Miguel Pereira Mendes

Computed torque-control of the Kinova JACO² Arm

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

September 2017

· U C ·

Universidade de Coimbra

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Computed torque-control of the Kinova JACO^2 Arm

Miguel Pereira Mendes

Coimbra, September 2017

• U C •

Computed torque-control of the Kinova JACO^2 Arm

Supervisor:

Prof. Dr. Rui Pedro Duarte Cortesão

Jury:

Prof. Dr. Jorge Manuel Moreira de Campos Pereira Batista

Prof. Dr. Rui Alexandre de Matos Araújo

Prof. Dr. Rui Pedro Duarte Cortesão

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

Coimbra, September 2017

Acknowledgements

Em primeiro lugar, quero prestar o meu mais sincero agradecimento ao meu orientador, o Professor Doutor Rui Cortesão, pelo seu acompanhamento contínuo, partilha de experiência, encorajamento e aconselhamento durante o desenvolvimento deste trabalho.

Ao meu colega de laboratório, Luís Santos, agradeço pelo acompanhamento e auxílio durante as fases mais críticas do trabalho e por se mostrar sempre disponível para ajudar.

Agradeço igualmente ao Instituto de Sistemas e Robótica de Coimbra, por proporcionar os meios para a realização deste trabalho, e por me permitir trabalhar numa área da qual nutro imenso gosto.

I also thank Martine Blouin at Kinova Robotics, for having the patience to deal with all my questions and problems and for always being helpful. Her contributions were essential in order to tackle some of the technical difficulties encountered in this work.

Aos meus amigos de curso, em especial, Daniel, Bruno, Beatriz, Dylan, Diogo, João, obrigado pela convivência e camaradagem durante este percurso de 5 anos.

Por fim, mas não menos importante, agradeço à minha família, aos meus pais, Manuel e Florinda, pelo seu constante esforço, dedicação e incentivo, por fazerem de mim o que sou hoje. Sem eles este percurso não poderia ter sido realizado. O meu muito obrigado. Aos meus irmãos, André e Sofia, obrigado pelo apoio e amizade durante todo este caminho.

Miguel Pereira Mendes

Coimbra, Setembro 2017

Resumo

Nos últimos anos, várias áreas ligadas à robótica têm evoluído, estando cada vez mais presentes na vida do ser humano. Uma dessas áreas, onde se têm notado mais progressos, é a área da robótica assistiva. Muitas pessoas portadoras de deficiências motoras beneficiam deste tipo de tecnologias, cuja utilidade aumenta consideravelmente a sua qualidade de vida. Existe, por exemplo, um braço robótico relativamente novo no mercado denominado de JACO^2 desenhado especificamente para pessoas possuidoras de deficiência motora ao nível dos membros superiores. Apesar do robô poder ser prontamente utilizado pelo utilizador convencional para execução de tarefas mundanas com o joystick de oferta, por vezes é desejado realizar movimentos específicos com elevada precisão, sendo, no entanto, necessário um conhecimento mais alargado de como o robô funciona internamente, e de como é estruturado fisicamente. Normalmente, isto é alcançado com o design de técnicas de controlo complexas que têm em consideração esses critérios. Arquiteturas de controlo avançadas, como o controlo de binário computarizado permitem o seguimento de trajetórias desejadas com um elevado grau de exatidão, necessitando, ainda assim, da integração de modelos robóticos.

O trabalho apresentado nesta tese envolve a definição dos "alicerces" necessários à aplicação dessas metodologias no braço robótico supracitado. Inicialmente é feita uma análise geral ao JACO^2 verificando-se as suas capacidades e limitações. Posteriormente, os modelos geométrico e cinemático são calculados, seguidos da derivação do modelo dinâmico. A computação deste último é realizada através da análise da energia interna do robô, isto é, com base no método de Euler-Lagrange, sendo os parâmetros dinâmicos obtidos a partir de uma abordagem baseada em elos aumentados. A fiabilidade dos modelos estimados é posta à prova com esquemas de controlo já bem conhecidos tanto ao nível do espaço tarefa como no espaço das juntas, sendo os resultados obtidos analisados.

Os resultados para o modelo geométrico são suficientemente conclusivos para se afirmar que a estimação é bem definida, enquanto que a dedução do modelo dinâmico resultou, na sua generalidade, em resultados aceitáveis e promissores. No entanto, é necessário um estudo mais aprofundado deste último modelo, por forma a melhorar a sua fiabilidade e para que técnicas de controlo mais avançadas possam ser desenvolvidas para este robô no futuro.

Por fim, este documento também propõe uma forma alternativa de controlar o JACO^2, substituindo o joystick disponível, por um rato 3D. Aliado ao facto de ser fisicamente bastante leve e de se apresentar como um sistema portátil, este controlo alternativo representa um dos primeiros passos para um possível desenvolvimento de uma aplicação tele-ecográfica no futuro.

Palavras-chave: JACO^2, robótica assistiva, controlo binário computarizado, modelo robótico, space mouse.

Abstract

In the last years, numerous robotic areas have evolved, being increasingly present in the life of the human. One of the areas whose progresses have been noticeable is the area of assistive robotics. Many people who suffer from mobility impairments or disabilities benefit from these technologies, having their quality of life greatly improved. To name an example, a relatively novel robotic arm is present in the market called the JACO^2 specifically designed for people with upper-body impairments. Although the robot can be promptly used by the general user to perform mundane tasks with the associated joystick, sometimes it is desirable to perform more specific high precision movements, although that requires a deeper insight into how the robot works internally and how it is physically structured. This is normally achieved with the design of complex control techniques that take into account those criteria. Advanced control architectures such as computed torque control allow the tracking of desired trajectories with a high degree of accuracy, albeit needing the integration of robotic models.

The work presented on this thesis revolves around establishing the groundwork for the application of those methodologies in the aforementioned robotic arm. Initially, a complete overview of the JACO^2 is done in order to infer its capabilities and limitations. Afterwards, the geometric and kinematic models are addressed followed by the dynamic model derivation. The computation of this last model is obtained via the analysis of the internal energy of the robot, via the Euler-Lagrange method, while the dynamic parameter identification is based on an augmented link approach. The reliability of the estimated models is then evaluated with already well established control schemes in both task space and joint space, with their results being analyzed.

The results for the geometric model are conclusive enough to affirm that its estimation is well defined, whereas the derivation of the dynamic model, in general, provided acceptable and promising results. However, it is necessary a further in-depth research in order to improve the dynamic model estimation, so that more advanced control techniques can be implemented on this robotic arm.

Lastly, this document also puts forward an alternative way of controlling the JACO^2, using a 3D mouse as a substitute to the available joystick. Coupled with a lightweight composition and a very portable system, this represents an introductory step into a possible development of a tele-echographic application in the future.

Keywords: JACO^2, assistive robotics, computed torque control, robot modeling, space mouse

“The noblest pleasure is the joy of understanding."

— Leonardo da Vinci

Contents

List of Acronyms

3D	Three Dimensional
API	Application Programming Interface
BCI	Brain Computer Interface
BLDC	Brushless Direct Current
CAD	Computer-Aided Design
COM	Center of Mass
DC	Direct Current
DOF	Degree of Freedom
DH	Denavit-Hartenberg
EEG	Electroencephalographic
EL	Euler-Lagrange
EMG	Electromyography
FSM	Finite State Machine
NE	Newton-Euler
OCS	Output Coordinate System
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
SDK	Software Development Kit
USB	Universal Serial Bus

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

For the past few decades, robots have become a major presence in the life of the human. Almost any industry related to automated processes has them incorporated in their assembly lines, mostly in order to replace the human from performing repetitive tasks or from being in a hazardous environment. At the same time, other types of robotic devices started to be developed, in areas like the military, space exploration or to interact with humans. More recently, developments in the area of health care started to be seen, where surgical robots began to be able to aid the surgeon in performing medical procedures, or collaborative robots were introduced as a way to help people with disabilities, allowing them to perform activities of the daily living. Indeed, the presence of robotics in the health care is not comprised of these last two examples. The tele-operation of robotic devices to perform medical tasks over long distances is already a reality [1] [2], allowing the possibility of providing specialized medical care to remote locations. Nonetheless, this potential advantage is shadowed by some limitations such as time delays in data transmission over the local site to the remote site [3] or data acquisition losses due to deficient telecommunications infrastructure. Normally, a system like this, composed of a local operator and a remote station, is referred to as a "master-slave" system, where the former sends desired commands to the latter. Traditionally, the master is provided with a small manipulator, so that the control is made in an easy and intuitive manner. The generated forces from the human through that device are sent and replicated by the slave manipulator. Still, in the vast majority of the applications where these systems are in place, there is the desire to know how the slave manipulator is behaving with the environment. This wish is emphasized when the applications are related to the medical field, as there is the utmost need to guarantee the safety of the patient. As such, the slave manipulator is normally coupled with force sensors whose data is then sent back to the master station. Through the use of an haptic device, the operator is then aware of the interactions of the remote manipulator and can take decisions accordingly. Furthermore, video feedback is also in place, being necessary to operate with accuracy and to get an even greater sense of being present in the environment. This kind of system is commonly called bilateral [4].

Of course, the operation and control of the robotic device placed at the remote site normally requires the development of specific control architectures, in order to represent the displacement generated from the haptic device with minimal error. These control methodologies can be based on geometric models, that establish the relation between the end effector coordinates with the correspondent joint angles of the manipulator; kinematic models, relating the velocity of the end effector and the velocity of the joint angles, and lastly, dynamic models, which combine the relation between the torques applied to the actuators of the manipulator and the position, velocity and acceleration of the joints [5]. Under these assumptions, computed torque control techniques can be conceived, which generally offer good tracking behavior and allow the design of compliant and force controls for human-robot interaction.

1.2 Objectives

Initially, this work was conceived with the idea of developing an alternative way of controlling the JACO^2. For that, the space mouse control suggested. Afterwards, a general study of the capabilities of the robotic arm was proposed, primarily focusing on the torque control mechanisms present on the arm.

1.3 Contributions

This work presents a general overview of a relatively novel robotic manipulator. Initially, its functions, capabitilies and limitations are studied. Afterwards, the kinematic, differential kinematic and dynamic models are estimated and their validity is analyzed under the application of computed torque control techniques.

Lastly, due to its portability and low cost, a further study is made on the possibility of using this arm as part of a tele-echographic system. Thus, a 3D mouse is used to control the trajectory of the arm through the implementation of a finite-state machine, that allows the changing between different configurations of the robot.

1.4 Organization

The document is organized as follows:

- Chapter 1, covers the background, objectives and document structure;
- Chapter 2, deals with the overview of the JACO^2 system, its characteristics and functionalities;
- Chapter 3, introduces the geometric, kinematic and dynamic model deductions;
- Chapter 4, is devoted to the presentation of the control architectures to be tested;
- Chapter 5, examines experimental results;
- Chapter 6, covers the finite-state machine implementation using a space mouse to control the JACO^2 arm;
- Chapter 7, sets out the final conclusions and suggestions for future work.

Chapter 2

The Kinova JACO^2

In this chapter, a general overview of the Kinova JACO^2 is introduced, as well as its main characteristics and functionalities.

2.1 System Overview

The JACO^2 is a robotic arm developed by Kinova Robotics[1], a Canadian company focused on assistive/rehabilitation devices. The first version, JACO Rehab Edition, reached the commercial market in 2010 [6] and was primarily aimed for people with reduced mobility or upper limb impairments. In 2012, an improved version was made for the scientific community, called JACO^2 Research Edition, which is the arm used on this work.

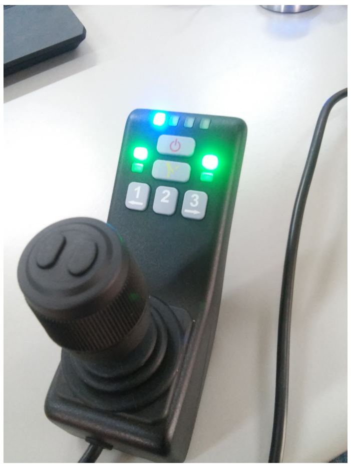
The JACO^2 is a six degree of freedom (DOF) manipulator (Fig. 2.1a) with a removable three fingered hand acting as the end effector.

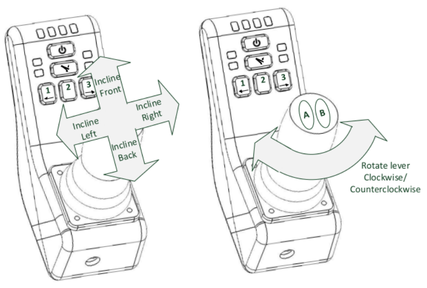
The manipulator is composed of six interconnected carbon fiber links, jointed together by six aluminum brushless direct current (BLDC) actuators. They possess no mechanical limitation allowing unlimited rotation around their axis. Unsurprisingly, due to the nature of the carbon fiber, the main structure ends up having a very lightweight composition. Additionally, this material also delivers structural integrity and reliability to the arm while the aluminum actuators contribute for better heat dissipation during arm movements. The hand consists of three fingers which can be individually controlled. These are made of plastic which allow them to firmly adjust to different sizes and types of objects.

From the user guide [7], additional specifications can be described. Table 2.1 summarizes that information. Additionally, a more detailed description of the JACO^2 is made in

[1]`http://www.kinovarobotics.com/`

(a) JACO^2.


(b) JACO^2 joystick.



(c) Joystick Movements. [7]

Figure 2.1: JACO^2 system.

appendix E, whereas for the actuators that compose the robot, appendix F describes more thoroughly their specifications.

Table 2.1: General Specifications

	JACO^2 6 DOF (without gripper)
Total Weight	4.4 Kg
Payload	2.6 Kg (mid-range continuous)
	2.2 Kg (full-reach peak/temporary)
Reach	90 cm
Maximum linear arm speed	20 cm/s
Power supply voltage	18 to 29 V (DC)
Average Power	25 W (5W in standby)
Peak Power	100 W
Communication Protocol	RS485

The arm is controlled through a three-axis joystick with five independent push buttons and four external auxiliary inputs placed at the back side of the controller. The figure 2.1b shows the front side of the controller, having, from top to bottom, the power button, HOME button, and other five different buttons responsible for the switching of operation modes, with two of them being placed on top of the joystick. The blue light displays the current operation mode while the green lights show that the robot is powered and ready to be used. The HOME button repositions the arm on a preprogrammed pose. The remaining buttons are assigned to the shift between three-axis and two-axis mode (more information in the appendix of [7]). Basically, the JACO^2 can be easily changed into translation mode, wrist mode, “drinking” mode and finger mode. The control of the JACO^2 arm using the joystick is considered cartesian as the user can only change the position/orientation of the gripper. That being said, the translation mode enables the controlling of the hand in space in the three axis of the cartesian coordinate system. For the wrist mode, the arm is controlled around a reference point (set in the middle of the end effector). The arm stays stationary around that point changing its orientation depending on the joystick commands. The “drinking mode”, allows the wrist to produce a rotation around another point in the space of the arm, through an offset in height and length from the reference point. This was named this way because it normally helps during the grasp of bottles or water cups. Finally, the finger mode lets the user open and close the hand at will. A multitude of other operation modes can be achieved by combining various buttons (refer to [7]).

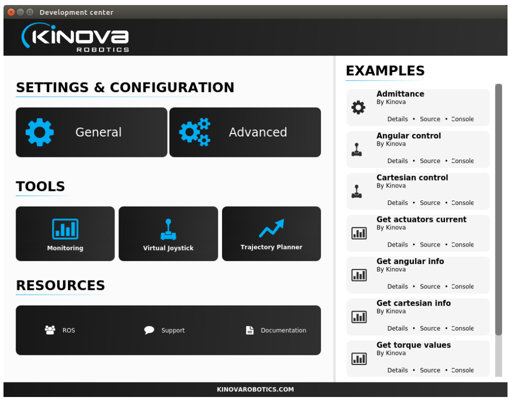
The joystick movements that can be executed are presented on figure 2.1c. The joystick provided by Kinova is intuitive at start, but the amount of different modes that can be chosen might be confusing for unfamiliar users. It requires practice, and as correctly pointed out by Herlant et al. [8], who interviewed actual users, the main struggle lies on the “switching between the various control modes” and that there are “a lot of modes, actions, and combination of buttons”. This is in line with what was experienced during this work.

The JACO^2 arm presents itself as a portable device, due to its very light frame. Adding to this, the power supply is considerable small which effectively puts into evidence the easiness of carrying this device to anywhere. Moreover, the arm possesses sensors to ascertain temperature, voltage, current and torque parameters. In terms of hardware connectivity, apart from the necessary connections for an approved power supply/battery and joystick, the arm comes with an ethernet port, for wireless connections, and a universal serial bus (USB) port to connect to a laptop or personal computer (PC), in order to send commands through the available software development kit (SDK). Analyzing table 2.1 one can see that the communication protocol is the RS485, detailed discussion pertaining this will be done on section 2.3.

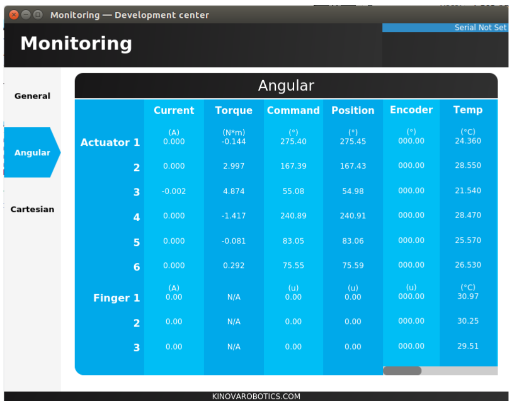
2.2 Software Development Kit

Coupled with the joystick, Kinova offers a complete graphical user interface to control the arm, both in cartesian and torque modes.

