Delta \dot{p}_{cd}) \tag{4.49}$$

As for the orientation control, it has the following control law:

$$\tau_o = \hat{M}(q)\alpha \tag{4.50}$$

with α being the control variable:

$$\alpha = \ddot{q} \tag{4.51}$$

The control variable α is going to be computed the following way:

$$\alpha = K_{p,o} \Delta \dot{q}_{cd} + K_{d,o} \frac{d\Delta \dot{q}_{cd}}{dt} \tag{4.52}$$

where:

- $\Delta \dot{q}_{cd}$ is a 3×1 vector with the error between the desired joint velocities ($\dot{q}_d$) and the current joint velocities ($\dot{q}_c$).

The current joint velocities ($\dot{q}_c$) are taken directly from the robot's sensors, in real time, while the desired joint velocities ($\dot{q}_d$) are computed through the following equation:

$$\dot{q}_d = K_1 J_\omega^{\#} \varepsilon_{cd} \tag{4.53}$$

where:

- K_1 is a $n \times n$ diagonal matrix, n being the number of joints, whose elements are set independently for each joint;
- J_ω is a $3 \times n$ matrix, n being the number of joints, representing the angular velocities Jacobian, the # symbol represents the pseudo-inverse, since this matrix is non-square so the inverse is impossible to compute.

The $J_\omega^\#$ matrix is computed the following way:

$$J_\omega^\# = M(q)^{-1} J_\omega^T \Lambda_o(q) \tag{4.54}$$

where:

- $\Lambda_o(q)$ is a 3×3 matrix, which represents the inertia in the task space for orientation control; this matrix is computed through the following equation:

$$\Lambda_o(q) = (J_\omega M(q)^{-1} J_\omega^T)^{-1} \tag{4.55}$$

4.4.1 Hybrid Controller with Null-Space

The last controller that it is going to be covered for this work is gonna be the same one as in section 4.4 except that the orientation control is going to be designed in null-space [21].

The computed torque control τ_c is going to be as follow:

$$\tau_c = \tau_p + \tau_{o|p} + \hat{g}(q) + J_v^T f_e \tag{4.56}$$

where:

- $\tau_{o|p}$ is the torque to perform orientation control constrained by position control.

In terms of position control, τ_p is computed the same way as equation 4.43 with the control variable w being calculated through equation 4.49, which means that no inertia shaping is considered so that no force feedback is needed. So, equation 4.56 becomes:

$$\tau_c = \tau_p + \tau_{o|p} + \hat{g}(q) \tag{4.57}$$

Next step, is going to be the formulation of $\tau_{o|p}$ which is the main difference between the previous controller (section 4.4). The manipulator dynamics with gravity in the joint space can be given by:

$$M(q)\ddot{q} = \tau_p + \tau_{o|p} \tag{4.58}$$

by setting $\tau_{o|p}$ as:

$$\tau_{o|p} = N_p^T M(q)\alpha \tag{4.59}$$

and multiplying both sides of equation 4.58 by $J_\omega^T \bar{J}_\omega^T$ it leads to:

$$J_\omega^T \bar{J}_\omega^T M(q)\ddot{q} = J_\omega^T \bar{J}_\omega^T \tau_p + J_\omega^T \bar{J}_\omega^T N_p^T M(q)\alpha \tag{4.60}$$

where:

- N_p is a $n \times n$ matrix, n being the number of joints, which represents the null-space projector of the robot's position;
- $\bar{J}_\omega$ is a $n \times 3$ matrix, n being the number of joints, which represents the dynamically consistent generalized inverse matrix of J_ω.

These two matrices are calculated through the following equations:

$$\begin{aligned} N_p &= I - \bar{J}_v J_v \\ \bar{J}_\omega &= M(q)^{-1} J_\omega^T \Lambda_o(q) \end{aligned} \tag{4.61}$$

where:

- I is a $n \times n$ identity matrix, n being the number of joints;
- $\bar{J}_v$ is a $n \times 3$, n being the number of joints, which represents the dynamically consistent generalized inverse matrix of J_v and it is computed in a similar way as $\bar{J}_\omega$ in equation 4.61.

Projecting equation 4.60 through N_p^T, the torques acting on the orientation control without affecting the position are given by:

$$N_p^T J_\omega^T \bar{J}_\omega^T M(q) \ddot{q} = N_p^T J_\omega^T \bar{J}_\omega^T \tau_p + N_p^T J_\omega^T \bar{J}_\omega^T N_p^T M(q) \alpha \tag{4.62}$$

Left multiplying both sides by $M(q)^{-1}$ and using the commutation properties:

$$\begin{aligned} M(q)^{-1} N_p^T &= N_p M(q)^{-1} \\ M(q)^{-1} J_\omega^T \bar{J}_\omega^T &= \bar{J}_\omega J_\omega M(q)^{-1} \end{aligned} \tag{4.63}$$

the corresponding joint acceleration for orientation control is given by:

$$N_p \bar{J}_\omega J_\omega \ddot{q} = N_p \bar{J}_\omega M(q)^{-1} \tau_p + N_p \bar{J}_\omega J_{\omega|v} \alpha \tag{4.64}$$

where:

- $J_{\omega|v}$ is a $3 \times n$ matrix, n being the number of joints, which combines the position control null-space operator with the angular Jacobian;

and it is computed the following way:

$$J_{\omega|v} = J_\omega N_p \tag{4.65}$$

Setting the control variable α as:

$$\alpha = \bar{J}_{\omega|v} J_\omega (\beta - M(q)^{-1} \tau_p) \tag{4.66}$$

and replacing equation 4.66 in equation 4.65, the orientation control closed-loop dynamics becomes:

$$N_p \bar{J}_\omega J_\omega (\ddot{q} - \beta) = 0 \tag{4.67}$$

where:

- β is a vector of size n, n being the number of joints, which corresponds to the new control variable;
- $\bar{J}_{\omega|v}$ is a $n \times 3$ matrix, n being the number of joints, which represents the dynamically consistent generalized inverse matrix of $J_{\omega|v}$.

A linear and decoupled joint acceleration in the position null-space shows up, performing the orientation control without inducing any motion in the position [21]. Plugging equation 4.66 in equation 4.59 and replacing $\bar{J}_{\omega|v}$ by:

$$\bar{J}_{\omega|v} = M(q)^{-1} J_{\omega|v} \Lambda_{o|p}(q) \tag{4.68}$$

$\tau_{o|p}$ becomes:

$$\tau_{o|p} = J^T_{\omega|v} \Lambda_{o|p}(q) J_\omega (\beta - M(q)^{-1} \tau_p) \tag{4.69}$$

where the null-space idempotent property $N_p^T = (N_p^T)^2$ has been used and $\Lambda_{o|p}(q)$ is given by:

$$\Lambda_{o|p}(q) = (J_\omega M(q)^{-1} J^T_{\omega|p})^{-1} \tag{4.70}$$

Substituting equations 4.69 and 4.43 in equation 4.57, the computed torque control to perform both controls is given by:

$$\tau_c = J_v^T \Lambda_p(q) w + J^T_{\omega|v} \Lambda_{o|p}(q) J_\omega (\beta - M(q)^{-1} \tau_p) + \hat{g}(q) \tag{4.71}$$

In the above formulation, there are present algorithmic singularities when both controls are in conflict, which causes $J^T_{\omega|v}$ to drop rank [21]. To correct this issues with such singularities, α is going to be computed the following by:

$$\alpha = \bar{J}_\omega J_\omega (\beta - M(q)^{-1} \tau_p) \tag{4.72}$$

where, the only difference from equation 4.66 is the replacement of term $\bar{J}_{\omega|v}$ for $\bar{J}_\omega$. Since J_ω is the angular Jacobian without any constraints from the force controller. Due to the manipulator structure, J_ω only loses rank at the external boundary of its reachable workspace, having no ill-condition issues near internal singular conditions, when both controls are in conflict [21]. With this approach and using the following commutation property:

$$N_p^T M(q) = M(q) N_p \tag{4.73}$$

equation 4.71 becomes:

$$\tau_c = J_v^T \Lambda_p(q) w + J_{\omega|v}^T \bar{J}_\omega^T (M(q)\beta - \tau_p) + \hat{g}(q) \tag{4.74}$$

The problem of this approach is that the closed-loop system found in equation 4.67 is no longer achieved. For this reason, tracking errors might arise in the orientation control, since the forces given by $\bar{J}_\omega[M(q)\beta - \tau_p]$ might have components that are influenced by the position control, which are minimized by N_p.

Still, in order to do the polishing task, the tracking of orientation control is not very important, however it is relevant that it stays almost constant throughout the task, so this controller is the most suitable one to do it.

Chapter 5. Experimental Results

The first section of this chapter is going to be focused on the experimental results from the control architectures described in chapter 4 taken from the Gazebo simulator, while the second section is going to be from the real physical robot.

Before starting the experimental results, it is going to be given a brief overview of the project's organization. A package called *kinova_spacenav* is created, which contains many folders and scripts. Two of this folders, **src** and **include**, contain the source code and header files, respectively, for classes **jaco_kdl** and **jaco_spacenav**, which were covered briefly in sections 2.4 and 3.3.

There is also a folder called **launch** which contains the launcher files in order to launch a server with all the ROS topics, among other things, which can be accessed during implementation of the controllers. It is created launch files for the simulator and robot, all based on what was provided by the *kinova_ros* stack (section 2.3).

5.1 Gazebo Simulator

In the *kinova_spacenav* a folder called **controllers_simulator** contains the algorithms developed for the controllers mentioned in chapter 4 for use in the simulator. The basis behind each controller is that, before beginning the control loop there is a switch between position and effort control, so that each $1ms$ torques are computed by the controller and sent to robot.

5.1.1 Computed torque control in the joint space (simulation)

For this controller the SpaceMouse is not used, so to study its performance and stability arbitrary trajectories are given to study it. The robot is going to execute a trajectory based on cubic polynomial between $t_i = 5s$ and $t_f = 15s$, t being the simulation elapsed time. The robot's initial and final joint positions (q_i and q_f) are as follows:

$$\begin{gathered} q_i = \begin{bmatrix} 4.8046852 & 2.924825 & 1.002 & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T \\ q_f = \begin{bmatrix} \pi & \pi & \pi & \pi & \pi & \pi \end{bmatrix}^T \end{gathered} \tag{5.1}$$

The polynomial trajectory uses the following equation:

$$q_d = q_i + \frac{3(q_f - q_i)(t - t_i)^2}{(t_f - t_i)^2} - \frac{2(q_f - q_i)(t - t_i)^3}{(t_f - t_i)^3} \tag{5.2}$$

with q_d being the desired joint positions.

From $t_i = 20s$ until $t_f = 80s$ the robot is going to perform a sine wave with the following amplitude (A) and period (T):

$$\begin{gathered} A = \begin{bmatrix} \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} & \frac{\pi}{4} \end{bmatrix}^T \\ T = 10s \end{gathered} \tag{5.3}$$

The sine wave has the following equation:

$$q_d = q_i + Asin\left(\frac{2\pi t}{T}\right) \tag{5.4}$$

Finally, between $t_i = 85s$ and $t_f = 95s$ the robot returns to its *home* position (q_i in equation 5.1), following the same cubic polynomial equation as 5.2.

As it was mentioned in section 4.1 the proportional (K_p) and derivative (K_d) gains were computed by following equation 4.7 while the integrative gains (K_i) was introduced manually. The following table shows the natural frequency (ω_n) and K_i values that best suited this controller:

Table 5.1: Table with the gains used for this controller.

Joint j	ω_{nj}	K_{ij}
1	35	0.005
2	10	0.005
3	15	0.005
4	35	0.015
5	30	0.005
6	40	0.025

In figure 5.1 it is shown the results of the procedure described during this section with the gains used in table 5.1.

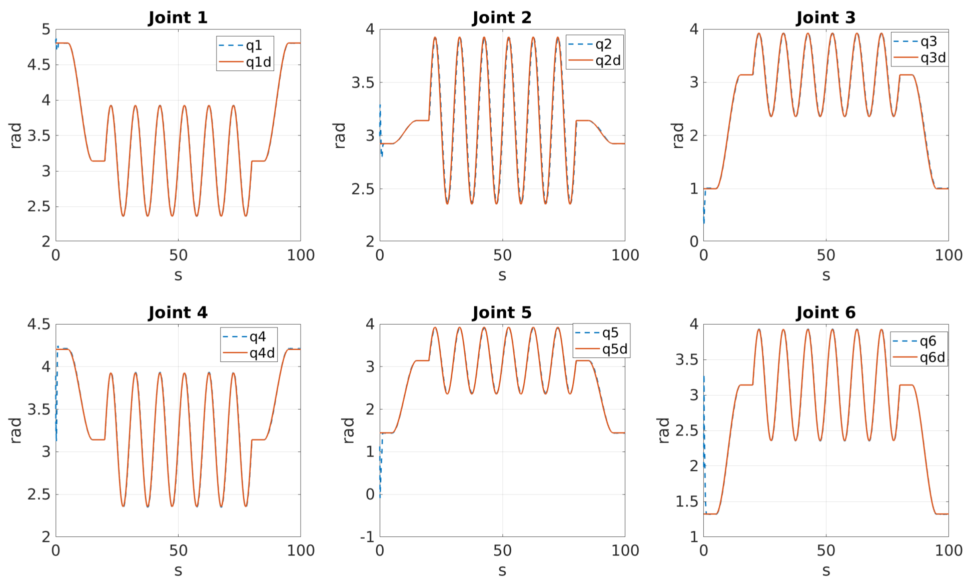


Figure 5.1: PID computed torque control in the joint space (simulation).

It is important to underline that this controller achieves a good joint position tracking, a small steady-state error and also the small oscillation in the beginning of the controller. This is due to the switch between position and effort control, which introduces a slight delay before the controller is activated. This is more noticeable in the simulator, in the robot (section 5.2) is less noticeable and sometimes it does not even happen.

5.1.2 Computed torque control in the task space (simulation)

Starting from this controller, all the experiments will be made by using the SpaceMouse, instead of giving an arbitrary trajectory to the robot. This is due to the fact that the SpaceMouse can control the robot in its 6 DOF and also because one of the objectives of this work is to introduce telemanipulation to the polishing task.

So, in terms of experiment, it will be performed movements in all 6 DOF so that the stability and performance of the controller can be evaluated. This is going to be true to all the controllers hereafter. The gains for this controller are shown in the following table:

Table 5.2: Table with the gains used for this controller.

DOF	K_p	K_d
p_x	300	50
p_y	300	50
p_z	450	50
o_x	350	20
o_y	350	20
o_z	350	20

In figure 5.2 it is shown the results of the experiment described with the gains shown in table 5.2. The orientation results are presented in euler angles with the Yaw and Pitch wrapped between $[-\pi, \pi]$ and the Roll between $[0, 2\pi]$, although the control was made with quaternions, this happens to all task space controllers.

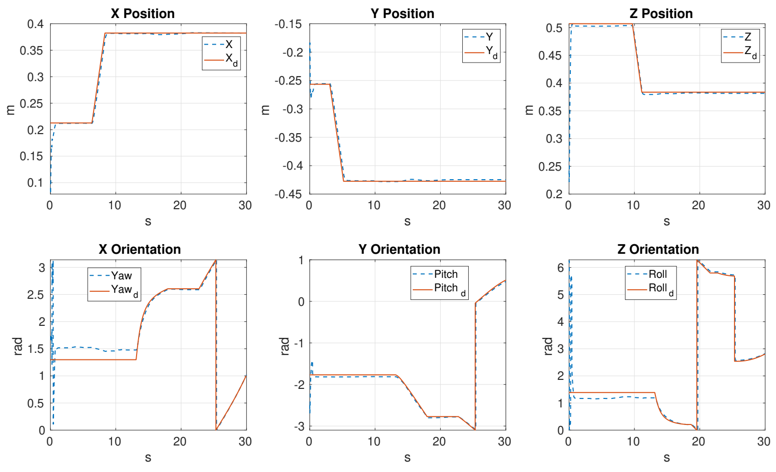


Figure 5.2: PD computed torque control in the task space (simulation).

It can be seen that, once again, the robot achieves a very good tracking and a small steady-state error, but with the absent of an integrative part the error is not being corrected as time goes by.

5.1.3 Impedance Control without force sensing and without inertia shaping (simulation)

Once again, the experiment in this controller is going to be made by evaluating the robot's performance and stability in all its DOF with values taken from the SpaceMouse.

During the development of this controller it is necessary to enhance virtually the inertia tensor in the last joint, which in turn influences the estimated mass matrix $[\hat{M}(q)]$, because the values returned are too small. This is due to the fact that the actuators friction is more dominant in the last 3 joints on the opposite of what happens in the first 3, where the mass is the most influential. The gains utilized for this controller are as follow:

Table 5.3: Table with the gains used for this controller.

DOF	K	D
p_x	100	20
p_y	100	20
p_z	200	20
o_x	20	2
o_y	20	2
o_z	20	2

In figure 5.3 represents the results for this controller with the gains shown in table 5.3. The results prove that the controller has a good tracking and small steady-state error.

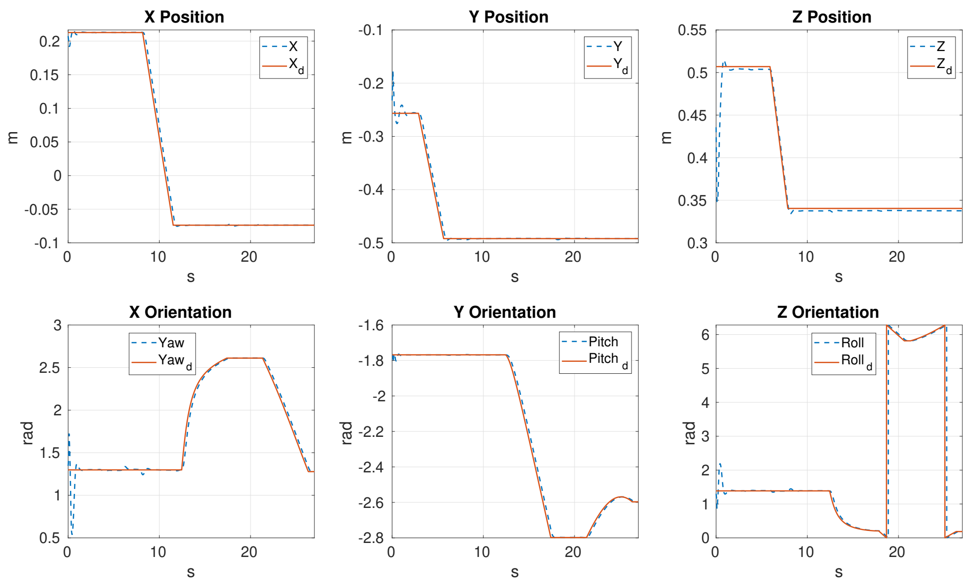


Figure 5.3: Impedance Control without force sensing and without inertia shaping (simulation).

5.1.4 Impedance Control with force sensing (simulation)

A similar procedure was done to test the performance and stability of this controller as it was used in the previous ones. The following table shows the gains used:

Table 5.4: Table with the gains used for this controller.

DOF	K	D
p_x	200	24
p_y	200	24
p_z	350	24
o_x	180	3
o_y	180	3
o_z	180	3

With the gains in table 5.4, the results shown in figure 5.4 were obtained. It can be seen that the steady-state error is even smaller with this controller, while the tracking stays very good.