The SDK grants the user complete control over the functionalities of the manipulator, without the need to use the joystick. This is attained by connecting the USB port of the robot to a PC via a USB cable. After installing and opening the program [9], the Development Center comes up (figure 2.2a). As one can observe, different tabs can be chosen, and distinct parameters of the arm can be changed or verified. For instance, the Monitoring section, shown in figure 2.2b, collects all the data from the sensors included in the manipulator, as well as the calculations of the angular position and velocity, a similar table is visualized if the cartesian subtab is chosen.


Development center
KINOVA ROBOTICS
SETTINGS & CONFIGURATION
General
Advanced
TOOLS
Monitoring
Virtual Joystick
Trajectory Planner
RESOURCES
ROS
Support
Documentation
EXAMPLES
Admittance By Kinova Details • Source • Console
Angular control By Kinova Details • Source • Console
Cartesian control By Kinova Details • Source • Console
Get actuators current By Kinova Details • Source • Console
Get angular info By Kinova Details • Source • Console
Get cartesian info By Kinova Details • Source • Console
Get torque values By Kinova Details • Source • Console
KINOVAROBOTICS.COM

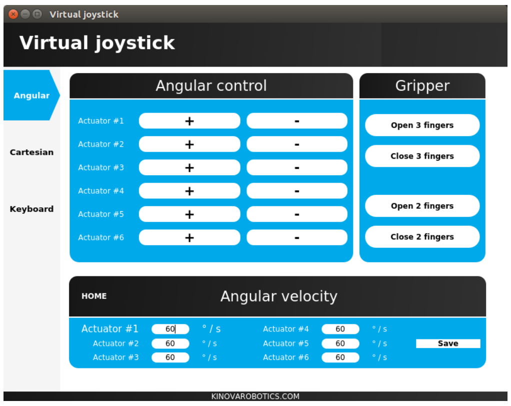
(a)



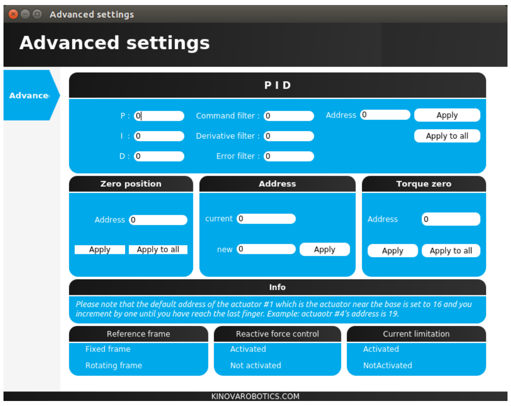
Angular

	Current	Torque	Command	Position	Encoder	Temp
Actuator 1	(A) 0.000	(N*m) -0.144	(°) 275.40	(°) 275.45	(°) 000.00	(°C) 24.360
2	0.000	2.997	167.39	167.43	000.00	28.550
3	-0.002	4.874	55.08	54.98	000.00	21.540
4	0.000	-1.417	240.89	240.91	000.00	28.470
5	0.000	-0.081	83.05	83.06	000.00	25.570
6	0.000	0.292	75.55	75.59	000.00	26.530
Finger 1	(A) 0.00	N/A	(u) 0.00	(u) 0.00	(u) 000.00	(°C) 30.97
2	0.00	N/A	0.00	0.00	000.00	30.25
3	0.00	N/A	0.00	0.00	000.00	29.51

(b)



(c)



Please note that the default address of the actuator #1 which is the actuator near the base is set to 16 and you increment by one until you have reach the last finger. Example: actuaotr #4's address is 19.

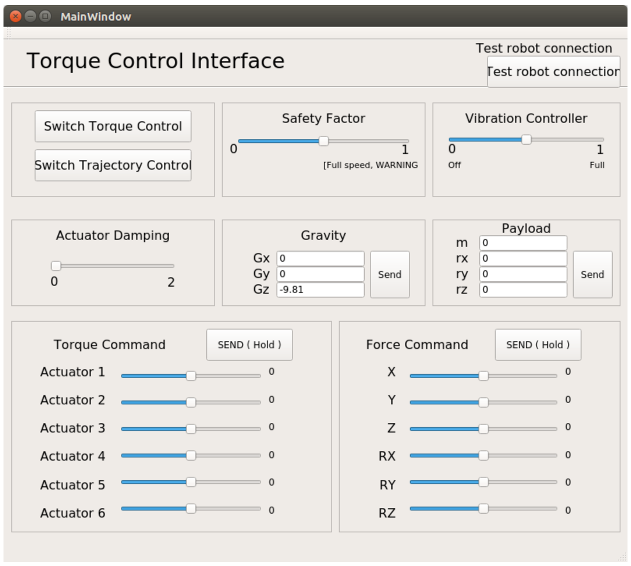
(d)

Figure 2.2: Software Development Kit.

A virtual joystick enables the control of the arm in a similar way as the real one. Figure 2.2c depicts the different actuators and the plus and minus sign refer to the correspondent direction of rotation. The cartesian mode is similar, although the user has the position and orientation axis to interact with. However, it is important to refer the fact that the cartesian velocities are referred to different reference frames. While the linear velocity is referred to the base frame, the angular velocity is associated to the reference frame of the end effector. On top of this, the advanced settings exhibit additional configuration parameters to manipulate.

A proportional-integral-derivative controller (PID) is accessible to manage the behavior of the position control of each actuator, zero position resets the actual position of the targeted actuator to 180°, and torque zero, assumes the actual torque of the considered actuator to be 0. Furthermore, the reference frame option enables the choosing between fixed and rotating frames. The former makes the orientation of the arm fixed while performing a translation, while the latter allows the arm to follow the orientation of the axis of the base actuator. The admittance control can be activated or deactivated with the reactive force control option. The remaining configuration options and menus can be referred to the SDK User Guide [9], which has more detailed information.

Alternatively, the user might wish to use the arm while in torque control. Like the Development Center, the Torque Console is another interface available to the users.



Gx 0
Gy 0
Gz -9.81

m 0
rx 0
ry 0
rz 0

Actuator 1 0
Actuator 2 0
Actuator 3 0
Actuator 4 0
Actuator 5 0
Actuator 6 0

X 0
Y 0
Z 0
RX 0
RY 0
RZ 0

Figure 2.3: Torque Console.

With torque control, the manipulator behaves smoothly when contact forces are applied. As observed from figure 2.3, the user has at his disposal different parameters to adjust as well as the choice between sending torque or force commands to the JACO^2. The safety factor is related to how the arm deals with high speed behaviors when applying high contact forces. If the parameter is set to one, the arm does not switch back to trajectory control no matter the current speed. On the other hand, if set to zero the arm is less prone to high speed motions and switches back to trajectory control more easily. If interaction with stiff environments is desired, a vibration controller/observer can be set to eliminate them. The actuator damping parameter simulates the damping effect applied directly to the actuators. The gravity vector can also be changed, and since the gripper can be removed the payload parameters can be finely tuned to other specific end effectors.

As a first approach, these consoles were useful to analyze and acknowledge the behavior of the arm to different parameters. Even so, the ideal way for developers to implement their desired control mechanisms and programs lies on the Kinova Application Programmming Interface (API), which will be covered on the next section.

2.3 Kinova API and Communication Modes

One of the first problems that was encountered when first addressing the objectives of this work, was the initial feeling of being in a presence of a "*black box*"[2]. The initial tests with the provided SDK demonstrated that entirely. Fortunately, for the developers who wish to create specific software programs, an API is available with a vast set of tools to meet their needs.

Previously, on section 2.2 the Development Center menu was illustrated (fig. 2.2a). On the right side, it is possible to visualize different examples, like admittance, angular or cartesian control. Those subroutines are built using the API functions. One such example can be checked on appendix A. The programming language from where the API is built is C++, which can be considered understandable even for unexperienced users.

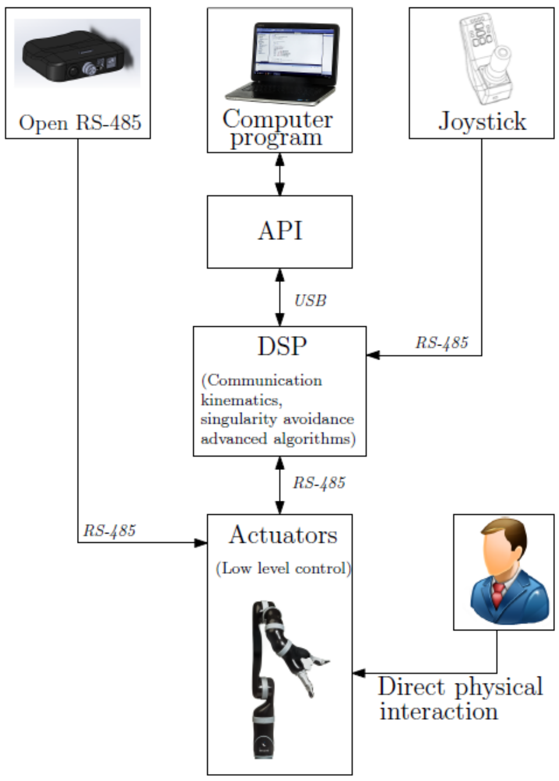

Open RS-485
Computer program
Joystick
API
USB
DSP
(Communication kinematics, singularity avoidance advanced algorithms)
RS-485
RS-485
RS-485
Actuators
(Low level control)
Direct physical interaction

Figure 2.4: Communication Modes.[10]

[2]A system where its internal workings are not visible to the user.

As described in table 2.1, the internal communication protocol used to control the actuators is the RS485. It is, of course, possible to interact directly with the actuators by using an appropriate RS485 interface or USB-RS485 communications module. Kinova delivers a specific user guide (included in the installation folder of the Kinova SDK) detailing how the RS485 message format should be constructed and the corresponding message types that make the arm execute the desired commands [11]. Still, it is not the only way of establishing a communication link with the arm, as it is summed up by figure 2.4. There is also the chance of using the Robotic Operating System (ROS), to achieve a communication link with the arm [12].

Naturally, the arm needs some way of processing the data that it receives. The digital processing unit (DSP) incorporated in the base of the arm acts as the “brain” of JACO^2, evaluating the commands that are sent and transmitting them to the actuators. Also, this crucial component of the arm is responsible for the gravity compensation, mode switching management and other advance algorithms that guarantee the safety of the arm and its users, as well as the execution of the desired tasks.

Having covered the RS485 and joystick communication methods, the API-JACO^2 communication method is left to be explained. Firstly, it is important to refer that the Kinova API and SDK both work with Windows or Linux. For this case, it is going to be considered the Ubuntu 16.04 system, as it was the one where this work was done. Observing the subroutine in appendix A, it is noticeable that some sequence of actions must be taken in order to connect to the arm. Apart from the necessary files to use the API under Ubuntu [10], the library containing the functions needed must be loaded and the respective functions initialized. Similarly, the arm must be started, but for this, two API functions need to be called: **MoveHome** and **InitFingers**. The former commands the arm to position itself in the HOME position while the latter initializes the fingers by opening the hand. Reaching the end of the program, the library must be closed before exiting the program. In fact, not following this last step will leave the current program running and render any attempts in trying to execute subsequent programs useless.

Last but not least, the control system frequency while using the API is either 100 or 500 Hz, depending on what communication method is used. If it is high level, that is, via USB, then the rate is between 100-500 Hz, but the refresh rate of the controller of the DSP is 100 Hz. In contrast, if the communication is made directly with the actuators, the low level approach, then the communication rate is 500 Hz. Clearly, it is obvious the advantages of using the low level API with the RS485. Nonetheless, all the work presented on this document was made following a high level approach, which means that all the control architectures follow the refresh rate represented by the DSP controller, that is 100 Hz. Further information can be found on appendix G.

2.4 Torque Control

So far, it has been studied the way the torque control can be tested and used with the Torque Console (fig. 2.3), but it remains to be mentioned how it is implemented internally.

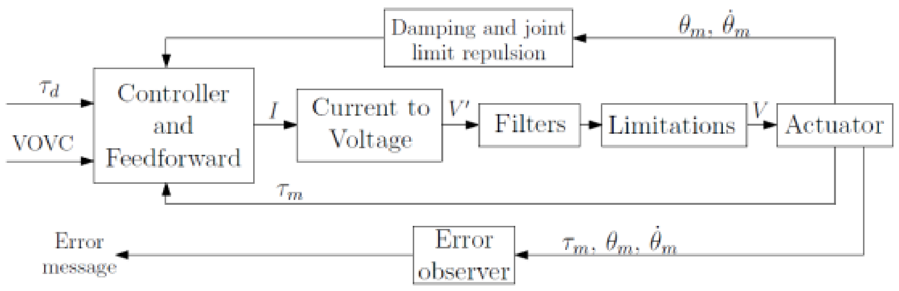

Damping and joint limit repulsion
$\theta_m, \dot{\theta}_m$
τ_d
VOVC
Controller and Feedforward
I
Current to Voltage
V'
Filters
Limitations
V
Actuator
τ_m
Error message
Error observer
$\tau_m, \theta_m, \dot{\theta}_m$

Figure 2.5: Low Level Torque Control Scheme.[10]

The scheme shown in 2.5 corresponds to the actuator control during torque control mode. The VOVC is the vibration observer/controller. This is the default control mechanism implemented in the DSP, during torque control mode. It can be assumed that the “Controller and Feedforward” block is responsible for the different algorithms included in the arm like singularity avoidance, kinematics and position/velocity estimation. Nonetheless, studying the behavior of the arm while using the default torque control, showed that its motions were not natural and smooth when handling the arm. Undoubtedly, additional control architectures were present which, for the purposes of this work, needed to be removed or eliminated. Needless to say, this was one of the biggest challenges of this work: to have complete knowledge over what control procedures were being executed by the arm, and how to remove them, since it was necessary to have direct access to the actuators without interference from preexistent control methodologies (refer to section 5.1 for detailed information related to this problem).

Chapter 3

JACO^2 Model Identification

This chapter will focus on the derivation of the kinematic, differential kinematic and dynamic models of the JACO^2. These models need to be estimated, in order to perform tasks in the workspace with the best possible fashion. The kinematic model, or also called geometric model, establishes the relation between a given configuration of the robot in joint variables and the correspondent position and orientation coordinates of the end effector. The differential kinematic, in short, entails the relationship between joint and end effector velocities. On the other hand, the dynamic model is based on the relation between forces/torques and the resulting motion, and allows a deeper analysis considering additional criteria such as mass, inertia and even friction terms that might influence the behavior of the robot during task operations. In the next sections, the deduction of each model is done as well as the necessary steps that were followed in order to obtain them.

3.1 Kinematic Model

The JACO^2 is comprised of six carbon fiber links. By definition, each of its links can be assumed as a having a relative position and orientation with regard to the base of the arm. In the field of robotics, this is called a reference frame, which is associated to the reference frames of the other links by applying homogeneous transformations.

Suppose an arbitrary point P in space needs to be represented with respect to the base frame o_0 (figure 3.1). If its position is only known from the reference frame o_1 then, one way to achieve this would be by considering the position from that reference frame, adjust the orientation of the point to match that of the base frame, and add the distance between the origin of both reference frames. Assuming that the 3×3 dimension matrix R_1^0 is the rotation matrix between frames o_1 and o_0 then the position of P with respect to the base frame is given by

$$p^1 = o_1^0 + R_1^0 p^1 \tag{3.1}$$

where $o_1^0 \in \mathbb{R}^3$ is the distance between the reference frames and $R_1^0 \in SO(3)$.

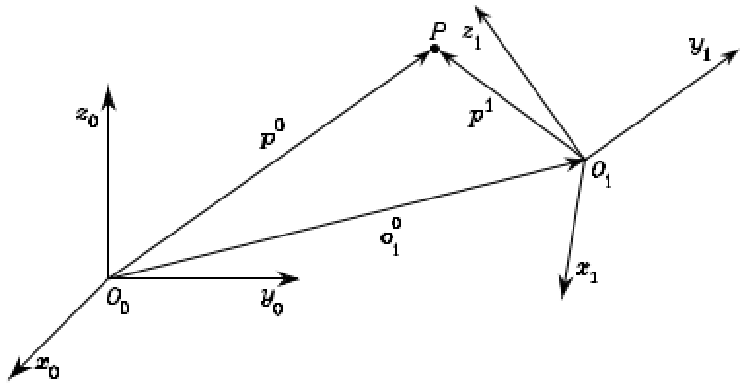


Figure 3.1: Representation of a point between reference frames. [13]

Knowing this, an homogeneous transformation is nothing more than the matrix representation between two frames. Following the example from figure 3.1, that would mean

$$T_1^0 = \begin{bmatrix} R_1^0 & o_1^0 \\ O^T & 1 \end{bmatrix} \tag{3.2}$$

where T_1^0 is commonly called in the literature as the homogeneous transformations matrix. This encapsulates information about the distance between the origins of the frames and the orientation between them by means of a rotation matrix. These 4×4 dimensional matrices are essential in order to derive the kinematic model.

Analogously, a 6 DOF robotic manipulator like the JACO^2 can be represented by a set of reference frames between each of its links, in a similar fashion as displayed by figure 3.1. Assuming that to each link of the arm a coordinate frame is fixed, then the transformation between the end effector and the base frame is given by

$$T_n^0(q) = T_1^0(q_1)T_2^1(q_2)\dots T_n^{n-1}(q_n) \tag{3.3}$$

where $n = 6$ for the case of the robotic manipulator studied.

3.1.1 Denavit-Hartenberg Parameters

Equation 3.3, presents the theoretical principle revolving the direct kinematics model. Still, one would wonder how to analytically obtain the transformation matrices of each link.