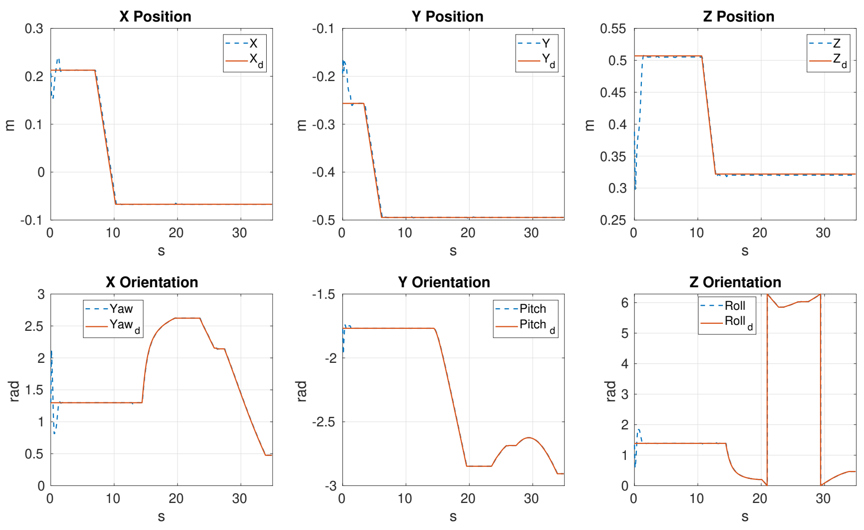


Figure 5.4: Impedance Control with force sensing (simulation).

5.1.5 Impedance Control with force sensing for redundant robots (simulation)

With a similar procedure described in the previous controllers a study was made for the stability and performance of this controller. The following table shows the gains used:

Table 5.5: Table with the gains used for this controller.

DOF	K	D
p_x	200	24
p_y	200	24
p_z	400	24
o_x	180	3.5
o_y	180	3.5
o_z	180	3.5

With the gains in table 5.5 the results in figure 5.5 were obtained. Very similar, to the previous controller, the steady-state error continues to be smaller, compared to the others and it achieves a very good tracking.

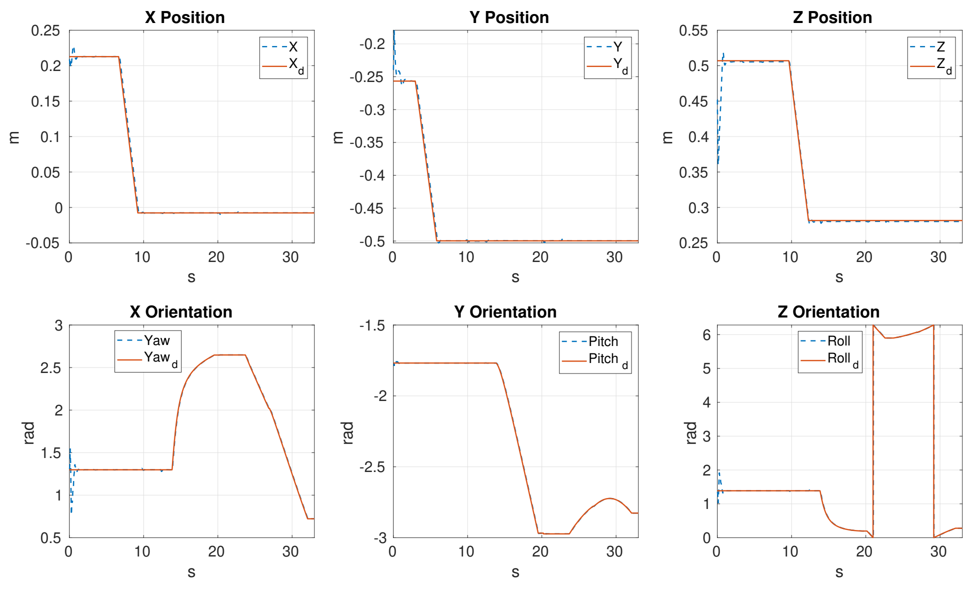


Figure 5.5: Impedance Control with force sensing for redundant robots (simulation).

5.1.6 Hybrid Controller (simulation)

A similar task was done to test this controller performance and stability. The following tables show the gains used:

Table 5.6: Tables with the gains used for this controller.

(a) Table with gains used for position control.

DOF	K	D
p_x	600	22
p_y	600	22
p_z	600	22

(b) Table with gains used for orientation control.

Joint j	K_{1j}	K_{pj}	K_{dj}
1	5	10	0.02
2	5	10	0.02
3	5	10	0.02
4	25	10	0.02
5	25	10	0.02
6	25	10	0.02

With the gains displayed in tables 5.6 the results shown in figure 5.6. Once again, the robot achieves a very good tracking and small steady-state error. Sometimes it can be seen that there is a little oscillation in all DOF which is going to be mitigated with the introduction of null-space in the end-effector's position.

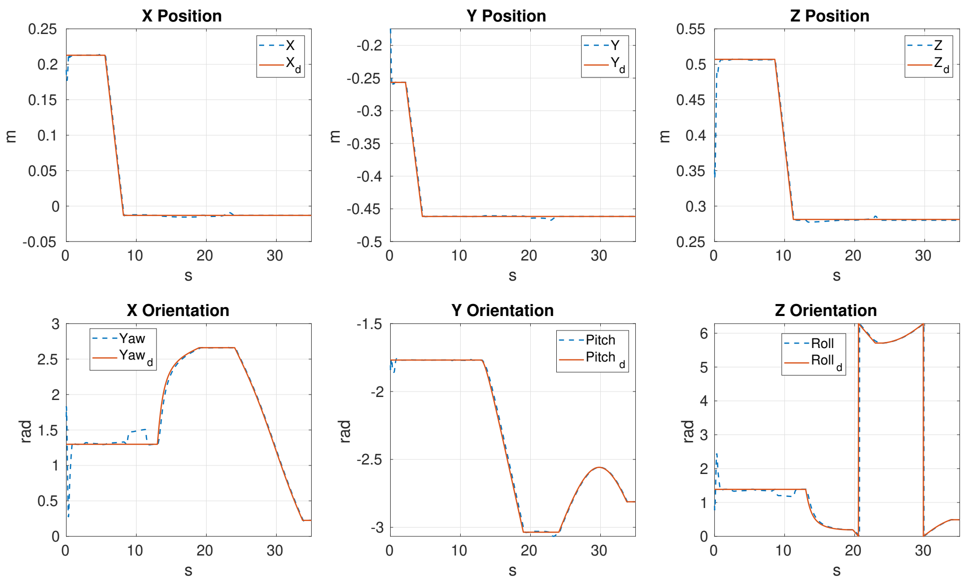


Figure 5.6: "Hybrid" Controller (simulation).

5.1.7 Hybrid Controller with Null-Space (simulation)

With a similar procedure as the previous controllers and with the following tables a study of the robot's stability and performance was made for this controller:

Table 5.7: Tables with the gains used for this controller.

(a) Table with gains used for position control.

DOF	K	D
p_x	200	22
p_y	200	22
p_z	400	22

(b) Table with gains used for orientation control.

Joint j	K_{1j}	K_{pj}	K_{dj}
1	5	10	0
2	5	10	0
3	5	10	0
4	30	10	0
5	30	10	0
6	30	10	0

With the gains shown in tables 5.7 the results shown in figure 5.7 were obtained. Once again, the robot achieves a good tracking and a small steady-state error. More noticeable is the absence of the little oscillations that were present in the previous controller results (section 5.6).

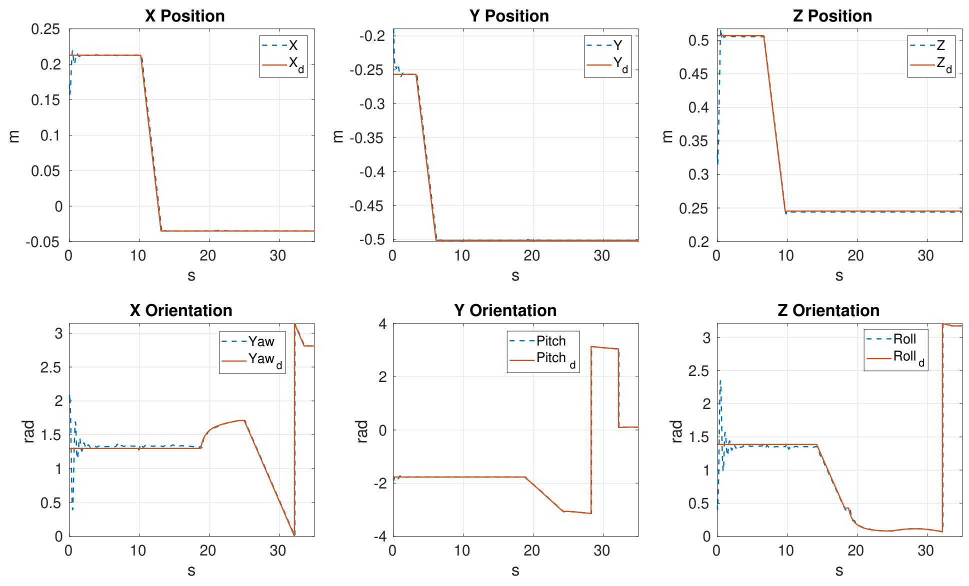


Figure 5.7: ”Hybrid” Controller with Null Space (simulation).

5.2 Real Kinova JACO^2 Robot

In the *kinova_spacenav* package there is a folder called **controllers_robot** that contains the algorithms developed for the controllers described in chapter 4 for use on the robot.

Before making the experimental tests in the robot some preparation has to be done. In the *kinova_driver* package is created a node called **kinova_api_funcs**. In this node some API functions are accessed to change some of the robot's parameters, namely:

- **SetTorqueActuatorGain**: this functions sets the actuators feedback gain, which is changed all to 0.0 for all actuators;
- **SetTorqueSafetyFactor**: this function is a safety feature used to specify the velocity threshold over which the robot will stop and return to position control, this is a safety feature to prevent the arm from performing undesired and/or dangerous motions, the value set is 1.0.
- **SetTorqueVibrationController**: this function is used to set the vibration observer/controller level, this is important for when the robot is in contact with a stiff environment which is the case for the polishing task (chapter 6), the value set is 1.0 which corresponds to its maximal level;
- **SetGravityOptimalZParam**: this function is used to set the gravity parameters using the optimal mode, these values are found by running the function **RunGravityZEstimationSequence** which is done when the robot has the standard end-effector and the polishing tool (figures 2.1b and 6.8a);
- **SetGravityType**: this function sets the gravity type to MANUAL_INPUT or OPTIMAL, the choice is the latter in concordance to the previous point.