To derive them, it is necessary to know how the different reference frames are physically related, and with the Denavit-Hartenberg (DH) parameters it is possible to ascertain that. The JACO^2 Advanced Specification Guide [14] describes the classic DH parameters for the manipulator as well as the length values for each link. Below, table 3.1 provides the dimensions of the arm, pictured in figure 3.2.

Robot length values (in meters)		
D1	0.2755	Base to elbow
D2	0.4100	Arm length
e2	0.0098	Joint 3-4 lateral offset
D3	0.2073	Front arm length
D4	0.0741	First wrist length
D5	0.0741	Second wrist length
D6	0.1600	Wrist to center of the hand

Table 3.1: Robot length values.

Auxiliary Variables	
aa	$\frac{\pi}{6}$
ca	$\cos(aa)$
sa	$\sin(aa)$
c2a	$\cos(2aa)$
s2a	$\sin(2aa)$
d4b	$(D3 + D5(\frac{sa}{s2a})$
d5b	$D4(\frac{sa}{s2a}) + D5(\frac{sa}{s2a})$
d6b	$D5(\frac{sa}{s2a}) + D6$

Table 3.2: Auxiliary Variables.

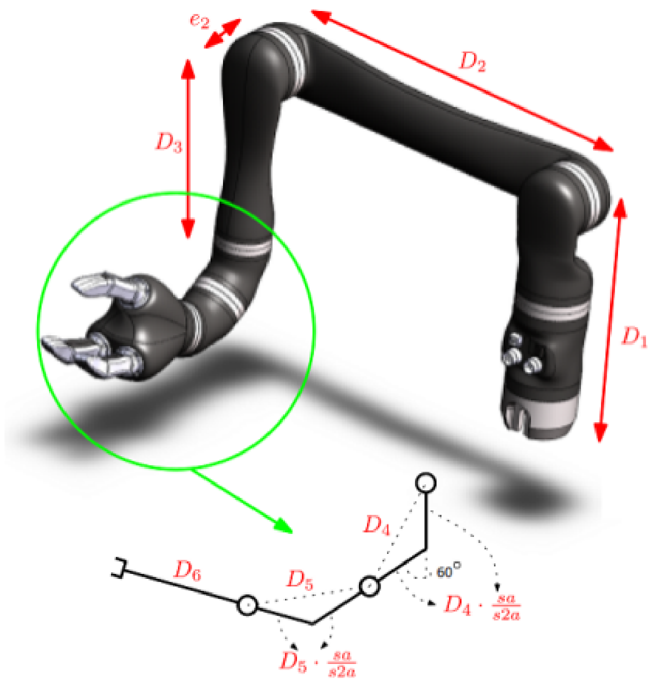


Figure 3.2: JACO^2 link lengths.

Coupled with this, the classic DH parameters are shown in table 3.3. The variables shown in that table are listed on table 3.2. The transformation matrices associated to each link, following the DH convention, are of the fashion

$$T_n^{n-1} = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)\sin(\theta_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	$\frac{\pi}{2}$	0	D1	q_1
2	π	D2	0	q_2
3	$\frac{\pi}{2}$	0	- e2	q_3
4	2*aa	0	-d4b	q_4
5	2*aa	0	-d5b	q_5
6	π	0	-d6b	q_6

Table 3.3: DH Parameters

DH Algorithm Angle	Physical Angle
q_1	$-q_{1_{Robot}}$
q_2	$q_{2_{Robot}} - 90°$
q_3	$q_{3_{Robot}} + 90°$
q_4	$q_{4_{Robot}}$
q_5	$q_{5_{Robot}} - 180°$
q_6	$q_{6_{Robot}} + 90°$

Table 3.4: Angle Transformation.

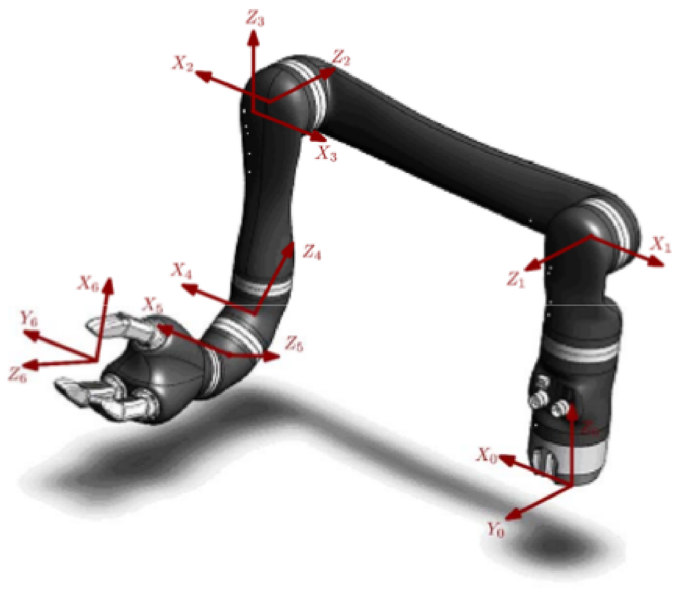


Figure 3.3: Coordinate frames for the DH Algorithm.

Additionally, the JACO^2 physical angles have to be converted into the angles of the Denavit-Hartenberg algorithm, which is given by relation shown in table 3.4. The DH parameters are obtained after establishing the different coordinate frames for each link which are exhibited in figure 3.3.

Since the DH parameters are known, by having the transformation matrices relating subsequent links by applying equation 3.4, then the forward kinematics model can be calculated through equation 3.3. With this, it is now of interest to introduce the differential kinematics model.

3.2 Differential Kinematic Model

As was referred in the beginning of this chapter, the relationship between the angular and linear velocities of the end effector and the joint velocities is named the differential kinematic model. This relation is encapsulated in a matrix called the Jacobian, which apart from being important in singularity analysis and inverse kinematics, can also be of useful for computing the torques being in effect at the joints when a specific force is being applied at the end effector. This, of course, is of particular interest to this work, as it will be seen in chapter 4, when operational space control schemes will be described [15].

First, let the end effector linear velocity be defined as $\dot{p}_e$ and the angular velocity ω_e, while the joint velocities be $\dot{q}$, then the relation described above is

$$\begin{cases} \dot{p}_e = J_P(q)\dot{q} \\ \omega_e = J_O(q)\dot{q} \end{cases} \tag{3.5}$$

where $J_P(q)$ is a $(3 \times n)$ matrix relating the linear velocity of the end effector $\dot{p}_e$ with the joint velocities $\dot{q}$, and the $J_O(q)$ is similarly a $(3 \times n)$ matrix detailing the relation between the angular velocity of the end effector and the joint velocities. Combined together, these submatrices make the Jacobian matrix J

$$v_e = \begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix} = J(q)\dot{q} \quad , \quad J(q) = \begin{bmatrix} J_P(q) \\ J_O(q) \end{bmatrix} \tag{3.6}$$

Extending equation 3.6, the linear and angular velocity components can be computed as

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

The joint velocities $\dot{q}_i$ are expressed differently if the joints are prismatic or revolute. In this case, the JACO^2 has six joints with all of them being revolute, therefore the angular and linear velocity for a revolute joint for subsequent links is denoted as

$$\begin{cases} \dot{p}_i = \dot{p}_{i-1} + \omega_i \times r_{i-1,i} \\ \omega_i = \omega_{i-1} + \dot{\theta}_i z_{i-1} \end{cases} \tag{3.8}$$

where z_{i-1} is the unit vector of the joint i axis and $r_{i-1,i}$ corresponds to the distance from the origin of the coordinate frame i with respect to the origin of the coordinate frame $i-1$. This means that the velocity in link i is the same as in link $i-1$ with the increment of a component related to the change due to the rotation of the link i. Then, assuming that the computation of the linear velocity of each link is made with respect to the coordinate frame of the end effector, the equalities presented in equation 3.7 are rearranged as

$$\begin{cases} J_{P_i}\dot{q}_i = \omega_{i-1,i} \times r_{i-1,e} = \dot{\theta}_i z_{i-1} \times (p_e - p_{i-1}) \\ J_{O_i}\dot{q}_i = \dot{\theta}_i z_{i-1} \end{cases} \Leftrightarrow \begin{cases} J_{P_i} = z_{i-1} \times (p_e - p_{i-1}) \\ J_{O_i} = z_{i-1} \end{cases} \tag{3.9}$$

where p_e is the distance from the origin of the end effector coordinate frame to the base frame and p_{i-1} the analogous distance from link $i-1$. Given these premises, the Jacobian can be translated into

$$J = \begin{bmatrix} J_{P_1} \dots J_{P_n} \\ J_{O_1} \dots J_{O_n} \end{bmatrix} \tag{3.10}$$

which encloses the conclusions mentioned above

$$\begin{bmatrix} J_{P_i} \\ J_{O_i} \end{bmatrix} = \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} \tag{3.11}$$

As observed, the computation of the Jacobian matrix can be achieved following a recursive methodology, where z_{i-1} and p_{i-1} are cyclically calculated. Thus, it is easily seen that the Jacobian will be dependent upon the configuration of the manipulator. This is because the variables described on equation 3.11 are extracted from the homogeneous transformations obtained from equation 3.3, which rely entirely on joint variables. That being said, p_e corresponds to the first three elements of the last column of T_n^0 and p_{i-1} is equivalent, but for the respective matrix T_{n-1}^0. As for the unit vector of the joint i, z_{i-1}, it is obtained from the third column of the rotation matrix R_{i-1}^0 which is easily retrieved, in a similar fashion, from equations 3.2 and 3.3.

Conceptually, with these calculations the end effector velocity is referred to the origin of the base frame, but by knowing the transformation matrices that make the direct kinematics model, it is possible to make the end effector velocity be referred to any other frame of the manipulator.

For this work, the Jacobian related to the linear velocities is referred to the base frame while the angular velocities are referred to the end effector frame, which means it is necessary to apply the following remapping

$$J^e_\omega = R^e_0 J^0_w \tag{3.12}$$

Being an orthogonal matrix, the application of the transpose to the matrix

$$(R^0_e)^{-1} = (R^0_e)^T = R^e_0 \tag{3.13}$$

is enough to reference the end effector angular velocity to its coordinate frame and is computationally less expensive than applying the inverse. This approached was assumed because the orientation control of the end effector proved to be more intuitive if done this way.

3.2.1 Inverse Differential Kinematic Model

After having deduced the direct kinematic model, and since the JACO^2 is non redundant the inverse kinematics model is calculated by simply inverting the Jacobian matrix

$$\dot{q} = J^{-1} v_e \tag{3.14}$$

Alternatively, the Kinova API has a function named **MyGetAngularPosition**, which retrieves the position of the joint angles. This enables the calculation of $\dot{q}$ by the derivative of q in discrete time

$$\dot{q}_k = \frac{q_k - q_{k-1}}{h} \tag{3.15}$$

where h is the period of the control system, which is 10 milliseconds. In this work, the angular velocity of the joints was obtained via this method, but due to real-time issues, sometimes the angular velocity was assumed to be zero, even during motions. This caused small jerky movements that negatively influenced the behavior of the manipulator. Consequently, some tweaking was required by doing

$$\dot{q}_k = \dot{q}_{k-1} \quad , \quad \text{if} \quad q_k - q_{k-1} \approx 0 \tag{3.16}$$

This improved the overall system performance of the arm during the experimental testing of the control architectures to be illustrated in chapter 4.

3.3 Dynamic Model

In the previous sections, the geometric and kinematic models were addressed. Now, it remains to be described and explained, the steps taken in order to estimate the dynamic model. The Jacobian already allows the design of computed torque techniques although those will only rely upon the geometry of the arm. On the other hand, the dynamic model, besides taking that into account, incorporates other factors called the robot dynamic parameters which entail the mass, inertia, frictions and other unknown parameters of the manipulator that can negatively affect its performance. Additionally, this model will grant the possibility of designing additional control architectures for both joint space and task space.

3.3.1 Theoretical Background

The dynamic model is based on the relation between the position, velocity and acceleration of the joints and the resulting torques delivered to the joints. Similar to the previous models, a forward or inverse approach can be pursued [5], where the first is based on the estimation of the motions of the robot, given the torques/forces applied to the joints

$$\ddot{q} = g(q, \dot{q}, \tau, F_e) \tag{3.17}$$

while the second resorts to the accelerations, velocity and positions being in effect, in order to derive the necessary joint torques to be sent

$$\tau = f(q, \dot{q}, \ddot{q}, F_e) \tag{3.18}$$

Both equations 3.17 and 3.18 are fundamental to the design and study of control architectures based on torque/force motions and to evaluate if the robot performs the desired task within the expected. As mentioned earlier, this model revolves around characteristics that are intrinsic to the robot like its mass either be it from the links or the actuators, and its inertia. Of course, these parameters are clearly of constant nature and their estimation, as long as it is made correctly, can greatly improve the performance of the robot during the execution of demanding or complex motions.

In the literature, various proposals are made regarding the model identification of robots recurring to their dynamics. Most of the research has been done applying two methods: Euler-Lagrange (EL) [16] [17] and Newton-Euler (NE) [18] [19], or even a combination of both [20].

The EL technique has its foundation on the kinetic and potential energies existent during robot motion. This can be summarized by the Lagrangian L, represented by the equality

$$L(q,\dot{q}) = K(q,\dot{q}) - P(q) \tag{3.19}$$

where $K(q,\dot{q})$ and $P(q)$ are scalar values representing the kinetic and potential energy of the manipulator. The kinetic energy is given by [21]

$$K(q,\dot{q}) = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{3.20}$$

whereas the potential energy is described as,

$$P(q) = -g_c^T r_c m \tag{3.21}$$

where g_c is the gravity acceleration vector (3 $\times$ 1) and r_c the center of gravity for each link of the arm (n $\times$ 1) and m is the vector containing the correspondent masses (n $\times$ 1). Deriving equation 3.19 with respect to time, entails the joint generalized non-conservative torque

$$\tau = \frac{d}{dt}\frac{\partial L(q,\dot{q})}{\partial q} - \frac{\partial P(q)}{\partial q} \tag{3.22}$$

which after some algebraic manipulation yields,

$$\tau = M(q)\ddot{q} + C(q,\dot{q}) + g(q) \tag{3.23}$$

where $M(q)$ is defined as a symmetric positive definite inertia matrix (n $\times$ n), $C(q,\dot{q})$ is the Coriolis and centripetal forces term ($n \times 1$) and $g(q)$ is the term relating to the gravity forces. With the derivation of equation 3.23, the calculation of these matrices[1] in real-time allow the application of this model in specific control architectures based on torque control.

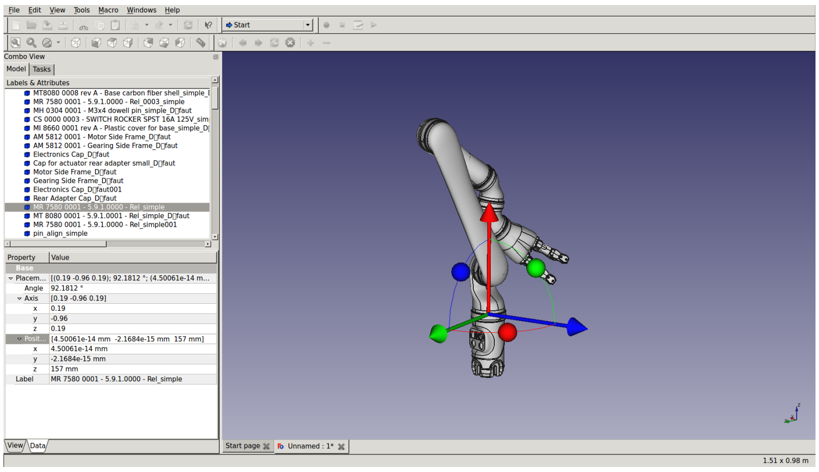
Alternatively, the NE technique can be pursued, which revolves around the velocities, accelerations and torques/forces in effect on each link. The method is recursive which means that the computations are made from link to link. Initially, the velocities and accelerations

[1]any advanced robotics textbook like [5] [13] [15] [22] illustrates the necessary steps.

of the augmented links[2] are calculated in an iterative way from the base to the end effector. Then, the iterative process is inverted, in order to find the resulting torques/forces exerted on the links. Naturally, both methods must yield the same results. There has been, however, some discussions regarding the computational efficiency of both methods. The EL method is computationally less efficient than the NE method, because apart from the fact that it is not recursive, it also makes all the necessary calculations with reference to the base frame, which leads to very complex expressions when dealing with a 6 DOF robot. Despite this, [23] claims that a recursive Lagrangian methodology can achieve the same efficiency as the recursive NE method [24].

3.3.2 Dynamic Parameters

To obtain the dynamic parameters of the JACO^2, Kinova provides to its costumers the computer-aided design (CAD) model of the robot. In addition to this, an Excel document is given as an auxiliary document, describing with more detail each link and actuator.



- MT8080 0008 rev A - Base carbon fiber shell_simple_D
- MR 7580 0001 - 5.9.1.0000 - Rel_0003_simple
- MH 0304 0001 - M3x4 dowell pin_simple_D□faut
- CS 0000 0003 - SWITCH ROCKER SPST 16A 125V_sim
- MI 8660 0001 rev A - Plastic cover for base_simple_D
- AM 5812 0001 - Motor Side Frame_D□faut
- AM 5812 0001 - Gearing Side Frame_D□faut
- Electronics Cap_D□faut
- Cap for actuator rear adapter small_D□faut
- Motor Side Frame_D□faut
- Gearing Side Frame_D□faut
- Electronics Cap_D□faut001
- Rear Adapter Cap_D□faut
- MR 7580 0001 - 5.9.1.0000 - Rel_simple
- MT 8080 0001 - 5.9.1.0001 - Rel_simple_D□faut
- MR 7580 0001 - 5.9.1.0000 - Rel_simple001
- pin_align_simple

Property	Value
Base	
Placem...	[(0.19 -0.96 0.19); 92.1812 °; (4.50061e-14 m...
Angle	92.1812 °
Axis	[0.19 -0.96 0.19]
x	0.19
y	-0.96
z	0.19
Posit...	[4.50061e-14 mm -2.1684e-15 mm 157 mm]
x	4.50061e-14 mm
y	-2.1684e-15 mm
z	157 mm
Label	MR 7580 0001 - 5.9.1.0000 - Rel_simple

Figure 3.4: CAD model of the JACO^2 robot.