Also, a python script called **calibration_robot** found in the *kinova_spacenav* package is developed to calibrate the robot's torques, this can also be done through the Kinova SDK via the ”Torque Zero” found in the ”Advanced Menu”. This has to be done sometimes when the user wants to work in effort control. For the calibration to be successful the robot's joint positions have to be as follows:

$$q = \begin{bmatrix} \pi & \pi & \pi & \pi & \pi & \pi \end{bmatrix} \tag{5.5}$$

5.2.1 Computed torque control in the joint space (robot)

Like in the simulator, in this control the SpaceMouse is not used and instead a trajectory is defined for the robot to execute in order to study its performance and stability.

Between $t_i = 5s$ and $t_f = 15s$ the robot is going to execute a cubic polynomial from the following initial and final joint positions:

$$\begin{aligned} q_i &= \begin{bmatrix} 4.8046852 & 2.924825 & 1.002 & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T \\ q_f &= \begin{bmatrix} 4.8046852 & \pi & \frac{\pi}{2} & 4.2031852 & 1.4458 & 1.3233 \end{bmatrix}^T \end{aligned} \tag{5.6}$$

The cubic polynomial is defined by equation 5.2. From $t_i = 20s$ until $t_f = 80s$ the robot is going to perform a sine wave, that is defined by equation 5.4, with the following parameters:

$$\begin{aligned} A &= \begin{bmatrix} \frac{\pi}{6} & \frac{\pi}{20} & \frac{\pi}{20} & \frac{\pi}{20} & \frac{\pi}{20} & \frac{\pi}{6} \end{bmatrix}^T \\ T &= 10s \end{aligned} \tag{5.7}$$

Finally, between $t_i = 85s$ and $t_f = 95s$ the robot is going to perform a cubic polynomial, defined by equation 5.2, from it's current joint positions until the *home* position (the first vector in equation 5.6).

For this experiment the mass matrix in the last 3 joints are not considered because the gains that had to be given are too large for the joints to execute any kind of movement. Instead, only a PD controller is going to be considered for this joints. Differently from what was shown in the simulator, the K_p and K_d gains for each joint are given manually and the K_i is completely removed. With that, the gains used for this control are shown in the following table:

Table 5.8: Table with the gains used for this controller.

Joint j	K_{pj}	K_{dj}
1	200	20
2	300	20
3	300	20
4	30	5
5	30	5
6	30	5

With the gains presented in table 5.8 the results shown in figure 5.8 were obtained. First of all, it can be seen that the steady-state error and performance of this control in the robot is not as good as in the simulator, which is to be expected, since in the simulation environment is mostly designed to achieve almost ”perfect” results. Still, the steady-state error is not very significant and the robot achieves a very good tracking. Also to note that the switch between position and effort control is not as noticeable as in the simulation, this is due to the fact that the robot already has a gravity vector implemented inside.

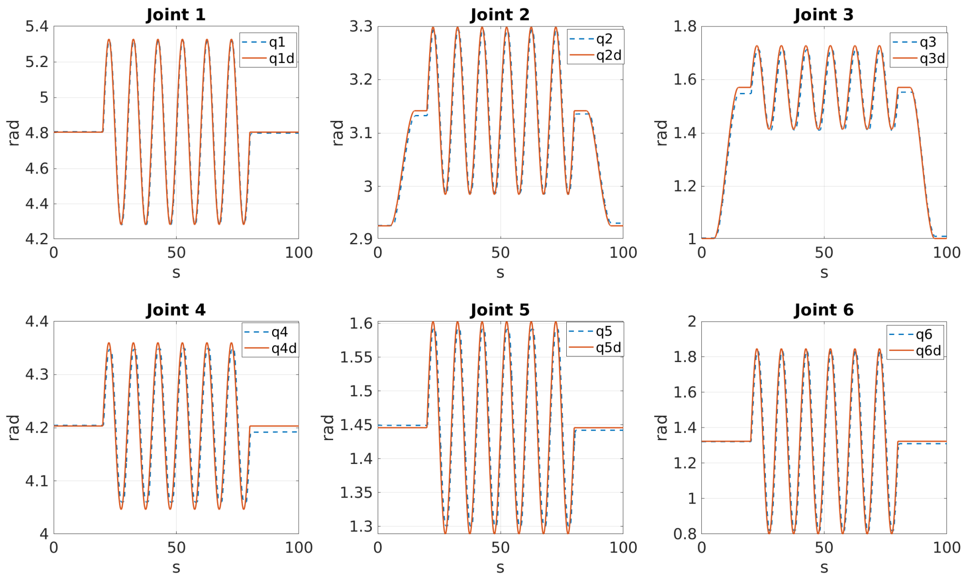


Figure 5.8: PD computed torque control in the joint space (robot).

5.2.2 Computed torque control in the task space (robot)

For this control a similar procedure is used as it was in the simulator (section 5.1.2). It is important to underline that the experiment to test the performance and stability of the control is done by using the SpaceMouse in all 6 of the robot’s DOF.

The main difference is that in the robot the inertia tensor that was used in all the controllers after the first impedance control (section 5.1.3) was also used for this one, so as to achieve better results. The following table shows the gains used:

Table 5.9: Table with the gains used for this controller.

DOF	K_p	K_d
p_x	200	24
p_y	200	24
p_z	200	24
o_x	150	6
o_y	150	6
o_z	150	6

With the gains in table 5.9 the results in figure 5.9 are obtained. Just to remind that the Yaw and Pitch angles are wrapped between $[-\pi,\pi]$ and the Roll between $[0,2\pi]$.

To note that this control also achieves a good tracking even though the steady-state error is not as good as in the simulation and there some coupling problems with the position control, which means that when the operator is controlling the robot in orientation, it will affect the its position and vice versa. This problem is more significant in the robot's X and Z orientation and also in Y position.

Because of the robot's last 3 joints the dominant physical aspect is the actuators frictions instead of their mass, which is more relevant in the first 3. This is the reason why it is opted to separate the position from the orientation control, thus the existence of the "hybrid" controllers (section 4.4). Still, the robot achieves an acceptable steady-state error and the robot is capable of correcting its pose, derived from the coupling problems.

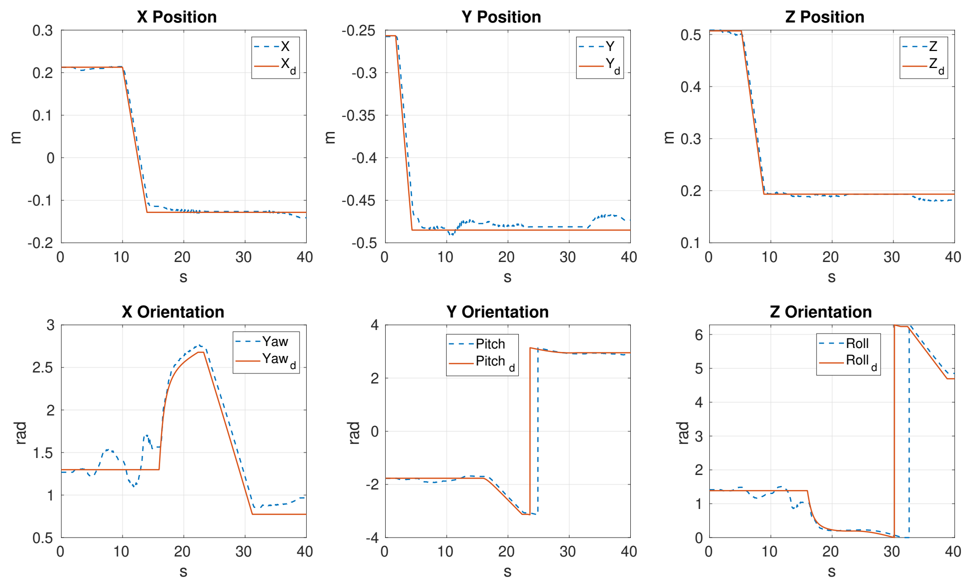


Figure 5.9: PD computed torque control in the task space (robot).

5.2.3 Impedance Control without force sensing and without inertia shaping (robot)

For this control a similar experiment is made in order to study the robot's performance and stability. The gains used for this control are as follow:

Table 5.10: Table with the gains used for this controller.

DOF	K	D
p_x	200	10
p_y	200	10
p_z	200	10
o_x	25	1.5
o_y	25	1.5
o_z	25	1.5

With the gains in table 5.10 the results shown in figure 5.10 are obtained. Once again, the performance is not as good as in the simulation environment and there are still coupling problems between the position and orientation controls, more noticeable in the X and Z orientation and Y position. However, the steady-state error continues to be small and the robot is capable of correcting its posture to match the reference, mostly, derived from the coupling problems.

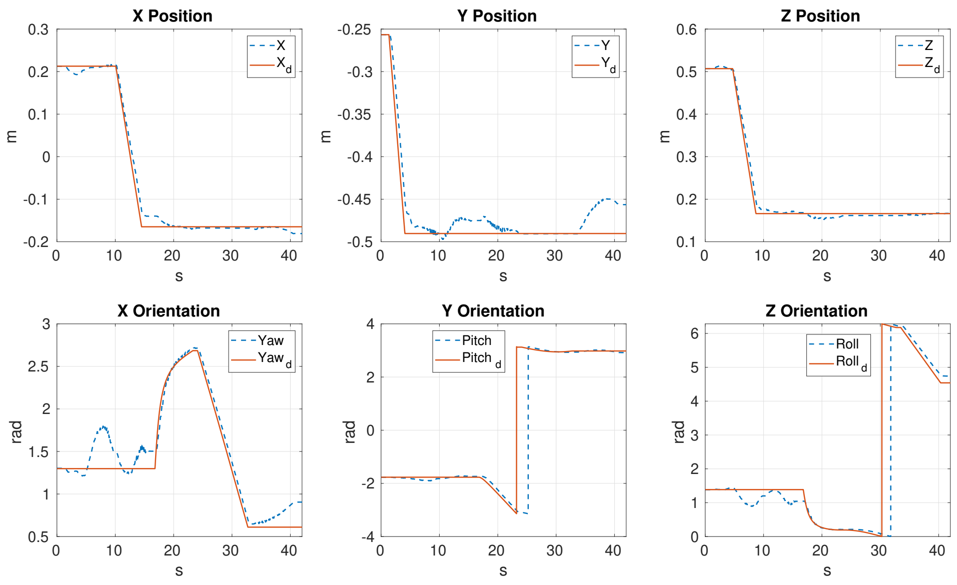


Figure 5.10: Impedance Control without force sensing and without inertia shaping (robot).

5.2.4 Impedance Control with force sensing (robot)

For this control the same procedure is made to test the robot's stability and performance. The gains utilized for this control are shown in the following table:

Table 5.11: Table with the gains used for this controller.

DOF	K	D
p_x	150	22
p_y	150	22
p_z	150	22
o_x	15	0.1
o_y	15	0.1
o_z	15	0.1

With the gains in table 5.11 the results shown in figure 5.11 are obtained. The steady-state error continues to achieve an acceptable value and the coupling problems are a little mitigated in some portions of the experiment, especially in Y position but still noticeable in X and Z orientation.

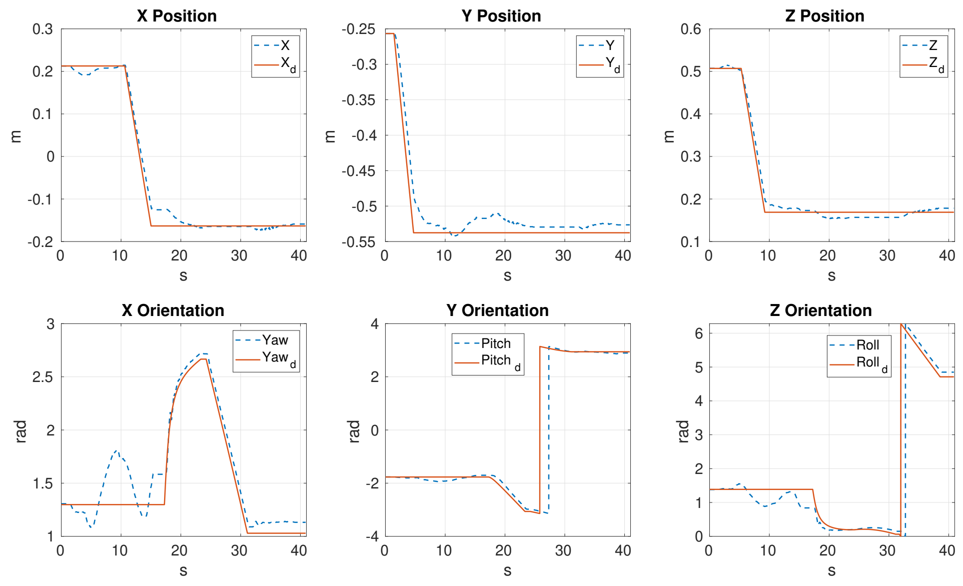


Figure 5.11: Impedance Control with force sensing (robot).

5.2.5 Impedance Control with force sensing for redundant robots (robot)

In this control a similar task is given to the robot in order to study the its stability and performance. The following table shows the gains used for this controller:

Table 5.12: Table with the gains used for this controller.

DOF	K	D
p_x	150	22
p_y	150	22
p_z	150	22
o_x	20	0.1
o_y	20	0.1
o_z	20	0.1

With the gains shown in table 5.12 the results shown in figure 5.12 are obtained. The coupling problems are still noticeable like in the previous control (section 5.2.4), but the robot is still capable of correcting its pose to achieve the reference, while the steady-state error continues to have a small acceptable value.

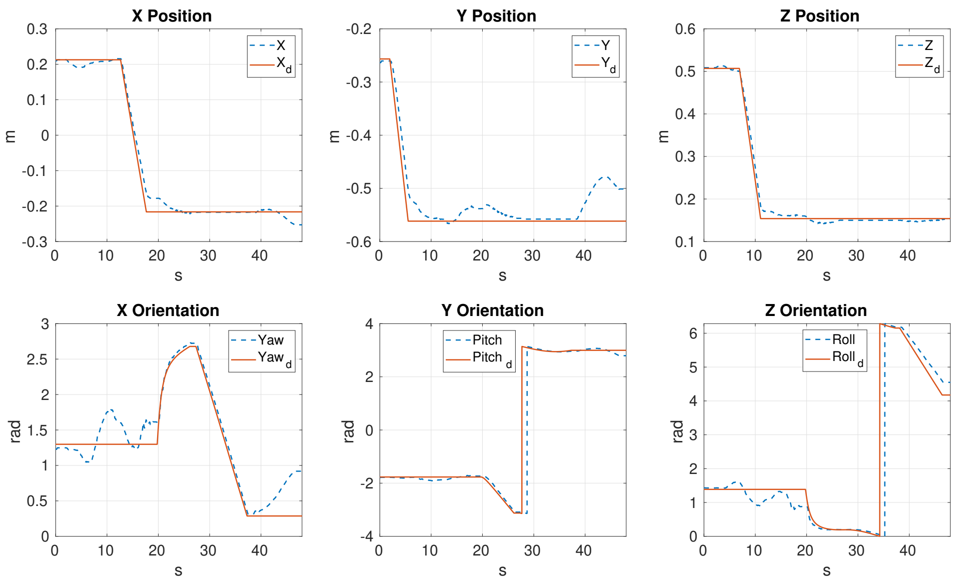


Figure 5.12: Impedance Control with force sensing for redundant robots (robot).

5.2.6 Hybrid Controller (robot)

A similar procedure is taken to study the performance and stability of this controller. The gains used in this controller are shown in the following table:

Table 5.13: Tables with the gains used for this controller.

(a) Table with gains used for position control.

DOF	K	D
p_x	500	10
p_y	500	10
p_z	500	10

(b) Table with gains used for orientation control.

Joint j	K_{1j}	K_{pj}	K_{dj}
1	5	10	0.3
2	5	10	0.3
3	5	10	0.3
4	25	10	0.3
5	25	10	0.3
6	25	10	0.3

With the gains shown in table 5.13 the results shown in figure 5.13 are obtained. The steady-state error is even smaller than in this previous controls and the coupling problems are mitigated in all of the components previously mentioned, proving that by controlling the position and orientation with different controllers it is possible to achieve a better stability.

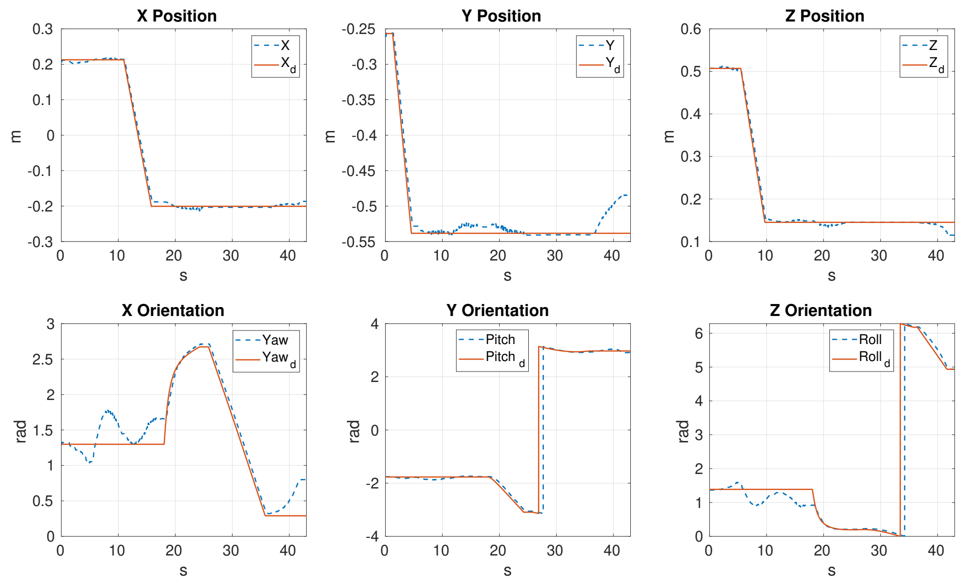


Figure 5.13: ”Hybrid” Controller (robot).

5.2.7 Hybrid Controller with Null-Space (robot)

For this control a similar procedure is executed to study the robot’s stability and performance. The gains used are shown in the following table:

Table 5.14: Tables with the gains used for this controller.

(a) Table with gains used for position control.

DOF	K	D
p_x	150	10
p_y	150	10
p_z	150	10

(b) Table with gains used for orientation control.

Joint j	K_{1j}	K_{pj}	K_{dj}
1	5	7.5	0
2	5	7.5	0
3	5	7.5	0
4	25	7.5	0
5	25	7.5	0
6	25	7.5	0

With the gains shown in table 5.14 the results shown in figure 5.14 are obtained. It can be seen that the coupling problems are less noticeable in the components that were more relevant previously and steady-state error is even smaller.

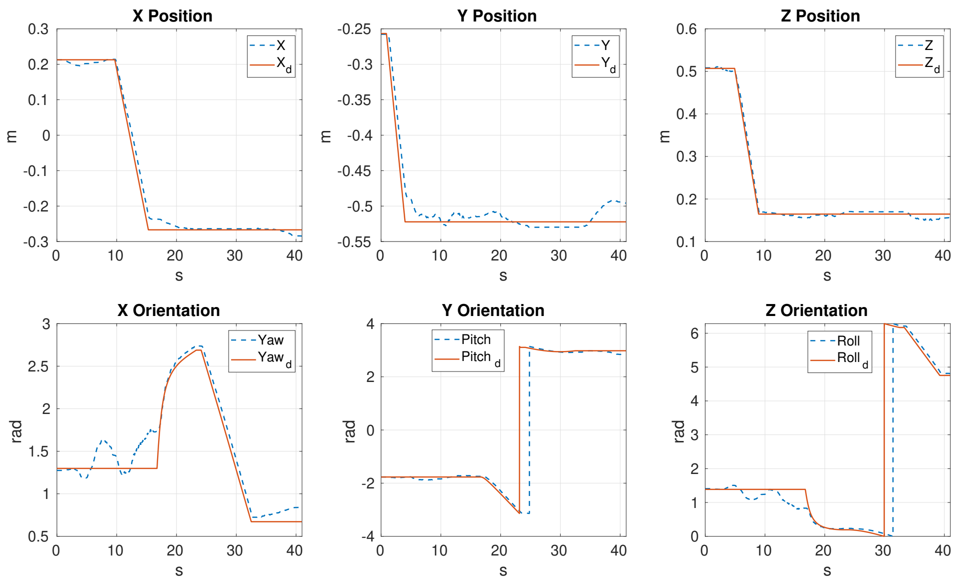


Figure 5.14: ”Hybrid” Controller with Null Space (robot).