The CAD model provides an overall visualization of the robot with its different aggregated components. The complete set of coordinate frames for the various links and actuators is illustrated along with their relative distances with respect to the base frame. Additional information is given in the Excel document, where the mass and inertia tensors for each link and actuator are described. Figure 3.5b entails the mass and inertia tensors for the base link.

The straight blue line represented in the figure is called the output coordinate system (OCS). For easier visualization the axes orientation for that system is shown in the bottom

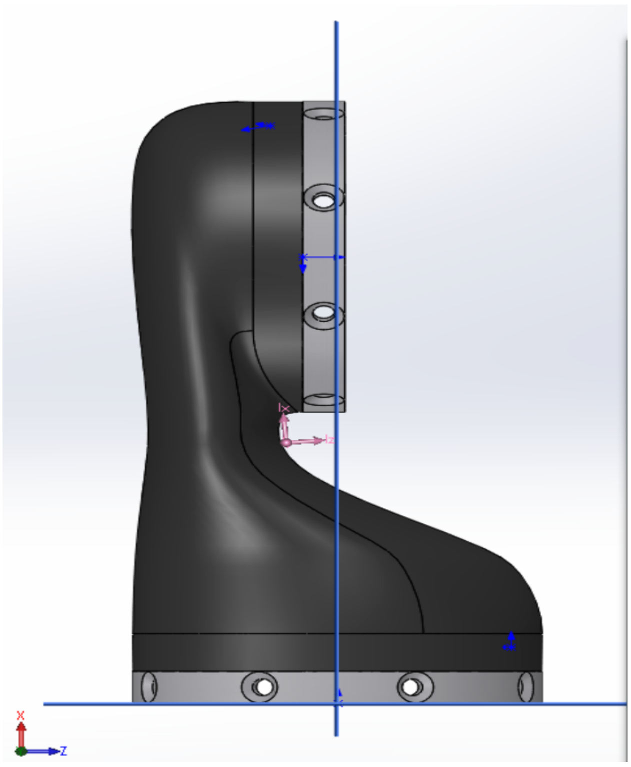
[2]refers to a system composed of a link + actuator

left corner of figure 3.5a. Each link and actuator can be described, dynamically speaking, in terms of their mass, center of mass and inertia tensor. The center of mass (COM) of a link/actuator i can be considered as a three dimensional vector with the position of the point in regards to a specific coordinate frame

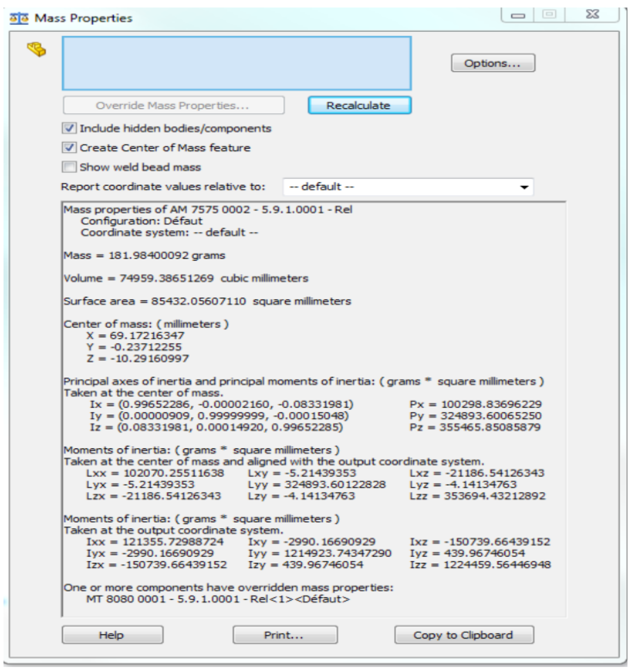
$$r_{l_i} = \begin{bmatrix} r_{l_{x_i}} \\ r_{l_{y_i}} \\ r_{l_{z_i}} \end{bmatrix} \tag{3.24}$$

The inertia tensor relative to the COM is given by

$$I_{l_i} = \begin{bmatrix} I_{l_{xx_i}} & I_{l_{xy_i}} & I_{l_{xz_i}} \\ I_{l_{yx_i}} & I_{l_{yy_i}} & I_{l_{yz_i}} \\ I_{l_{zx_i}} & I_{l_{zy_i}} & I_{l_{zz_i}} \end{bmatrix} \tag{3.25}$$



(a) COM and inertia tensor (colored in pink)



[✓] Include hidden bodies/components
[✓] Create Center of Mass feature
[ ] Show weld bead mass
Report coordinate values relative to: -- default --

```
Mass properties of AM 7575 0002 - 5.9.1.0001 - Rel
    Configuration: Défaut
    Coordinate system: -- default --

Mass = 181.98400092 grams

Volume = 74959.38651269  cubic millimeters

Surface area = 85432.05607110  square millimeters

Center of mass: ( millimeters )
     X = 69.17216347
     Y = -0.23712255
     Z = -10.29160997

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
      Ix = (0.99652286, -0.00002160, -0.08331981)        Px = 100298.83696229
      Iy = (0.00000909, 0.99999999, -0.00015048)         Py = 324893.60065250
      Iz = (0.08331981, 0.00014920, 0.99652285)          Pz = 355465.85085879

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 102070.25511638     Lxy = -5.21439353          Lxz = -21186.54126343
     Lyx = -5.21439353         Lyy = 324893.60122828      Lyz = -4.14134763
     Lzx = -21186.54126343     Lzy = -4.14134763          Lzz = 353694.43212892

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
     Ixx = 121355.72988724     Ixy = -2990.16690929       Ixz = -150739.66439152
     Iyx = -2990.16690929      Iyy = 1214923.74347290     Iyz = 439.96746054
     Izx = -150739.66439152    Izy = 439.96746054         Izz = 1224459.56446948

One or more components have overridden mass properties:
     MT 8080 0001 - 5.9.1.0001 - Rel<1><Défaut>
```

(b) Mass and inertia parameters.

Figure 3.5: Link 1 inertial parameters.

For demonstration purposes, most textbooks present examples of dynamic modeling of a robot in an idealized way, neglecting, for instance, the inertial parameters of the actuators. For a practical situation like this one, it is important to take into consideration all the major structural components of the arm in order to get the best representation possible of the dynamic parameters. In tables 3.5, 3.6 and 3.7, the mass of each link and actuator, as well as the plastic components that cover each actuator[3] are listed. Due to their position, and to simplify calculations, the plastic rings were integrated with the mass of the respective actuator

$$m_{mpr_i} = m_{m_i} + m_{pr_i} \tag{3.26}$$

where m_{mpr_i} is the combined actuator and plastic ring masses, whereas m_{m_i} and m_{pr_i} are the masses from tables 3.6 and 3.7, respectively.

Table 3.5: Links

Link	Mass (g)
1	181.984
2	423.993
3	211.004
4	68.96
5	68.96
6	727

Table 3.6: Actuators.

Actuator	Mass (g)
1	573.68
2	559.72
3	573.341
4	341.41
5	341.41
6	341.41

Table 3.7: Rings

Plastic Ring	Mass (g)
1	12
2	12
3	12
4	7
5	7
6	7

The dynamic parameters can be attained pursuing different methodologies. One can assume that each dynamic contribution of the links and actuators is calculated separately, having separate COMs and inertia tensors, or, due to the inherent linearity of these parameters [25], it is possible to combine an actuator and a link creating an augmented link. This work followed this approach.

Recalling figure 3.5, the given coordinates pertaining the COM are with respect to the OCS (refer to the reference frame in the bottom left corner). Of course, this is not ideal as the OCS is not common to all parts. The COM needs to be transformed in order to be with respect to the orientation of the DH coordinate system of the robot, the one illustrated in figure 3.3. The transformation is of the order

$$r'_{l_i} = Tr_{l_i} \tag{3.27}$$

where r'_{l_i} is the newly mapped COM vector, T the chain of transformations needed to match the orientation of the OCS of the link i to the correspondent DH coordinate frame i and

[3]the six gray covers in each actuator from 2.1a.

the position of the augmented COM to this coordinate frame, and r_{l_i} the coordinates of the COM in the OCS. The augmented link i, being composed of link i and actuator i, is formed by the combination of the newly mapped COMs, which is obtained via

$$r_{aug_i} = \frac{m_{l_i} r'_{l_i} + m_{mpr_i} r'_{m_i}}{m_{l_i} + m_{mpr_i}} \tag{3.28}$$

where r_{aug_i} is the COM of the augmented link, and r'_{m_i} the application of equation 3.27 to the actuators.

Analyzing figure 3.5b, different inertia tensors are given. The inertia tensor taken at the COM and aligned with the OCS, with the generalized notation

$$L_i = \begin{bmatrix} L_{xx_i} & L_{xy_i} & L_{xz_i} \\ L_{yx_i} & L_{yy_i} & L_{yz_i} \\ L_{zx_i} & L_{zy_i} & L_{zz_i} \end{bmatrix} \tag{3.29}$$

is the one used in this work, since it gives the possibility of directly applying the Huygens-Steiner theorem (also known as the parallel axis theorem), in order to calculate the inertia tensor referred to the augmented COM[4]

$$L_{a_i} = L_i + (m_{mpr_i} + m_{l_i}) S^T(r_{aug_i}, r'_{l_i}) S(r_{aug_i}, r'_{l_i}) \tag{3.30}$$

with,

$$r_{aug_i}, r'_{l_i} = r'_{l_i} - r_{aug_i} \tag{3.31}$$

where L_{a_i} is the inertia tensor L_i plus an increment due to the displacement of position between the COM of the link and the augmented COM. For clarity reasons, assume that ω translates the displacement shown in 3.31 The skew-symmetric operator S is of the form,

$$S = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} \tag{3.32}$$

Equation 3.31, establishes the augmented inertia tensor with the combined masses with regards to the contribution of the links. The aforementioned method might suffice as a first estimation of the dynamic model. However, as one can expect, such approach can be generalized to include the actuators as well, making the model slighter robust. By calculating both augmented inertia tensors referred to the newly augmented COM, the overall inertia tensor is simply the summation of both contributions

[4]refer to page 260 of [15] for detailed information about this theorem.

$$L_{t_i} = L_{a_i} + L_{r_i} \tag{3.33}$$

where L_{r_i} is the contribution wit respect to the actuator i, obtained following the same criteria as L_{a_i} (equivalent approach with equations 3.27, 3.28, 3.30 and 3.31).

In appendix B a table illustrates the derived dynamic parameters for the JACO^2.

3.3.3 Lagrangian Formulation

With the dynamic parameters already established, numerous approaches can be used to calculate the dynamic model itself. The literature commonly details the traditional methods briefly described in section 3.3.1, but there has been a growing trend in the development of symbolic robot modeling toolboxes [26] [27] [28] [29] that are both computationally more efficient and less strenuous for the user. Even so, this work is going to be based on the EL method, as it reasonably easy to understand and presents a good starting point to the analysis of the dynamic model estimation and its validity.

Recalling the explanations entailed in section 3.3.1, the EL method has its methodology based on the computation of the kinetic and potential energies that are acting on a manipulator. The computation relies on having all the contributions referred to the base frame which puts into evidence the computational cost of this method.

The kinetic energy of a manipulator is given in the general form as

$$\mathcal{T} = \sum_{i=1}^{n} (\mathcal{T}_{l_i} + \mathcal{T}_{m_i}) \tag{3.34}$$

being the summation of the contribution of each link and actuator. From equation 3.20, the mass (or inertia) matrix is given by

$$M(q) = \sum_{i=1}^{n} [(m_{mpr_i} + m_{l_i}) J_P^{(aug_i)T} J_P^{(aug_i)} + J_O^{(aug_i)} R_i L_{t_i} R_i^T J_O^{(aug_i)}] \tag{3.35}$$

where the Jacobian computation is adapted from equation 3.11

$$\begin{cases} J_{P_j}^{(aug_i)} = z_{j-1} \times (p_{aug_i} - p_{j-1}) \\ J_{O_j}^{(aug_i)} = z_{j-1} \end{cases} \tag{3.36}$$

where p_{j-1} is the position of the origin of the DH frame $j-1$ in base frame coordinates, p_{aug_i} is the COM of the augmented link with respect to the base frame and z_{j-1} is the unit vector of axis z of frame $j-1$. The Jacobian is then constructed in the following fashion

$$J_P^{(aug_i)} = [J_{P_1}^{(aug_i)} \quad \cdots \quad J_{P_i}^{(aug_i)} \quad 0 \quad \cdots \quad 0] \tag{3.37}$$

$$J_O^{(aug_i)} = [J_{O_1}^{(aug_i)} \quad \cdots \quad J_{O_i}^{(aug_i)} \quad 0 \quad \cdots \quad 0] \tag{3.38}$$

Recall that the augmented inertia tensor deduced in the previous section is of constant nature. The similarity transformation depicted on equation 3.35 shown by the term $R_i L_{t_i} R_i^T$ represents the mapping of the inertial contribution of each augmented link with respect to the base frame[5].

The Coriolis term, $C(q, \dot{q})$ is often calculated based on the Christoffel symbols technique, alternatively, with the derivative of the inertia matrix

$$C(q, \dot{q}) = \dot{M}(q) - \frac{1}{2}\dot{q}^T \frac{\partial M(q)}{\partial q} \tag{3.39}$$

The calculation of this term is very costly computation-wise. Generally, it can be dropped as its contribution, in the majority of the cases, is negligible. The experimental tests pursued with the JACO^2 were done with low speed motions, making this term even less contributive. Consequently, for the purposes of this work, this term was not considered.

The potential energy can be extended to include the contribution due to the actuators

$$\mathcal{U} = \sum_{i=1}^{n} (U_{l_i} + U_{m_i}) \tag{3.40}$$

which following some algebraic manipulation and with the help of equation 3.21

$$\mathcal{U} = -\sum_{i=1}^{n} (m_{l_i} g_0^T p_{l_i} + m_{mpr_i} g_0^T p_{m_i}) \tag{3.41}$$

where p_{m_i} is the COM of actuator i referred to the base frame, whereas the gravity acceleration in the base frame is denoted as $g_0 = [0 \quad 0 \quad -9.81]^T$. The gravity term $g(q)$ is then calculated deriving equation 3.41 with respect to each angle variable,

$$g(q) = \frac{\partial \mathcal{U}(q)}{\partial q} \tag{3.42}$$

With these assumptions, the inverse dynamic equation entailed in the expression 3.23 can be calculated.

[5]The absence of superscript means that the it with reference to the base frame.

Chapter 4

Control Architectures

In the previous chapter, the robot model identification for the JACO^2 was outlined. Of course, by being a practical implementation, these models represent an estimate and need to be analyzed experimentally to infer their viability for real time applications. For that reason, different control architectures were tested. This chapter delves into this, exposing the theory behind them and explaining the different procedures that were tested for the manipulator in study.

4.1 Operational Space Control

In the following subsections, the task space control architectures developed and tested in the JACO^2 are detailed. First, a simple operational space control in the task space is presented with the application of the Jacobian transpose. Then, a null space control is integrated and analyzed.

4.1.1 Jacobian Transpose Control

In section 3.2, it was described the Jacobian as the relation between the joint velocities and the end effector velocities. Notwithstanding, the same principle can be applied to the static torque/force relationship, by the concept of virtual work

$$\delta w = F^T \delta X - \tau^T \delta q \tag{4.1}$$

where δX and δq are small displacements in end effector position and joint angles, caused by a force F. These changes in position are mathematically still related by a Jacobian

$$\delta X = J(q)\delta q \tag{4.2}$$

Hence, with equations 4.2 and 4.1, and assuming that the manipulator is in equilibrium, then $\partial w = 0$, which yields

$$\tau = J(q)^T F \tag{4.3}$$

Equation 4.3 directly relates the applied forces at the end effector and the resulting joint torques with the transpose of the Jacobian. This results in an important relation for motion control. Normally, if the robot is tasked with the operation of performing some specific motion in the environment where precision movements from the end effector are a must then an operational space control scheme might suit ones needs.

The basis for the Jacobian transpose control lies on the real time definition of a desired trajectory/position (task space) while the robot performs the necessary joint movements to fulfill the task (joint space). A control law based in a PID control can be defined in order to meet these criteria.