Although, not ”perfect” it is the best control that was developed during this work, so it is the one that is going to be used for the polishing task.

Chapter 6. Polishing Task

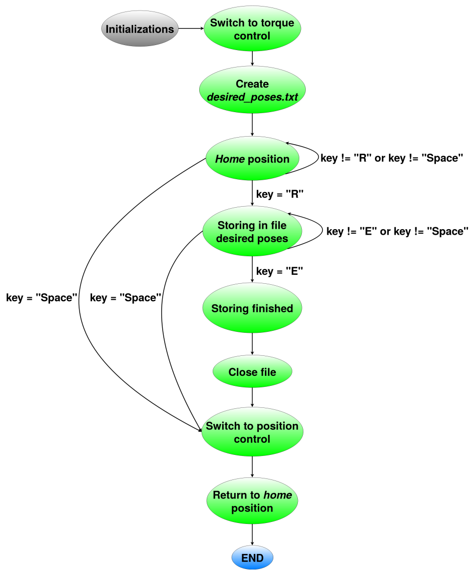
This chapter is going to start with a brief explanation about the algorithm that is developed for the robot to achieve the polishing task. Then, there will be some tests in both the simulation and the real robot environment.

6.1 Algorithm

To start off, the algorithms for the polishing task in the simulator and the real robot are identical. To achieve the polishing task there are actually two algorithms developed, one of them is to ”train” the robot to achieve a certain trajectory and other one is for it to replicate the movements previously executed with the first algorithm. This is useful because the operator can teach the robot to polish an arbitrary surface only once and then it can replicate the same trajectory to an identical surface.

In the *train* algorithm the robot initializes in torque control with a control loop that stabilizes the robot in its *home* position. When the user is ready he/she can press the ”R” key in their personal computer’s keyboard to start ”recording” the robot’s trajectory. When the ”R” key is pressed the user then uses the SpaceMouse to move the robot and at the same time its current poses (p_x, p_y, p_z, ε_x, ε_y, ε_z and η) are stored in a file called *desired_poses.txt*. When the user is satisfied, he/she can press the ”E” key to close the file and turn off the controller. The robot then changes to position control and returns to its *home* position. If the user is not satisfied or something goes wrong with the robot he/she can cancel anytime by pressing the ”Space” key.

The *execute* algorithm opens the file previously created by the *train* algorithm and stores all its poses in a dynamic vector which are going to be the desired poses in the control loop. Then, the robot enters the control loop where it stays in the *home* position until the user presses the ”Enter” key and the robot initializes the trajectory previously trained. As it was in the *train* algorithm, the user can interrupt anytime he/she wants by pressing the ”Space” key. The figures 6.1a and 6.1b illustrate the flow charts for both the *train* and *execute* algorithm.



(a) *Train* flow chart.

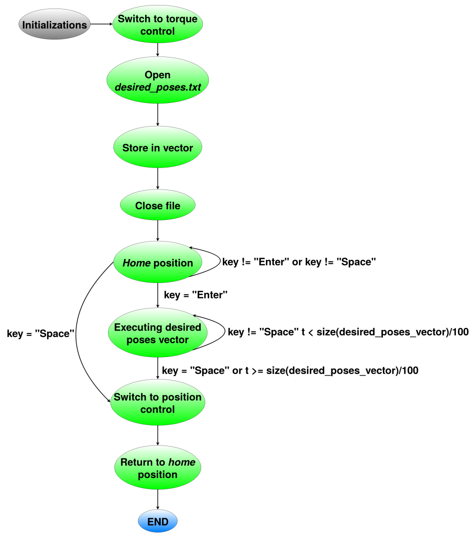
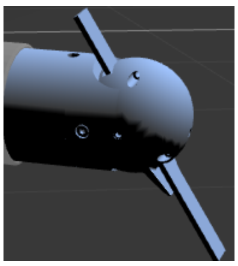

Initializations
Switch to torque control
Open desired_poses.txt
Store in vector
Close file
Home position
key != "Enter" or key != "Space"
key = "Enter"
Executing desired poses vector
key != "Space" t < size(desired_poses_vector)/100
key = "Space"
key = "Space" or t >= size(desired_poses_vector)/100
Switch to position control
Return to home position
END
(b) *Execute* flow chart.

Figure 6.1: Polishing task flow charts.

6.2 Polishing Task in Gazebo Simulator

To prove the concept and test the polishing task an environment is going to be developed. First off, the robot's end-effector is going to be replaced by a polishing tool and a steel mold is going to be added to which its surface is where the tests are going to be executed, both in gazebo and in the real robot. Also, the same environment is shown in Rviz that permits the use of markers which can be activated to show the areas in the mold that were polished. Figures 6.2a and 6.2b demonstrate the polishing tool and steel mold in Gazebo and in figures 6.3a and 6.3b both of these models are represented in Gazebo and Rviz environment, respectively.


(a) Polishing tool in Gazebo.

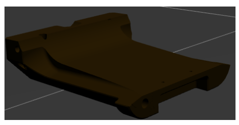
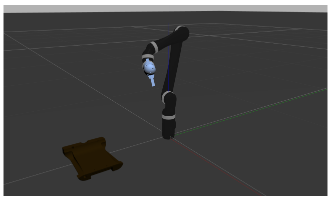
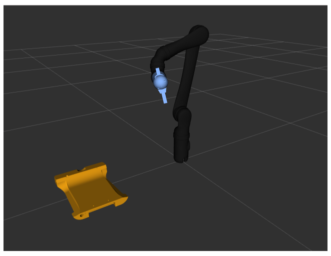

(b) Steel mold in Gazebo.

Figure 6.2: Polishing tool and steel mold in Gazebo.



(a) Polishing task environment in Gazebo.



(b) Polishing task environment in Rviz.

Figure 6.3: Polishing task environment in both simulators.

The controller used in simulation is the impedance control with force sensing (section 4.3.2). Also, the end-effector's axis is changed to match the tip of the tool. The critical point for the polishing task is when the tool comes in contact with the surface which is shown in figure 6.4.

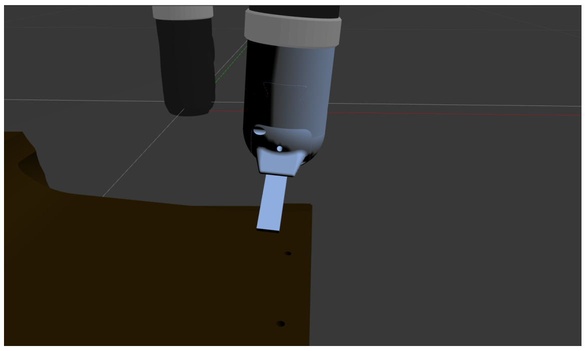


Figure 6.4: Polishing tool comes into contact with the steel mold, the critical point of the polishing task (simulation).

For the polishing task in the simulator the following gains are used:

Table 6.1: Table with the gains used the polishing task in simulation.

DOF	K	D
p_x	200	24
p_y	200	24
p_z	200	24
o_x	9.5	1.2
o_y	9.5	1.2
o_z	9.5	1.2

With the gains in 6.1 the results shown in figures 6.5 and 6.6 are obtained. It is important to recall that the Yaw and Pitch angles are wrapped between $[-\pi, \pi]$, while the Roll angle between $[0, 2\pi]$.

Firstly, it can be seen that the robot's performance and stability stays very acceptable throughout the whole process.

In time interval between $t = 20s$ and $t = 40s$ in figure 6.5 it can be seen that the reference in the Z position is lower than the robot's position, that's when the polishing tool is in contact and it stays this way throughout the experiment. The polishing task is performed in the Y axis, this can be seen because its position is going back and forth while the X axis is only used to move to a new section of the surface in order to continue the polishing. The robot's orientation stays almost the same throughout the task, this is important so that the angle of the tool does not change too much during the process.

Finally, in figure 6.6 to note the sudden changes in the end-effector's and contact force in the Y axis during the polishing task which is where it is more relevant.

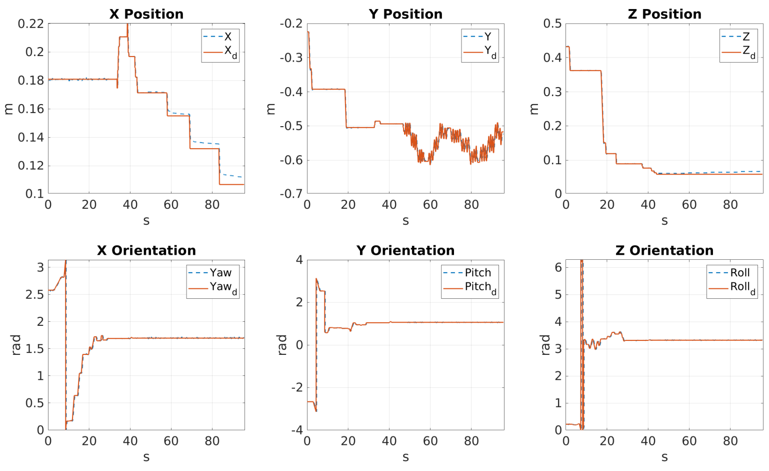


Figure 6.5: Polishing task pose results (simulation).

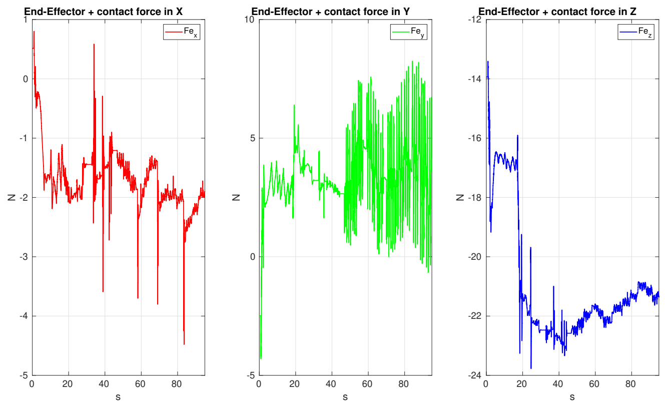


Figure 6.6: Polishing task end-effector plus contact force (simulation).

In figure 6.7 it can be seen which area of the mold was actually polished during the task by recurring to the Rviz visualization markers.

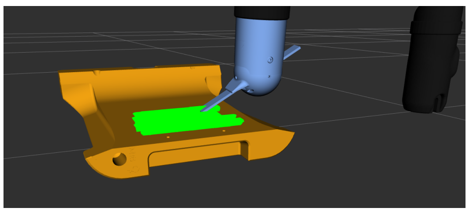
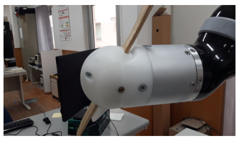


Figure 6.7: Area polished as shown in Rviz.

6.3 Polishing Task in Real JACO² Robot

A similar setup is going to be made to prove the concept with the real robot. Figures 6.8a and 6.8b contain the real polishing tool and steel mold, respectively, which are going to be used to build the setup for the polishing task in the real world. In figure 6.9 it is shown the complete setup for the task.


(a) Polishing tool in the real world.

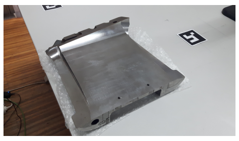

(b) Steel mold in the real world.

Figure 6.8: Polishing tool and steel mold in the real world.

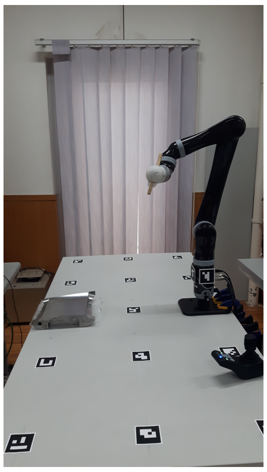


Figure 6.9: Polishing task setup.

Unlike in the simulator, the controller that is used for the polishing task is the ”hybrid” controller with null-space (section 4.4.1) because it was the one studied that had the best performance and results (section 5.2.7). Once again, the critical point of the polishing task is when the tool comes into contact with the surface, that is shown in figure 6.10.

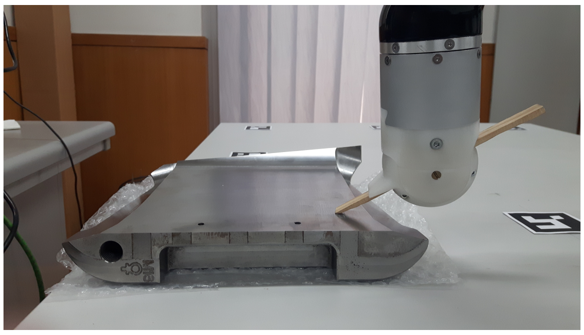


Figure 6.10: Polishing tool comes into contact with the steel mold, the critical point of the polishing task (robot).

For the polishing task in the robot the gains that are shown in the following table are used:

Table 6.2: Tables with the gains used for the polishing task in the robot.

(a) Table with gains used for position control.

DOF	K	D
p_x	180	10
p_y	180	10
p_z	350	10

(b) Table with gains used for orientation control.

Joint j	K_{1j}	K_{pj}	K_{dj}
1	5	7.5	0
2	5	7.5	0
3	5	7.5	0
4	25	7.5	0
5	25	7.5	0
6	25	7.5	0

With the gains in table 6.2 the results shown in figures 6.11 and 6.12 are obtained. From $t = 0s$ and $t = 10s$ the robot performs a third degree spline (equation 5.2, with the joint being cartesian positions instead) to a ready position near the surface where the robot waits for a command to start the task, depending on the algorithm that is executed. While the robot waits for the operator's command, he/she can adjust the surface their own away, which in turn satisfies the philosophy of collaboration between human and machine.

After $t = 10s$ is when the polishing task initiates, like in the simulator, the Z position reference is lower than the robot's current which permits that tool stays in contact with the surface even if there are movements in the other DOF. The polishing task in performed in the Y axis has shown by the reference going back and forth with the X axis only changing the sections of the surface to be polished.

Contrarily to what happened in the simulation, the robot's orientation does not stay the same in the robot. This can be problematic because for the polishing task to be well executed, the tool should not change its angle to much. In the robot, its orientation does not stay the same, even with this control that tries to completely decouple the position from the orientation control. Still, these results prove that the concept can work in the real robot.

Just like in the simulator, the force in the Y axis has sudden changes between positive and negative values signify that it is the axis where the polishing is more relevant, also the change of the force in the Z axis when the tool comes into contact with the surface. In appendix C there are some photos of the work developed during this project, including a pick and place environment developed in Gazebo to test the SpaceMouse manipulation.

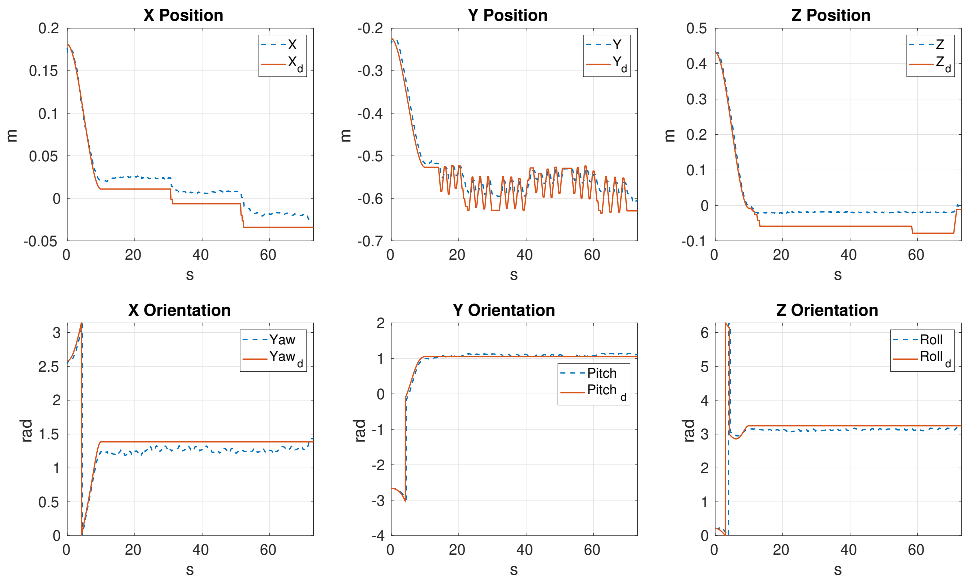


Figure 6.11: Polishing task pose results (robot).

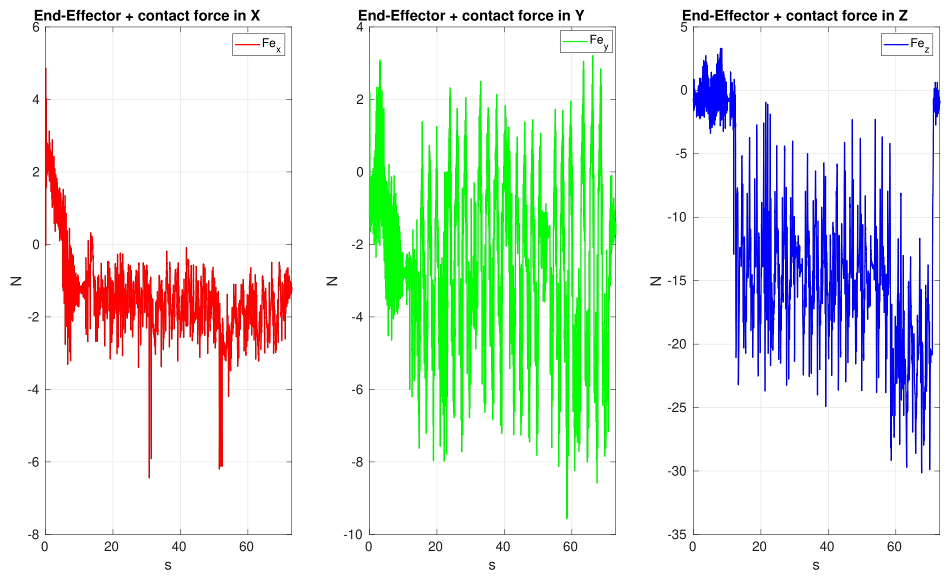


Figure 6.12: Polishing task end-effector plus contact force (robot).

Chapter 7. Conclusions

Despite difficulties found throughout this work, the main objectives were achieved with average to good results. The following topics show some remarks about the work developed:

- The results in the Gazebo simulator permitted the test of the various controls before being tested on the robot, giving a general idea of the robot's behavior when subjected to the control. That way there was little trouble in damaging the robot when testing;
- The results in the real robot are very promising and show that by making more effort and spending more time many controls can be developed for a variety of applications;
- The algorithm developed makes it possible for operators to use the robot to fully polish a surface and then just use the given movements for the robot to replicate them in its entirety. That way, the operators only have to execute the polishing task once, for each surface, and then the robot can safely do the work for as many molds as they want;
- The polishing task results in the Gazebo simulator prove that the concept can be applied with telemanipulation, while the results in the real robot still need more work specially in the part of fixing its orientation so that the tool's angle stays the same throughout the task.
- Finally, as the experimental results in the robot went on, it was found that the robot was only getting the computed values at a rate of approximately 50 Hz which is about half of the control cycle loop rate, which means that the robot was skipping some computed values. That problem was mitigated by removing some unneeded API functions from the file **kinova_arm** which is found in the *kinova_driver* package and the rate got to approximately to 100 Hz but sometimes it still drops too low, at about 60 Hz.