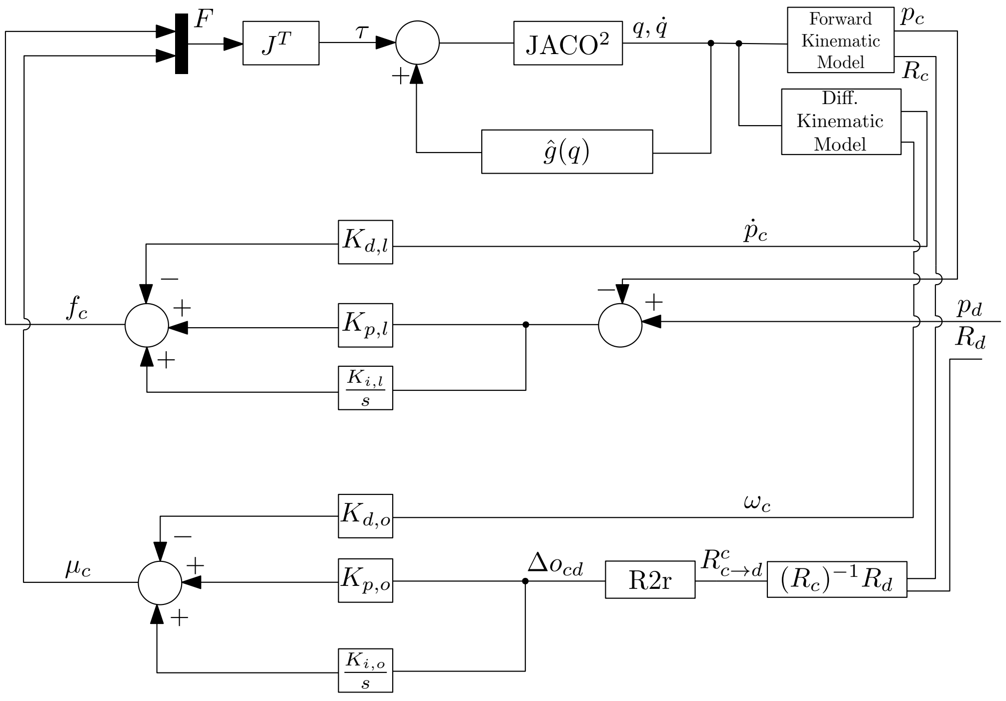


Figure 4.1: Block scheme of the Jacobian transpose control.

In figure 4.1 one such example is illustrated. The forward kinematic model is obtained via the Kinova API with the help of the function **MyGetCartesianPosition**, which gives the position and orientation (in Euler angles with XYZ convention) of the end effector referred to the base. They are represented in the control scheme by p_c and R_c. The differential kinematics model is obtained from equations 3.6, 3.15 and 3.16. K_P, K_I and K_D are positive diagonal matrices corresponding to the proportional, integrative and derivative terms for both position and orientation.

4.1.2 Orientation Control

The block diagram displayed in the previous section shows that the orientation control is performed differently than the position control. The orientation can be expressed with three Euler angles, which translate into the nine elements present in a rotation matrix of a homogeneous transformation matrix (acknowledged in equation 3.2). Considering that the Euler angles retrieved from **MyGetCartesianPosition** are set as $\varphi = [\alpha \quad \beta \quad \gamma]^T$, then following the XYZ convention [30]

$$R(\varphi) = R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} c_\beta c_\gamma & -c_\beta c_\gamma & s_\beta \\ s_\alpha s_\beta c_\gamma + c_\alpha s_\gamma & -s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & -s_\alpha c_\beta \\ -c_\alpha s_\alpha c_\gamma + s_\alpha s_\gamma & c_\alpha s_\beta s_\gamma + s_\alpha c_\gamma & c_\alpha c_\beta \end{bmatrix} \tag{4.4}$$

where $R_x(\alpha)$, $R_y(\beta)$ and $R_z(\gamma)$ are the rotation matrices about each of the three coordinate axes. There are various ways to minimally represent the orientation of the end effector: Euler angles, angle-axis representation, or quaternions. With the angle-axis representation and taking as a basis the work developed in [31], let $R^c_{c\to d}$ be the computation of the rotation from the current orientation to the desired one, from the coordinate frame of the end effector

$$R^c_{c\to d} = (R^0_c)^{-1} R^0_d \tag{4.5}$$

where R^0_c and R^0_d are the current and desired end effector orientation with respect to the base frame. The angle-axis representation can be parameterized as

$$\begin{cases} \theta = \cos^{-1}(\frac{R_{11}+R_{22}+R_{33}-1}{2}) \\ \nu = \frac{1}{2\sin(\theta)} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \end{cases} \tag{4.6}$$

where R refers to a rotation matrix like $R^c_{c\to d}$. The axis is represented by the three dimensional vector ν while the angle of rotation along that axis is referred to θ. The orientation displacement between the current and desired orientations is then given by

$$\Delta o_{cd} \equiv r \equiv \theta \nu \tag{4.7}$$

Clearly, this method is prone to singularities when θ is approximately zero or π. To avoid this, a threshold of 0.03 rad is assumed where the range of angles up to this value are neglected and the orientation error is assumed to be zero, or below the π minus this threshold, where the angle values are not considered, and the error is instead simply the maximum accepted π-threshold. Similar to the position control, a PID control can be applied to minimize the orientation error. The addition of the integrative component, not usually suggested in the literature, proved to be essential. Its calculation, for an angle-axis parametrization was defined for an instant k as

$$\int_{k-1}^{k} \Delta o_{cd} = (r_{k-1} \cdot \nu_k + \theta_k)\nu_k \tag{4.8}$$

Alternatively, a composition of two successive angle-axis rotations obtained via application of the Euler-Rodrigues parameters [32] yields the same result. In figure 4.1 a block named R2r is displayed. This is responsible for the calculation of equation 4.7. Lastly, R_d is calculated with the desired Euler angles which are then inputed in equation 4.4.

4.1.3 Null Space Control

The scheme displayed in figure 4.1, shows the simultaneous control of the translation and orientation of the JACO^2, by considering exclusively its geometry. Experimental tests suggested that the acting forces for the tracking of the desired position and orientation often generated coupling effects between each other resulting in unwanted disturbances and decreasing the overall performance of the system. Such a problem can be minimized by exploring the concept of null space control. The idea lies on the establishment of a primary goal, responsible for a specific controller, like the position control, and a secondary goal, operating in the null space of the primary controller. Thus, the overall torque sent to the manipulator is then

$$\tau = \tau_t + \tau_N \tag{4.9}$$

where τ_t is the torque related to the primary goal and τ_N is the null space term entailing the filtered torque correspondent to the secondary goal. Therefore, let the position control be defined as the primary task, and the orientation control the secondary task. Equation 4.9 breaks down into,

$$\tau = J_v^T f_c + (I - J_v^T J^{\#T})\tau_0 \tag{4.10}$$

where $J^{\#}$ is the generalized inverse of J_v and the matrix $(I - J_v^T J^{\#T})$ projects the vector τ_0 into the null space of J_v

$$J^{\#} = W^{-1} J_v^T (J_v W^{-1} J_v^T)^{-1} \tag{4.11}$$

where $W = M(q)$ if task space dynamics are considered. For now, let $W = I$

$$J^{\#} = J_v^T (J_v J_v^T)^{-1} \tag{4.12}$$

which entails a geometric based approach. The $J^{\#}$ is then simply the Moore-Penrose pseudo-inverse. Analyzing the control scheme pictured in figure 4.1, equation 4.10 is specified as

$$\tau = J_v^T f_c + (I - J_v^T J^{\#T}) J_\omega^T \mu_c \tag{4.13}$$

where J_v and J_ω are the 3×6 submatrices of J (refer to equation 3.6), I is a 6×6 identity matrix. Figure 4.2 depicts the application of the null space control.

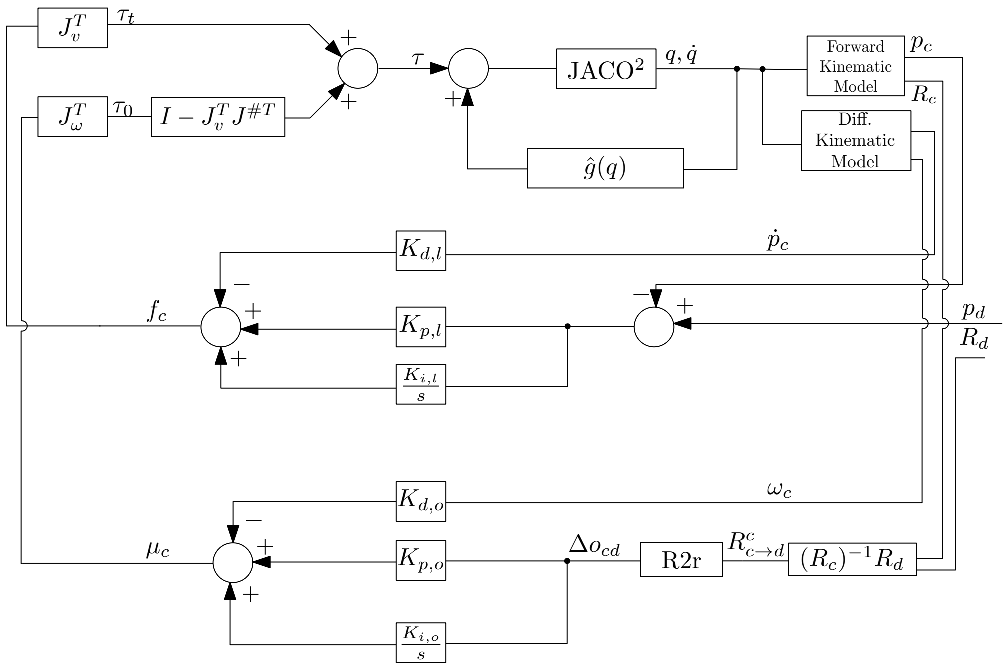


Figure 4.2: Task space control with geometric null space implementation.

Results with the so-called dynamically consistent generalized inverse, that is with the application of $W = M(q)$ are also going to be analyzed with an equivalent control scheme.

4.2 Computed Torque Control in the Joint Space

The previous control techniques were built applying the kinematic and differential kinematic models, without any attention to the dynamics of the manipulator. This can be enough for general task operations whose precision does not need to be very high. However, including the dynamic model into the control design generally improves the stability, performance and accuracy during task operations as long as its estimations are approximate to the real model. Additionally, there is the significant advantage of making the system linearized, reducing the coupling effects described in the last section. Thus, taking as a starting point equation 3.23, then the generalized torque τ can be assumed as

$$\tau = \tau_c - \tau_f - \tau_e \tag{4.14}$$

where τ_c , τ_f and τ_e are respectively the computed, friction and external torques acting on the manipulator, which by directly substitution yield

$$\tau_c - \tau_f - \tau_e = \hat{M}(q)\ddot{q} + \hat{C}(\dot{q}, q) + \hat{g}(q) \tag{4.15}$$

where $\hat{M}(q)$, $\hat{C}(\dot{q}, q)$, and $\hat{g}(q)$ are the estimations of the dynamic model obtained from the implementations outlined in sections 3.3.2 and 3.3.3. Throughout the work, external and friction torques were not considered[1]. Thus, equation 4.15 is denoted as

$$\tau_c = \hat{M}(q)\ddot{q} + \hat{C}(\dot{q}, q) + \hat{g}(q) \tag{4.16}$$

Equation 4.16 illustrates a system nonlinearly dependent upon the joint positions and velocities. Due to this, feedback linearization and decoupled control are applied in order to cancel nonlinear effects. Hence, the system is linearized using a nonlinear feedback law of the fashion

$$\tau_c = \hat{M}(q)w + \hat{C}(\dot{q}, q) + \hat{g}(q) \tag{4.17}$$

where $w = \ddot{q}$ is the new control variable. Equation 4.17 is the inverse dynamic control or also commonly referred as computed torque control. As can be easily seen, the direct application of this control law can be used for joint space control techniques.

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

[1]This will be a target of further discussion in chapter 5.

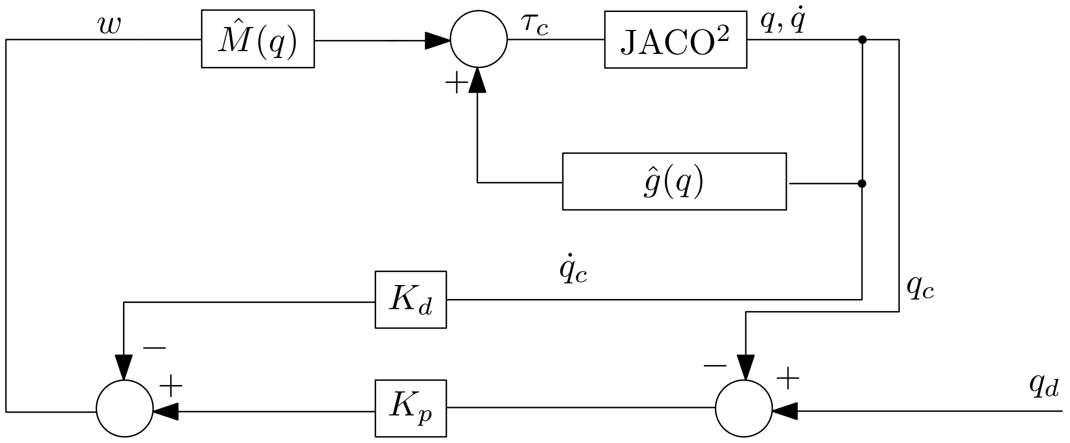


Figure 4.3: Joint Space Control with Dynamic Model.

Each of the six joints are independently controlled with the proportional-derivate terms, which gives a broader perspective of the overall performance of the robot. The inertia matrix $\hat{M}(q)$ and the gravity term $\hat{g}(q)$ are computed in real time with the control variable w being

$$w = K_p(q_d - q_c) - K_d\dot{q}_c \tag{4.18}$$

where from $w = \ddot{q}$, the closed-loop system dynamics is given by

$$\ddot{q} + K_d\dot{q} + K_p q_c = K_p q_d \tag{4.19}$$

which yields a exponentially stable solution if K_p and K_d are properly chosen. The schematic from figure 4.3 depicts the described control architecture.

4.3 Joint Space Control with Task Posture Reference

In the previous sections different operational space control schemes were addressed. Although acceptable results can be attained using these architectures, their control gain design is limited. This is because, experimentally, the same control gain was assigned to every value of the diagonal of each matrix. If done otherwise, the system would behave erroneously. The reason for this pertains to the fact that the control gains are applied directly to the task space errors.

For this reason, a new approach must be followed. Although, the last section already introduced the topic of controlling each joint individually, it is not very useful for task operation purposes. Consequently, another approach must be devised so that the control of the system is improved. In this section, such approach is presented as an adaption of works [33] [34].

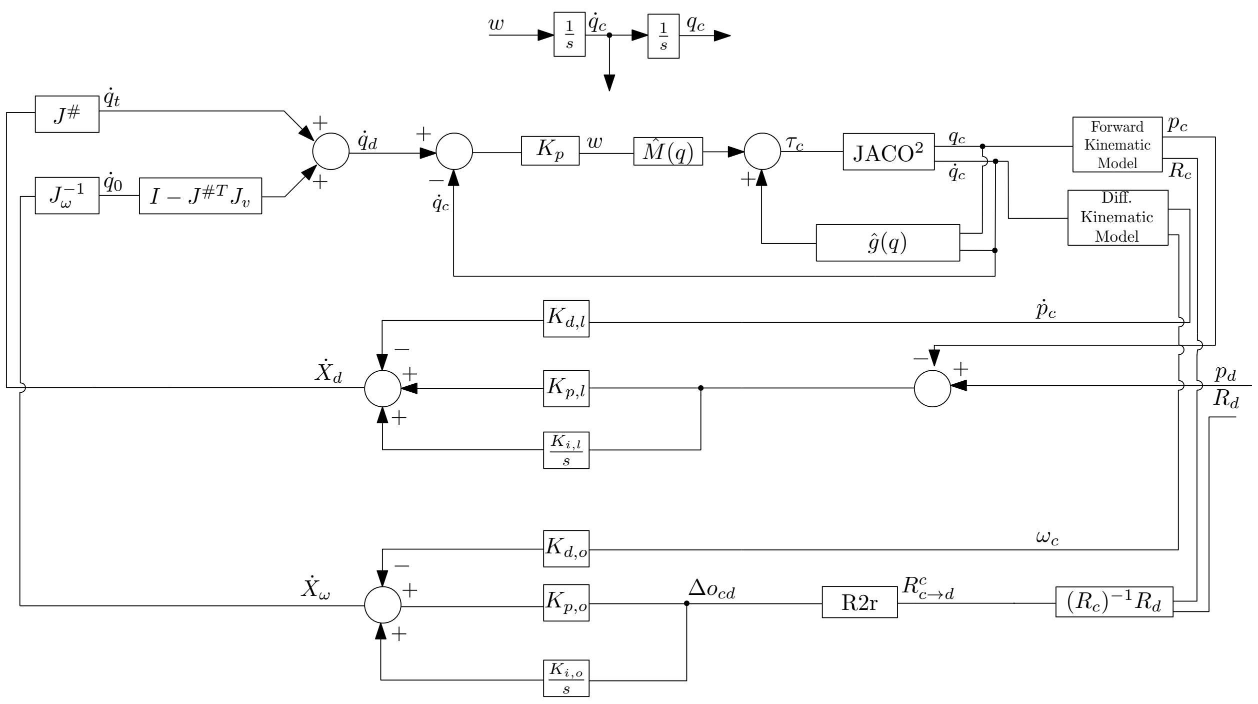


Figure 4.4: Joint Space Control with Task Space Reference.

In short, a task space reference is assumed, but the posture errors are mapped into joint velocity references which ensure a joint-based control. Additionally, a null space joint velocity control is added to the system. As a starting point, let $\dot{q}_0$ be the null space velocity vector pointing to the joint position error. Then, the desired joint position q_d is directly obtained by integrating $\dot{q}_d$ resulting in,

$$\dot{q}_d = J^{\#}\dot{X}_d + (I - J^{\#}J_v)\dot{q}_0 \tag{4.20}$$

with the Moore-Penrose pseudo-inverse $J^{\#}$ obtained from equation 4.12. Like the computed torque control with the null space orientation control in section 4.1.3, the concept of null space is also applied, but now with respect to the joint velocity reference. From equation 4.20 $\dot{X}_d$ is the desired task space linear velocity which is easily computed from desired and current positions, although experimentally the fully outer-loop PID control was also considered. The null space velocity vector $\dot{q}_0$ is obtained from the application of the inverse differential kinematics model

$$\dot{q}_0 = J_\omega^{-1}\dot{X}_\omega \tag{4.21}$$

with the following premise

$$\dot{X}_w \propto \Delta o_{cd} \tag{4.22}$$

With these assumptions, and resorting to equation 4.16, the control variable $w = \ddot{q}$ that allows a linear and decoupled control is simply known as

$$\ddot{q} = \dot{q}_d - \dot{q}_c \tag{4.23}$$

where $\dot{q}_c$ is obtained from the calculations delved in section 3.2.1. In figure 4.4 the proposed control scheme is illustrated. The inner joint velocity control is composed of a single proportional control gain K_p with size 6×6 whose diagonal elements are set independently for each joint. Normally, proportional-derivative (PD) controllers are designed for such control systems, but after some tests, it was verified that the derivative component was introducing too much noise and instability to the system, mainly due to the high fluctuation inherent to the sudden rate of change of joint velocities.