7.1 Future Work

Many different works and experiments can come from this work, so presented next are some future works that can come from this one:

- The controls implemented, although with average to good results, should be more explored and also implement new ones that could benefit the robot's performance;
- The algorithm for the polishing task could be modified so that the robot could execute the polishing task by only knowing the object's shape recurring to its CAD file;
- The inclusion of a camera near the tool can be a good way for the operator to execute the polishing task and can also be used for the robot to execute a better task;

- An extensive study should be made about the force that can be applied to the polishing tool before it breaks since it is important for an operator to know the full extent that the tool can go before breaking;
- Change the telemanipulation peripheral to one that includes an haptic sensor so that the operator is capable of feeling the force that is being applied to the surface by the tool;
- Development of a Graphical User Interface (GUI) with a database for various different surfaces and molds where the operator can choose which one the user wants and then the robot executes the task.

Bibliography

[1] B. Chen, J. Qi, and X. Hu. Polishing trajectory planning method based on the geometry and physics. *2016 IEEE International Conference on Information and Automation (ICIA)*, pages 1461–1466, 2016. doi: 10.1109/ICInfA.2016.7832049.

[2] M. J. Tsai, Jou-Lung Chang, and Jian-Feng Haung. Development of an automatic mold polishing system. *IEEE Transactions on Automation Science and Engineering*, 2(4):393–397, 2005. doi: 10.1109/TASE.2005.853723.

[3] R. Asaga, S. Yamato, and Y. Kakinuma. Analysis of tool posture control method on curved surface using polishing machine with 5-axis serial-parallel mechanism. *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 2979–2984, 2017. doi: 10.1109/IECON.2017.8216503.

[4] J. E. Speich and M. Goldfarb. An implementation of loop-shaping compensation for multidegree-of-freedom macro-microscaled telemanipulation. *IEEE Transactions on Control Systems Technology*, 13(3):459–464, 2005. doi: 10.1109/TCST.2004.839576.

[5] Kinova Robotics. Kinova ultra lightweight robotic arm webpage. `https://www.kinovarobotics.com/en/products/robotic-arm-series/ultra-lightweight-robotic-arm`, 2018.

[6] Kinova Robotics. Gripper kg-3 webpage. `https://www.kinovarobotics.com/en/products/gripper-series/gripper-kg-3`, 2018.

[7] Kinova Robotics. *KINOVA Ultra lightweight robotic arm 6 DOF - Specifications*, 2018.

[8] Kinova Robotics. Kinova controllers webpage. `https://www.kinovarobotics.com/en/accessories/controllers`, 2018.

[9] Kinova Robotics. *KINOVA JACO Prosthetic robotic arm - User Guide*, 2018.

[10] Miguel Pereira Mendes. Computed torque-control of the kinova jaco^2 arm. Master's thesis, University of Coimbra, 2017.

[11] Hélio José Batista Ochoa. Compliant control of the kinova robot for surface polishing. Master's thesis, University of Coimbra, 2018.

[12] Kinova Robotics. *KINOVA Actuator Series KA-75+ KA-58 - Specifications*, 2018.

[13] Kinova Robotics. Kinova sdk webpage. https://drive.google.com/file/d/17_jLW5EWX9j3aY3NGiBps7r77U2L64S_/view, 2018.

[14] Kinova Robotics. Kinova-ros github webpage. https://github.com/Kinovarobotics/kinova-ros, 2018.

[15] Kinova Robotics. Gazebo for kinova robots github webpage. https://github.com/Kinovarobotics/kinova-ros/wiki/Gazebo, 2017.

[16] Gazebo. Gazebo: Tutorial: Ros control. http://gazebosim.org/tutorials/?tut=ros_control, 2014.

[17] 3Dconnexion. Spacemouse compact webpage. https://www.3dconnexion.com/spacemouse_compact/en/, 2018.

[18] ROS. Spacenav_node package webpage. http://wiki.ros.org/spacenav_node, 2017.

[19] Rui Cortesão. Medical robotics, 2016.

[20] Luís Santos and Rui Cortesão. Joint space torque control with task space posture reference for robotic-assisted tele-echography. *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, 2012.

[21] Luís Santos and Rui Cortesão. Computed-torque control for robotic-assisted tele-echography based on perceived stiffness estimation. *IEEE Transactions On Automation Science And Engineering, Vol. 15, No. 3, July 2018*, 2018.

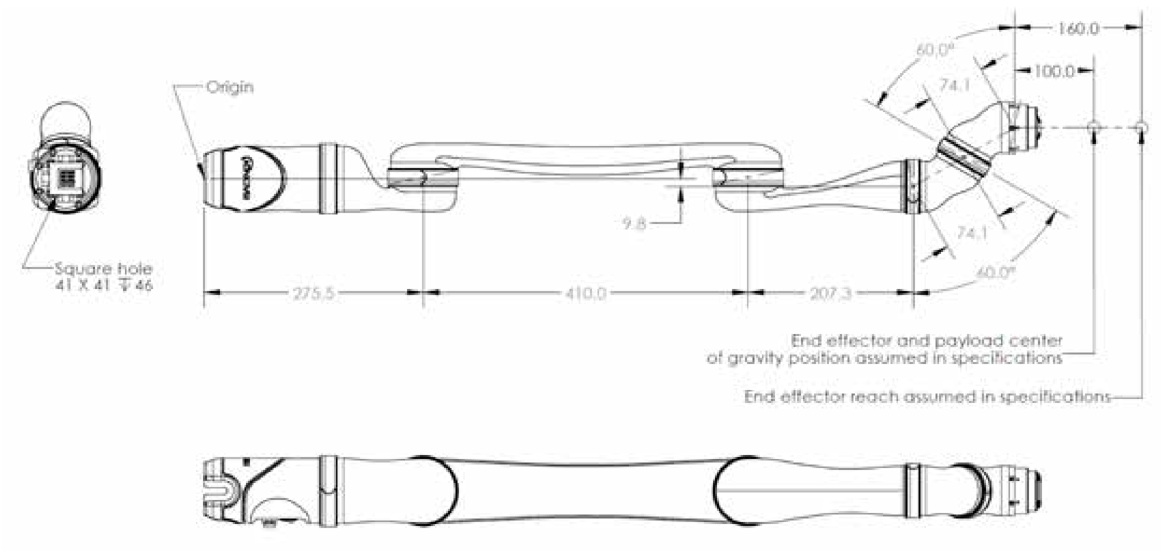
Appendices

A Kinova JACO² Specifications

Tech Specs

KINOVA™
Ultra lightweight robotic arm

6 DOF



160.0
60.0°
100.0
74.1
Origin
Square hole 41 X 41 ⊽ 46
9.8
275.5
410.0
207.3
74.1
60.0°
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)	Up to 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	KA-75+
Actuators #4, #5 & #6	KA-58

KINOVA

+1 514-277-3777 kinovarobotics.com

B Kinova JACO² Actuators and Controller Specifications

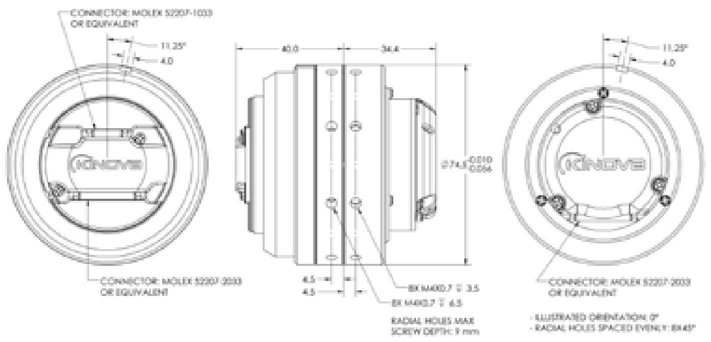
Tech Specs

KINOVA™ *Actuator series*

KA-75+ KA-58

KA-75+

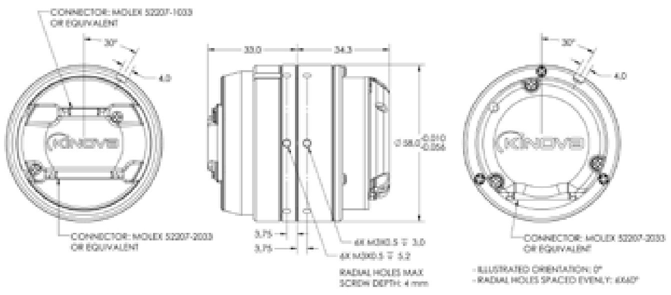
Ø74.5 mm, 12.0 Nm nominal, 37 Nm peak
Brushless DC motor, ratio 160 Harmonic Drive™



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
CONNECTOR: MOLEX 52207-2033 OR EQUIVALENT
- ILLUSTRATED ORIENTATION: 0°
- RADIAL HOLES SPACED EVENLY: 8X45°

KA-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
CONNECTOR: MOLEX 52207-2033 OR EQUIVALENT
- ILLUSTRATED ORIENTATION: 0°
- RADIAL HOLES SPACED EVENLY: 6X60°

GEARED MOTOR (WITH 24V SUPPLY)

	KA-75+	KA-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	1.7 to 5.2 Nm	0.8 to 7 Nm

SENSORS

	KA-75+	KA-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

	KA-75+	KA-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

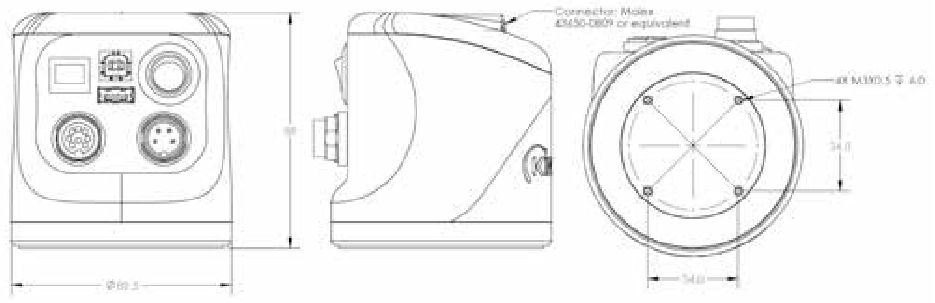
Operating temperature range	-10 °C to 40 °C
Max frame temperature *(overheat protection triggered)*	75 °C

	KA-75+	KA-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)	Up to 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.

C Photo Gallery