Chapter 5

Experimental Results

To evaluate the control architectures presented in the previous chapter and the derivation of the geometric, differential and dynamic models of the JACO^2 6 DOF robot described in chapter 3, experiments are carried out with the robotic arm, specifically by performing a planned trajectory. Before discussing the results achieved, a brief description must be done regarding the experimental setup, the different internal adjustments that had to be done to the robot, and the experiment itself.

5.1 Setup

Before attaining the results shown in the next sections, the robot had to be studied in order to understand what type of controllers where integrated on its system. Since the work revolves around computed torque techniques, the torque control console (figure 2.3) provided by Kinova was initially studied. Initial experiments suggested that the manipulator was under specific control mechanisms that could become an unwanted variable in the computed torque architectures devised in this work.

Naturally, if one desires to develop control architectures like the ones described in chapter 4, it is relevant that internally the robot has the least amount of controllers active, to minimize those external factors. In table 5.1 are displayed the Kinova API functions that were used as a setup for the experimental tests, and that allowed the minimization of some of the internal control mechanisms. In short, the most important functions were **MySetGravity-OptimalZParam** and **MySetTorqueActuatorGain**. The former granted the elimination of the gravity compensation imbued in the robot. The latter improved the arm's behavior during torque control, by apparently omitting some control branches present in the DSP of the arm.

Table 5.1: Setup Parameters.

Function	Description	Value
MySetTorqueSafetyFactor	If the velocity of an actuator gets to a specific threshold the robot stops and changes back to trajectory control. Feature that prevents the robot from taking high speed motions. Setting to 1 disables the feature.	1
MySetTorqueVibrationController	Vibration observer/controller to eliminate vibrations during contact with stiffness environments. Adjust from 0 to 1, to enable or disable vibrations.	1
MySetTorqueActuatorDamping	Set actuators damping gain.	0
MySetGravityType	Set the gravity type: either manual or optimal. With manual the user can specify each mass of the link and COM , while in optimal mode the robot performs a task to obtain the best set of parameters.	Optimal
MySetGravityOptimalZParam	Function used to set the gravity parameters using the optimal mode.	0
MySetTorqueActuatorGain	Set the actuators feedback gain	0

Before initiating the experimental tests, it is important to calibrate the torque sensors by setting them to zero, while the arm is positioned in a configuration where gravity does not influence the joint torques. The position is suggested in the advanced specification guide as being $q_{calib} = [*, 180, 180, 0, 0, 180]$ with $*$ being any joint value in degrees desired. Likewise, the position of the actuators is also calibrated due to the common displacement occurred when the actuators are suddenly stopped. For both calibrations, functions **My-SetTorqueZero** and **MySetJointZero** are used, respectively. The commanded torque τ_c is sent to the robot with the function **MySendAngularTorqueCommand**.

For all the results illustrated on this chapter, an operational space trajectory was planned to infer the tracking capabilities of the robotic arm. The trajectory is a third degree polynomial (spline) denoted as

$$X(t) = X_0 + \frac{3}{t_f^2}(X_f - X_0)t^2 - \frac{2}{t_f^3}(X_f - X_0)t^3 \tag{5.1}$$

where $X(t)$ represents the end effector pose vector for the instant t, X_0 the initial pose,X_f the final pose and t_f is the duration of the trajectory,

$$\begin{cases} X_0 = \begin{bmatrix} 0.212444, & -0.257293, & 0.506643, & 1.65789, & 1.11875, & 0.113634 \end{bmatrix}^T \text{ (m)} \\ X_f = \begin{bmatrix} 0.312452, & -0.407218, & 0.706502, & 2.44364, & 0.0715527, & 0.637327 \end{bmatrix}^T \text{ (m)} \\ t_f = 16 \ (s) \end{cases} \tag{5.2}$$

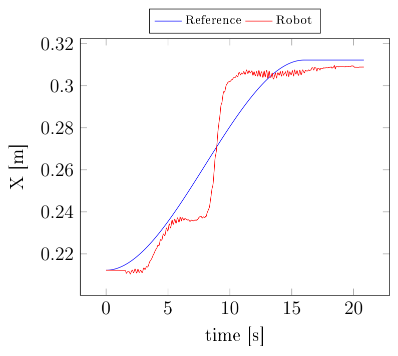
The initial pose refers to the HOME position/orientation of the robot, while the final pose was set as to stress the robot into performing difficult motions to better ascertain the reliability of the estimated models (refer to appendix D for a more clear observation of both the initial and final poses).

5.2 Task Space Control

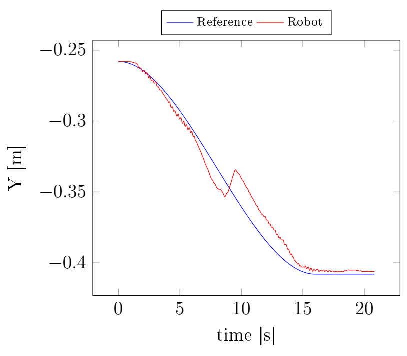
The first operational space control, called the Jacobian transpose control was detailed on section 4.1.1. This control resorts solely on the geometry of the arm, which might not grant the best possible tracking behavior. The 6×6 diagonal positive-definite gain matrices are listed in table 5.2.

Table 5.2: Control Gains.

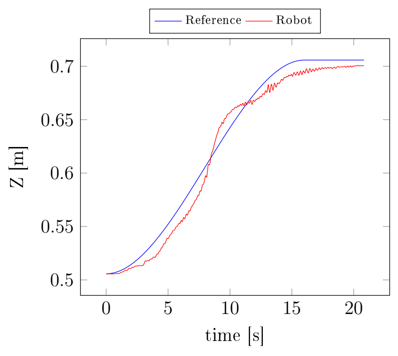
Position	Gain	Orientation	Gain
$K_{p,l}$	400	$K_{p,o}$	20
$K_{i,l}$	150	$K_{i,o}$	8
$K_{d,l}$	25	$K_{d,o}$	1.5



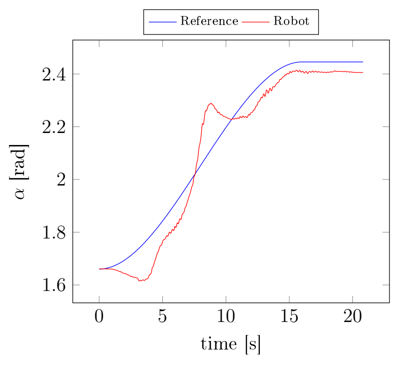
(a) X Position.



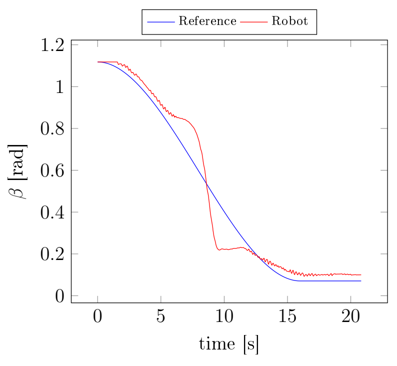
(b) Y Position.



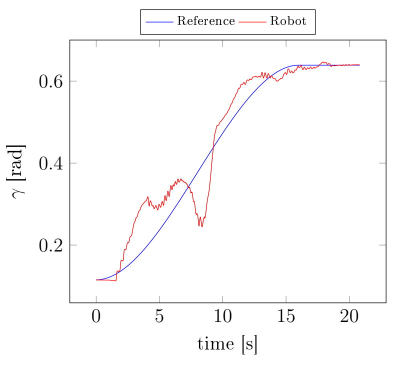
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

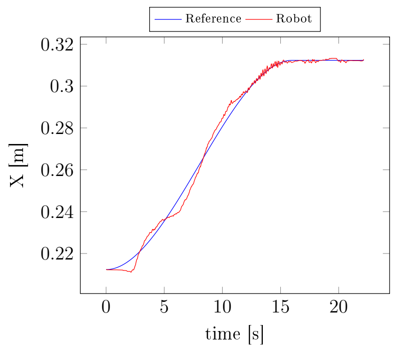
Figure 5.1: Tracking with Jacobian transpose control.

As can be seen from the figure 5.1, the steady state error is acceptable. The figures show the tracking of position in meters and the orientation expressed in radians through the nomenclature evoked in section 4.1.2. Prior to the addition of the integrative component, the steady state error was difficult to minimize especially regarding the orientation. The derivative term, which regulates the rate of error change by acting as a dampener, could not be increased too much, as the robot would rapidly start to perform oscillatory motions. The reasons behind this are not completely understood, although it might suggest that the robot has some degree of damping effect associated to its actuators which was still noticeable even after applying the commands shown in table 5.1. With the control gains expressed in table 5.2 the results are conclusive. Observing figures 5.1a, 5.1b and 5.1c one can see that although the tracking is initially reasonably acceptable, the adjustment of the end effector orientation as displayed in figures 5.1d, 5.1e and 5.1f causes a position displacement with respect to the reference, at around 10 seconds. This suggests the existence of force coupling, from both the position control and orientation control, acting on the robot while it reaches the desired pose. That is particularly noticeable in 5.1f which shows oscillatory motion around the interval [2.5,7.5] (s). This, naturally, is made more noticeable due to the strenuous trajectory that is imposed, which in terms of orientation requires a change of 45° along X, 60° along Y and 30° along Z.

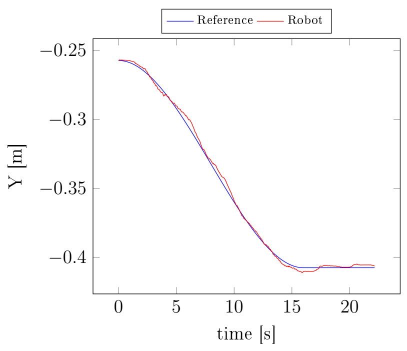
To minimize this, attempts were made by adjusting the control gains accordingly, and despite being possible, the minimization of the steady state error would be compromised. Thus, a balance was met using these control gains, which best combined both considerations. Alternatively, a null space approach might improve the performance of the robot and reduce these problems.

5.3 Null Space Orientation Control

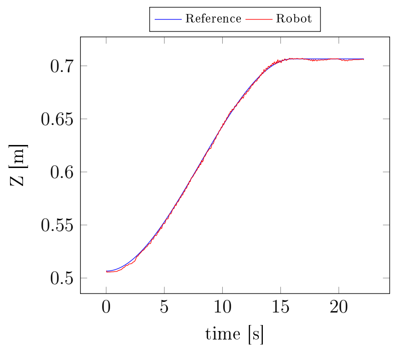
In the last section the Jacobian transpose control was analyzed, and its main drawbacks as a torque control technique were discussed. Fortunately, an alternative approach can be designed allowing both position control and orientation control to work simultaneously without interference between them. In section 4.1.3, the concept of null space was introduced, which ensures the application of multiple controllers at once. In this case, the orientation control is defined as the secondary task, while the position control is the primary task. This means that the orientation control operates in the null space of the position control. With this, the commanded torque apart from being composed by the control signal of the position control has also a filtered component resulting from the null space of the position control which will not interfere with it. Obviously, with this requirement the desired orientation might not be reached. Applying the same control gains from table 5.2, the control architecture of figure 4.2 generates the results shown in figure 5.2.



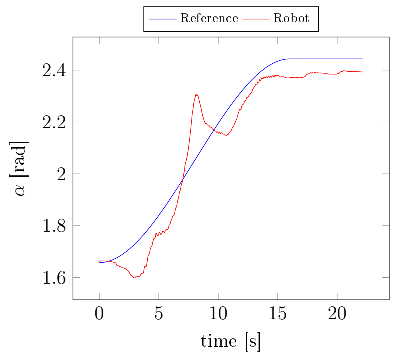
(a) X Position.



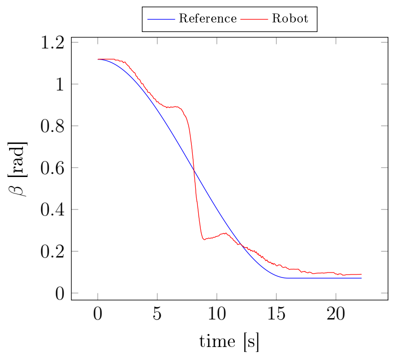
(b) Y Position.



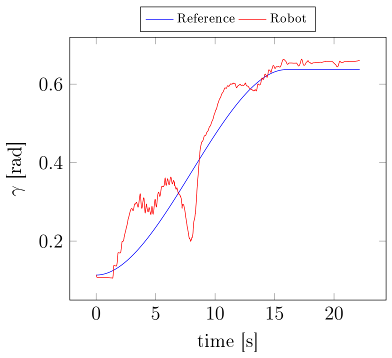
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.

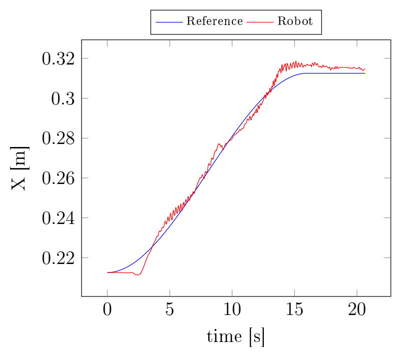


(f) Z Orientation.

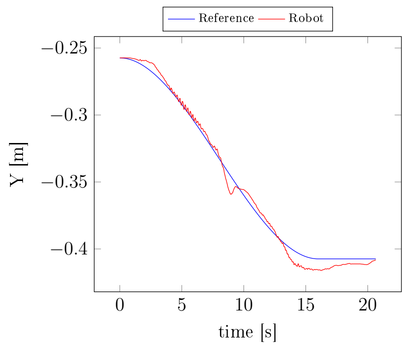
Figure 5.2: Tracking with Null Space Orientation Control.

Clearly, it is possible to observe from figures 5.2a, 5.2b and 5.2c that the tracking accuracy with respect to the primary task, is improved comparing with figures 5.1a, 5.1b and 5.1c. The secondary task, displayed in figures 5.2d, 5.2e and 5.2f, despite being a filtered signal, holds a similar response to the previous controller. Still, it is worth reminding that this control architecture relies upon the Moore-Penrose pseudo-inverse, which recalling equation 4.11, means that $W = I$. As one can expect, this is not ideal, as only the geometry of the robot is considered. Ideally, $W = M(q)$ where the mentioned equation entails the dynamically consistent generalized inverse. By including the dynamic parameters of the robot, it is expected that better results can be achieved, and one can safely assume that no secondary task forces interfere with the primary task forces. As a result, with $W = M(q)$ the system behaves as shown in figure 5.3. The control gains for this test are the same (refer to table 5.2).

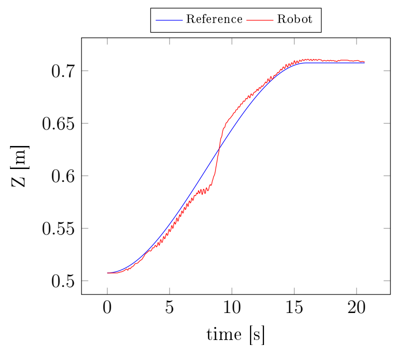
It is expected that the null space implementation has an increase of performance by



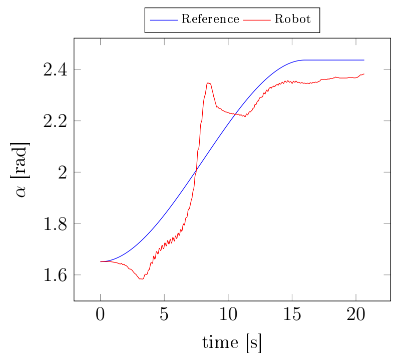
(a) X Position.



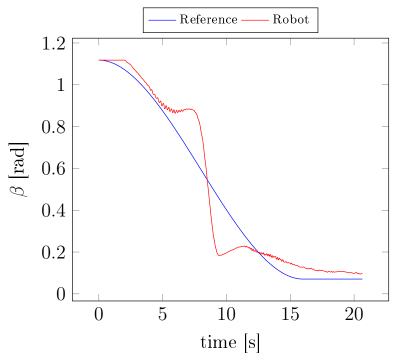
(b) Y Position.



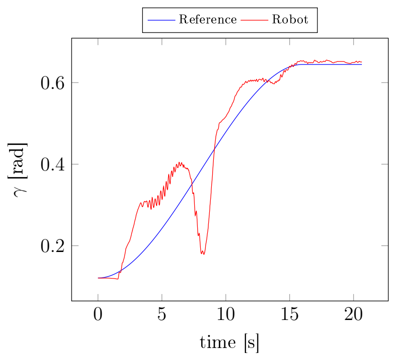
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

Figure 5.3: Tracking with dynamically consistent generalized inverse.

assuming the dynamically consistent generalized inverse. Analyzing figures 5.3a, 5.3b and 5.3c with their counterparts from figure 5.2, it can be seen that although the overall steady state error is still within acceptable margins, the performance has not increased as it was expected, being practically the same.

Naturally, this lack of improvement might be due to uncertainties and estimation errors related to the derived dynamic model. Further discussion pertaining this is made throughout the remaining chapter.

5.4 Dynamic Control in the Joint Space

In section 4.2, a simple PD controller is described (figure 4.3) which is purposedly used as a quick way to evaluate the derived dynamic model. This was done, mainly because the calculations surrounding the dynamic model identification of a general robot are strenuous and easily error-prone. Consequently, it is important to start from the simplest control architecture possible, and then build up towards more complex systems based on an already well known model. In the same way as the geometric and kinematic-based control architectures shown in the previous sections, a desired trajectory is defined. Hence, for each joint a sinusoidal wave is generated with the form

$$q_d = A\sin(\omega t) \tag{5.3}$$

where the angular frequency $\omega = \frac{2\pi}{T}$ with T being the period of the trajectory which is set to 8 seconds, and amplitude A of 20 degrees.

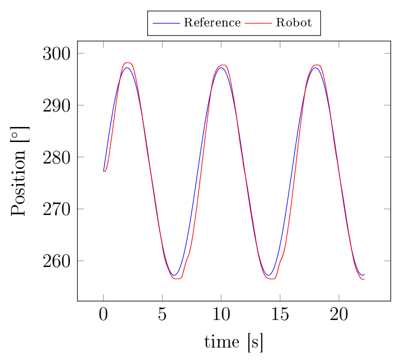
Table 5.3: Control Gains.

Joint i	K_{p_i}	K_{d_i}
1	32.5	12.25
2	40.5	15.25
3	55	22.5
4	65	28.5
5	70	30.5
6	850	150

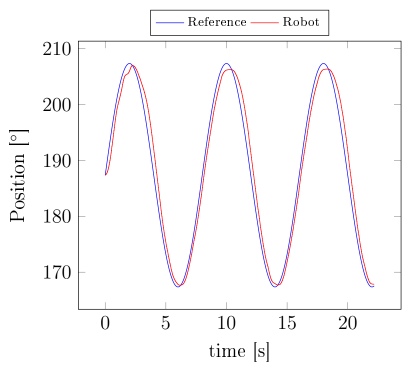
From a predefined position in the environment, all joints are set simultaneously with the desired trajectory in order to boost motion stress, and to draw broader conclusions. In figure 5.4 the results are shown with the proportional-derivative gains detailed in table 5.3. As can clearly be observed, the joints follow the desired motion with relatively good tracking response performing stable movements during the trajectory. This suggests that the proposed dynamic model is reasonably well estimated. The individualized joint control grants a wider management and adjustment to how the robot should operate given a specific task. Experimentally, with the application of the aforementioned gains it was also visible that the coupling effect between the joints was low, which further suggests that the system is approximately linearized and provides additional evidence into the dynamic model validity and viability.

Although the depicted results are conclusive enough to draw these assumptions, it is important to refer that the dynamic model is calculated with the inertial parameters described in the CAD model of the manufacturer. This is well known for not producing the best estimations possible, since the CAD models are known to be prone to errors due to the fact that the geometry of the links is usually complicated to define precisely and certain structural parts of the robot are often dismissed [5].

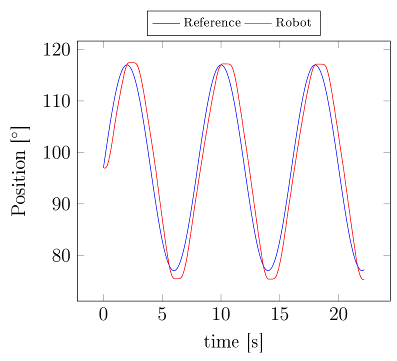
Also, it was observed amid the experimental tests, that a considerably higher proportional gain had to be set for the last joint, so that it could execute the required sinusoidal trajectory



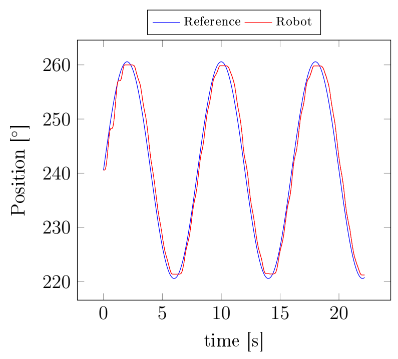
(a) Joint 1.



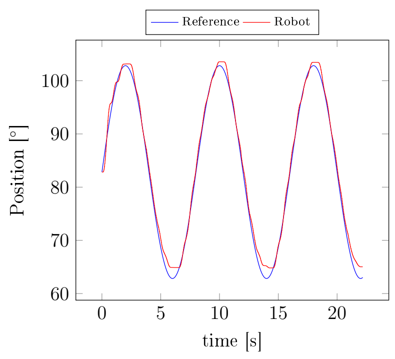
(b) Joint 2.



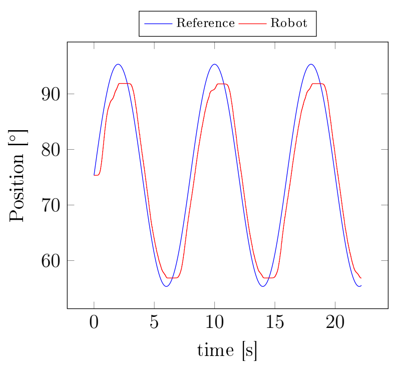
(c) Joint 3.



(d) Joint 4.



(e) Joint 5.



(f) Joint 6.

Figure 5.4: Tracking with dynamic control in the joint space domain.

to the end effector. The increase of this control term was met with ease, and no discernible coupling effects were seen. Still, one has to wonder why such deviant control gain is required compared with the remaining joints. Various hypotheses can be conceived as a reason for that. On one hand, the end effector base inertial parameters provided in the CAD model might be underestimated, which consequently makes the model under evaluate the necessary torque command to be sent to the last actuator, and despite being a legitimate reason, it is unlikely to be the real cause. On the other hand, it is also possible to be due to assumed estimations during the calculation of the dynamic parameters, or in other words, inconsistencies with the derived dynamic model. This last claim might be reasonably fair as the estimation of the model is easily mistakable, but if that were the case, propagation effects due to this increased control gain would have been made more visible to the other joints, making the system generally unstable during task operation and prone to small jerky motions.

5.5 Joint Space Control with Task Space Reference

In the previous section, the proposed dynamic model was analyzed and discussed and its veracity inferred. Naturally, the joint control is not the most practical way of making the robot perform a specific task operation. Therefore an alternative approach is suggested. As detailed in section 4.3 with the control scheme of figure 4.4, a task space reference is designed similarly to the task space controllers previously shown, with the resulting task space errors being converted into joint velocity references, by virtue of the known differential kinematic relation. This conduces to an individualized control of each joint, just like the controller of the last section, but having the advantage of possessing a task space reference.

Table 5.4: Joint Velocity Control Gains.

Joint i	K_{p_i}
1	60.5
2	70.5
3	87.5
4	90
5	90
6	600

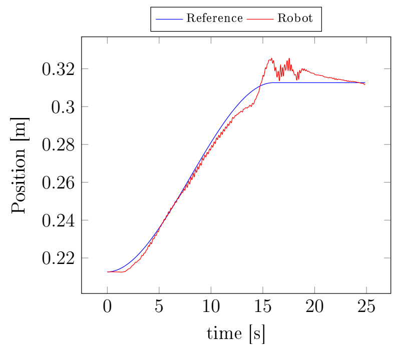
Table 5.5: Task Space Control Gains.

Position	Gain	Orientation	Gain
$K_{p,l}$	80	$K_{p,o}$	25
$K_{i,l}$	20	$K_{i,o}$	10
$K_{d,l}$	5	$K_{d,o}$	3.5

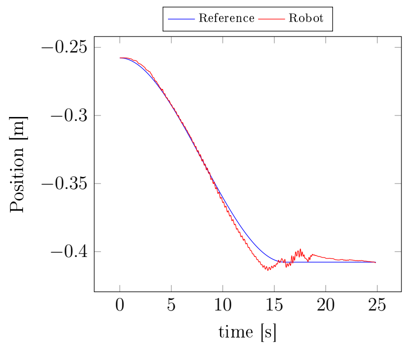
To improve the performance of the system a null space velocity component is defined, relying on the geometry of the robot. The control gains for both the task space control and consequent joint space velocity control are shown in tables 5.4 and 5.5. The system response with this controller, is shown in figure 5.5. Detailed observation of figures 5.5a, 5.5b and 5.5c show that the desired position is practically achieved around the end of the trajectory (16 seconds).

By virtue of the null space implementation, the position control takes precedence over orientation control therefore assuming an adequate response. Nonetheless, small disturbances are still experienced during the position control, around the interval [15,20] seconds.

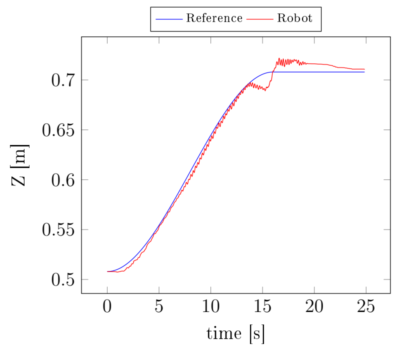
The secondary task denoted by figures 5.5d, 5.5e and 5.5f, is adjusted in the latter stages of the trajectory motion during the aforementioned time interval. Even though, there are still small non filtered torque commands interfering with the position control, as it was just referred. The sudden readjustment of orientation might be the cause for such occurrence.



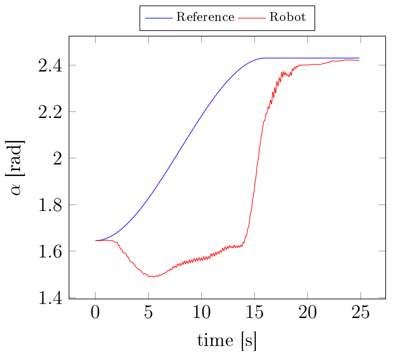
(a) X Position.



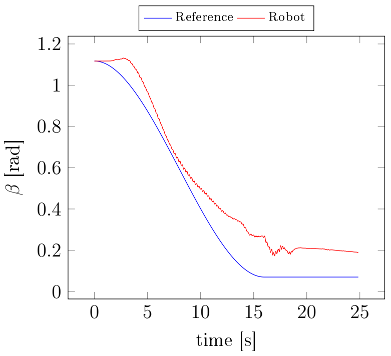
(b) Y Position.



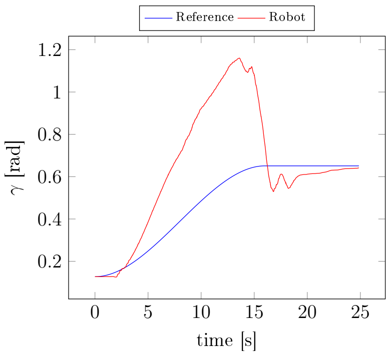
(c) Z Position.



(d) X Orientation.



(e) Y Orientation.



(f) Z Orientation.

Figure 5.5: Joint Space Control with Task Posture Reference.

Observing with more detail figures 5.5d and 5.5f, one can easily see that the tracking lefts much to be desired in the first stages of the task operation. Attempts were made to add an integrative term to the inner joint velocity control to tackle this issue, since the derivative term was found to be too noisy and unstable. Unfortunately, no advantages were obtained with it, as coupling effects started to be felt. For that reason, this term was also dropped remaining only a proportional term to control the inner joint velocities.

Moreover, the last section already discussed the possibility of existing small inconsistencies in the derived dynamic model. In fact, as it was considered initially, the addition of the dynamic model of the robot should increase its accuracy regarding the execution of more demanding or high precision tasks. With the results obtained from figure 5.5, and comparing with the ones gathered in figures 5.2 and 5.3, one might call into question the results seen in this section. With this in mind, these tests, albeit acceptable, are conclusive enough to affirm that more research and study needs to be addressed by future works.

Furthermore, it is important to emphasize that the planned trajectory for the experimental tests illustrated in this chapter was not particularly simple, involving a complex reorientation of the end effector, which naturally highlights even more the inconsistencies of the deduced model and puts into evidence other unmodeled terms such as friction.

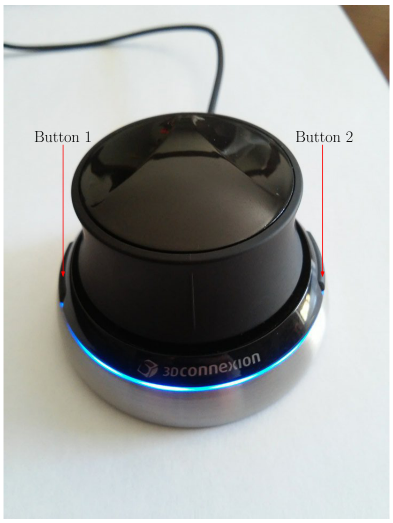
Despite all these premises, the system is able to reach its final pose with an overall acceptable steady state error.

Lastly, all the control gains shown in chapter 5 were manually tuned. Further improvements might be obtained by properly designing the system to have a proper damping behavior. Commonly, for these systems, it is wished to have the least amount of oscillatory movements and reach the steady state as fast as possible. This means designing the system to have a critically damped response.

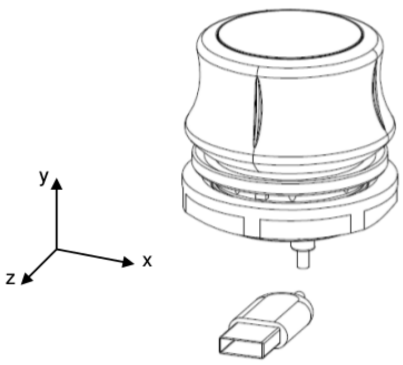
Chapter 6

Space Mouse Control

During the first stages of this work, before computed torque techniques were studied and applied to this robot, it was desired to design a more intuitive way of teleoperating the arm as an alternative to the difficult and complex joystick that is provided. For this reason, the SpaceNavigator® for Notebooks from 3Dconnexion[1] was used. This is a mouse commonly used professionally for CAD or 3D applications, allowing an easy and natural interaction.



(a) Space Mouse.



(b) Axes Orientation.

Figure 6.1: Product Specification.

The 3D mouse has 6 DOF, which makes it ideal to control a robotic manipulator like the JACO[2]. The mouse can be horizontally shifted in X and Z directions, vertically shifted in the Y direction, tilted around X and Z and twisted around Y axis, figure 6.1b, shows the axis orientation of the mouse [35]. Additionally, its sensitivity and other parameters can be adjusted with the associated software retrievable from the website of 3Dconnexion along

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

with C code examples in order to understand how to program the mouse and to develop applications with it.

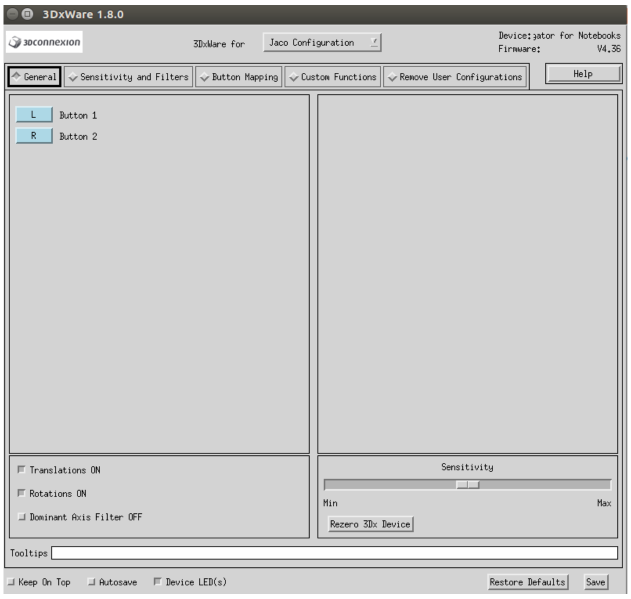


Figure 6.2: 3DxWare 1.8 (Unix Version) with JACO^2 configuration.

The software allows the switching of different mouse configuration modes depending on the program/window where the mouse is being used. The menu illustrated in figure 6.2 depicts the 3DxWare software with a manually set configuration to be used with the robot. To initialize the space mouse, it is mandatory to have the 3DxWare software opened.

The 3D mouse is also integrated with two configurable buttons (labeled in figure 6.1a), which can also be programmed, either with the 3DxWare software or through the C programming libraries developed for the mouse. This last option was pursued since it allowed the fully integration of the programming functions of the mouse into the developed C++ code used to control the arm with the Kinova API functions. All the work described in this chapter was developed under the Ubuntu 16.04 Linux distribution, but alternative software is also available for Windows platforms.

Initially, the objective was simply to control the 6 DOF of the arm with the mouse itself. The first step is defining which control option, from the ones that come with the manipulator, is best suited for this control method. For that decision to be made, it was necessary to know what kind of data the mouse outputted while manipulating it. After experimental tests with the mouse, it was found that for each DOF the output values were bound to the interval [350,-350], between the positive and negative maximum values, respectively. Having this, and knowing the different control types that the JACO^2 possesses (refer to appendix C), it was decided that the most straightforward control option would be cartesian velocity control.

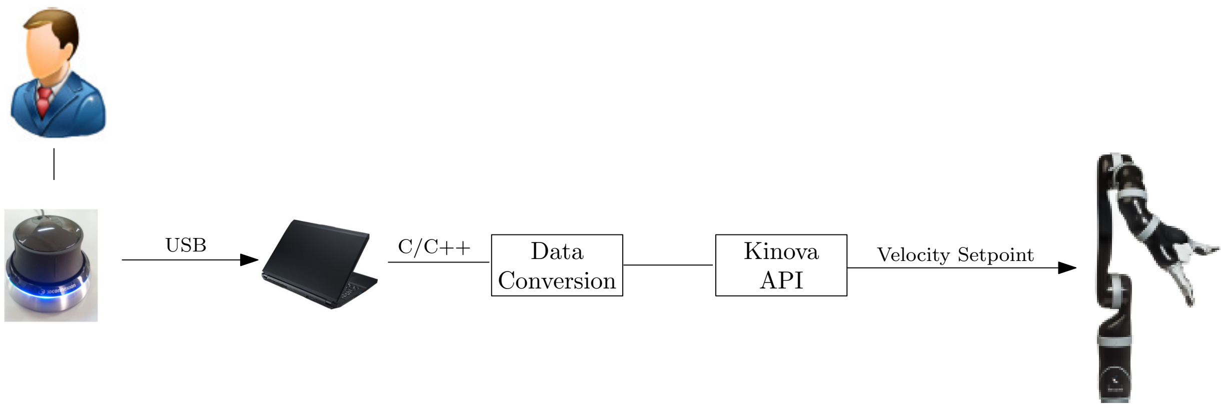


Figure 6.3: Simplified control of the JACO^2 with the SpaceNavigator®.

Figure 6.3 summarizes the proposed control scheme. The input commands are sent, with the real time manipulation of the mouse by the user, to the computer where they are processed. It is necessary to convert the data, scaling it to the range of values accepted by the robot, which is easily obtained by applying a rule of three,

$$v_{R_i} = \frac{S_i v_{R_m}}{S_m} \quad , \quad i = 1, \ldots, 3 \quad , \quad \mu_{R_i} = \frac{S_i \mu_{R_m}}{S_m} \quad , \quad i = 4, \ldots, 6 \tag{6.1}$$

where v_{R_i} is the translational velocity of the arm for each direction X, Y and Z, μ_{R_i} refers to the rotational velocity around each axis of the end effector, S_i is the space mouse input value for the i DOF, S_m is the maximum read value from the mouse, and lastly v_{R_m} and μ_{R_m} are the maximum defined values for the translational and rotational velocity for the JACO^2. These last two values are defined in accordance with the suggested manufacturer maximum values, which are $v_{R_m} = 0.2$ m/s and $\mu_{R_m} = 1$ rad/s. Additionally, it was noted that the default axes orientation of the mouse, depicted in figure 6.1b, could be made more intuitive for the user, if it matched the base reference frame of the JACO^2, which is where the translational motions are referred (recall figure 3.3). Consequently, the data sent by the mouse is readjusted according to the following relations,

$$\begin{cases} v_{R_x} = S_x \\ v_{R_y} = S_z \\ v_{R_z} = S_y \\ \mu_{R_x} = S_{rx} \\ \mu_{R_y} = S_{rz} \\ \mu_{R_z} = S_{ry} \end{cases} \tag{6.2}$$

With the coordinate axes of the mouse adjusted to the ones of the JACO^2, the control of the robotic arm was made more natural and easy for the user. With this resolved, Kinova API functions are then used to send the appropriate commands to the robot. A **TrajectoryPoint** structure is ready to host the newly mapped commands from the mouse, having its type of position set to **CARTESIAN_VELOCITY**. Finally, the velocity setpoints are sent with the function **MySendBasicTrajectory** with a sampling time of 10 ms.

The control of the robot is now be done in a natural fashion, however since the mouse integrates two configurable buttons, further features can be designed so that the user can maneuver the robot more precisely so that more complex tasks can be executed. Given these assumptions, a finite state machine (FSM) was developed allowing the change between the integrated operation modes in a quick manner.

As a first implementation, the space mouse is programmed with a limited set of operation

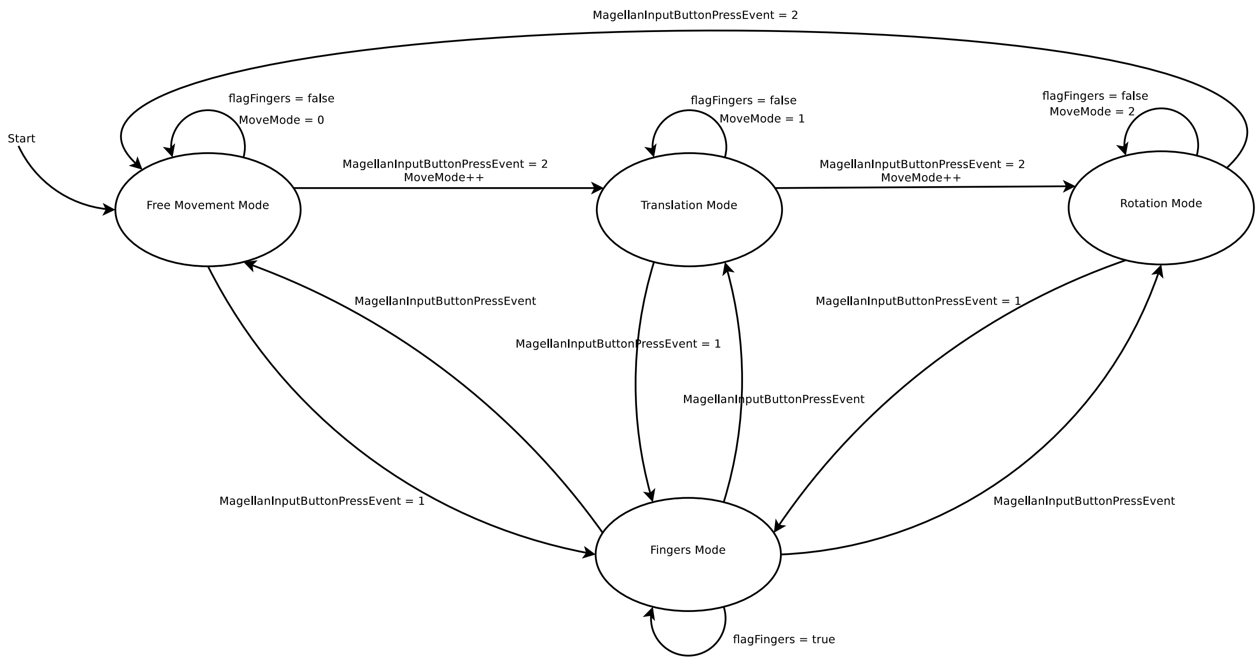

MagellanInputButtonPressEvent = 2
Start
flagFingers = false
MoveMode = 0
flagFingers = false
MoveMode = 1
flagFingers = false
MoveMode = 2
MagellanInputButtonPressEvent = 2
MoveMode++
MagellanInputButtonPressEvent = 2
MoveMode++
Free Movement Mode
Translation Mode
Rotation Mode
MagellanInputButtonPressEvent
MagellanInputButtonPressEvent = 1
MagellanInputButtonPressEvent = 1
MagellanInputButtonPressEvent
MagellanInputButtonPressEvent = 1
MagellanInputButtonPressEvent
Fingers Mode
flagFingers = true

Figure 6.4: Finite State Machine for Space Mouse Control of the JACO^2.

modes, although future work can certainly add more features for the user. The operation modes are then:

- Free Movement Mode - The 6 DOF of the JACO^2 are controlled simultaneously by the 6 DOF commands of the space mouse;
- Translation Mode - The robot is controlled exclusively in regards to its translation with respect to the base frame, which means that only the horizontal/vertical shift of the space mouse produces any motion;
- Rotation Mode - Similar to Translation Mode, but with only the tilting/twisting of the space mouse in order to induce end effector reorientations;
- Finger Mode - The end effector is composed of a hand where its fingers can be opened and closed. The space mouse can be twisted in order to produce that effect.

The FSM depicted in figure 6.4 portraits the proposed system. Both push buttons 1 and 2, are designed to grant the possibility of changing between these operation modes. Successive pushes of button 2, trigger the switching of free movement, translation and rotation modes, with the third press restarting the sequence. Concurrently, the user can press button 1 to access the finger mode, effectively immobilizing the robot and activating the opening or closing of the hand as desired. Specifically, if the space mouse is currently on Fingers Mode, the user can either press button 1 or 2 to return to the movement mode where it was before. The schematic shown above might give the hint that one can go from the rotation mode to the free movement mode, using the Fingers Mode. That is not the case, and it was only drawn this way, to simplify the scheme. The velocity of the fingers is also scaled appropriately in the same principle as entailed in equation 6.1, with the speed being set no higher than 6800 °/s, which is relatively lower than the maximum recommended value by the manufacturer set at 30 mm/s or 10800 °/s.

Overall, the proposed control scheme proved to be intuitive, and coupled with the control of the hand, greatly improved its usefulness.

Chapter 7

Conclusions

Globally, this work presented many challenges and difficulties, but it was certainly pleasurable to study this robotic arm. Although it can be said that the main objectives were achieved, it is of no lesser importance to refer that further improvements are recommended. The concluding remarks can be summarized in the following topics:

- As an introductory work, this thesis establishes the groundwork for future projects with this robot. Suffice to say that little to no similar research was found regarding specifically the topic of computed torque control with the JACO^2, being restrained mostly to EEG/EMG research areas [36] or clinical studies [37];

- The derived geometric and kinematic models produced acceptable results, which can suggest that their estimations were well assumed. Additionally, the dynamic model was evaluated and its estimation analyzed, with the results being promising. Future improvements to this model are certainly recommended as some terms were neglected like the friction terms. This can be crucial, by the fact that the JACO^2, is a gear driven manipulator with a considerable gear reduction ratio of 160:1, meaning that friction can play a considerable role when performing complex or high dynamic motions. This was visually attested throughout the work;

- The orientation control throughout chapter 5 was done under the application of the angle-axis representation methodology. Although it works fairly well, it has the limitation of becoming ill-defined at the representation of singularities. Future works might be better off with the unit quaternion representation, as it overcomes those drawbacks;

- The influence of the internal controllers present in the JACO^2 is still an unknown variable for the developer/researcher. Although this work tried to overcome this barrier, as seen in section 5.1, it is still assumed as not being completely cleared. Ideally, one would desire to have a complete and clear access to the internal works of the robot but, comprehensibly, Kinova itself explains this situation as a way of protecting their intellectual property;

- Although not particularly related to the main topic expressed in this document, the space mouse control brought a new perspective into possible future implementations with this portable and practical robotic arm, and future studies can certainly be done regarding computed torque control with the application of this 3D mouse as a task space reference generator.

7.1 Future Work

The basic foundation for the establishment of future implementations with this robot was accomplished. Numerous works can now spawn from the findings of this document:

- **Dynamic model identification using linear regression methods** - It is known that CAD-based dynamic models produce the least accurate results, so regression model techniques are suggested throughout the literature [15] [38] as a recommended method to better identify the robot dynamic model;

- **Analysis of more complex control architectures** - Control schemes including contact forces, impedance and admittance to infer the capabilities of the robot in a human-robot environment;

- **Evaluation of the space mouse as an integral part of a tele-echographic system** - Experimentally, its usefulness has been already proved. The intuitive and natural way it can be handled can potentially be used to replicate the motions of a conventional haptic device, but with the advantage of being much cheaper. Furthermore, the end effector of the JACO^2 is also removable, being possible to attach ecographic probes to it. This, coupled with the mouse might possibly be used to create a low cost tele-echographic system;

- **Tele-operation of the robotic arm, with pose delay analysis** - The JACO^2 possesses a ethernet port, which has been made available for all its users very recently. The work could revolve around analyzing the performance of the robot with different distances, inferring the inherent telecommunication delay.

References

[1] Sotiris Avgousti, Eftychios G. Christoforou, Andreas S. Panayides, Sotos Voskarides, Cyril Novales, Laurence Nouaille, Constantinos S. Pattichis, and Pierre Vieyres. Medical telerobotic systems: current status and future trends. *BioMedical Engineering OnLine*, 15(1):96, 2016.

[2] Jaques Marescaux and Francesco Rubino. Transcontinental robot-assisted remote telesurgery, feasibility and potential applications. In *Teleophthalmology*, pages 261–265. 2006.

[3] J Arata, H Takahashi, S Yasunaka, K Onda, K Tanaka, N Sugita, K Tanoue, K Konishi, S Ieiri, Y Fujino, Y Ueda, H Fujimoto, M Mitsuishi, and M Hashizume. Impact of network time-delay and force feedback on tele-surgery. *International Journal of Computer Assisted Radiology and Surgery*, 3(3-4):371–378, 2008.

[4] Peter F. Hokayem and Mark W. Spong. Bilateral teleoperation: An historical survey. *Automatica*, 42(12):2035–2057, 2006.

[5] W. Khalil and E. Dombre. Modeling, identification and control of robots. *Applied Mechanics Reviews*, 56:483, 2004.

[6] International Center of Excellence in Intelligent Robotics and Automation Research. `http://www.iceira.ntu.edu.tw/en/industry-development-and-applications/310-jaco`. Accessed: 2017-08-07.

[7] Kinova Robotics. JACO^2 User Guide. `http://www.kinovarobotics.com/wp-content/uploads/2017/08/JACO%C2%B2-User-Guide.pdf`, 2017.

[8] Laura V Herlant, Rachel M Holladay, and Siddhartha S Srinivasa. Assistive teleoperation of robot arms via automatic time-optimal mode switching. In *Human Robot Interaction*, 2016.

[9] Kinova Robotics. Kinova SDK. `http://www.kinovarobotics.com/wp-content/uploads/2017/08/Kinova-SDK-Development-Center-User-Guide.pdf`, 2017.

[10] Kinova Robotics. Kinova API Documentation, 2017.

[11] Kinova Robotics. USB-RS485 Documentation, 2015.

[12] Kinova Robotics. Kinova ROS Package. `https://github.com/Kinovarobotics`, 2017.

[13] John J. Craig. *Introduction to Robotics: Mechanics and Control (3rd Edition)*. Pearson, 2004.

[14] Kinova Robotics. Jaco^2 6 DOF Advanced Specification Guide Version 1.0.2, 2015.

[15] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. 2009.

[16] Soumya Bhattacharya, H. Hatwal, and A. Ghosh. An on-line parameter estimation scheme for generalized stewart platform type parallel manipulators. *Mechanism and Machine Theory*, 32(1):79–89, 1997.

[17] G. Lebret, K. Liu, and F. L. Lewis. Dynamic analysis and control of a stewart platform manipulator. *Journal of Robotic Systems*, 10(5):629–655, 1993.

[18] Philippe Guglielmetti and Roland Longchamp. A closed form inverse dynamics model of the delta parallel robot. *IFAC Proceedings Volumes*, 27(14):51–56, 1994.

[19] Bhaskar Dasgupta and Prasun Choudhury. General strategy based on the Newton-Euler approach for the dynamic formulation of parallel manipulators. *Mechanism and Machine Theory*, 34(6):801–824, 1999.

[20] Rui Cortesão, Brian Zenowich, Rui Araújo, and William Townsend. Robotic comanipulation with active impedance control. In *Proc. ASME/AFM 2009 World Conference on Innovative Virtual Reality (WINVR 2009)*, pages 129–135, Chalon-sur-Saône, France, February 25-26 2009. ASME.

[21] Rui Cortesão. Medical Robotics Course. University of Coimbra. 2014.

[22] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. Robot modeling and control, 2006.

[23] W. M. Silver. On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators. *The International Journal of Robotics Research*, 1(2):60–70, 1982.

[24] John M Hollerbach. A Recursive Lagrangian Formulation of Maniputator Dynamics and a Comparative Study of Dynamics Formulation Complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(11):730–736, 1980.

[25] Lorenzo Sciavicco, Bruno Siciliano, and Luigi Villani. On dynamic modelling of gear-driven rigid robot manipulators. *IFAC Proceedings Volumes*, 27(14):543–549, 1994.

[26] Peter Corke. *Robotics, Vision and Control - Fundamental Algorithms in MATLAB*. 2011.

[27] S. Menon. The Standard Control Library. `http://samirmenon.org/scl.html`, 2011.

[28] Cristóvão D. Sousa and Rui Cortesão. SageRobotics: open source framework for symbolic computation of robot models. In Sascha Ossowski and Paola Lecca, editors, *Proceedings of the 27th Symposium on Applied Computing*, pages 262–267. ACM, 2012.

[29] Wisama Khalil, Aravindkumar Vijayalingam, Bogdan Khomutenko, Izzatbek Mukhanov, Philippe Lemoine, and Gael Ecorchard. OpenSYMORO: An open-source software package for symbolic modelling of robots. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pages 1206–1211, 2014.

[30] Bruno Siciliano and Luigi Villani. *Robot force control*, volume 53. 2013.

[31] Cristóvão Sousa, Rui Cortesão, and Luís Santos. Computed torque posture control for robotic-assisted tele-echography. In *18th Mediterranean Conference on Control and Automation, MED'10 - Conference Proceedings*, pages 1561–1566, 2010.

[32] Simon L Altmann. *Rotations, Quaternions, and Double Groups*, volume 3. 1986.

[33] L. Santos and R. Cortesão. Joint space torque control with task space posture reference for robotic-assisted tele-echography. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 126–131, Sept 2012.

[34] Luís Santos and Rui Cortesão. A dynamically consistent hierarchical control architecture for robotic-assisted tele-echography. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1990–1996, 2014.

[35] 3Dconnexion. Product Specification SpaceMouse® Module USB, 2014.

[36] Laurent Bougrain, Olivier Rochel, Octave Boussaton, and Lionel Havet. From the decoding of cortical activities to the control of a JACO robotic arm: a whole processing chain. *Control Architecture of Robots (CAR)*, pages 1–7, 2012.

[37] Veronique Maheu, Philippe S. Archambault, Julie Frappier, and François Routhier. Evaluation of the JACO robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities. In *IEEE International Conference on Rehabilitation Robotics*, 2011.

[38] Oussama Khatib (Eds.) Bruno Siciliano. *Springer, Handbook on Robotics*, volume 25. 2014.

Appendix A

Angular Control Example

```
#include <iostream>
#include <dlfcn.h>
#include <vector>
#include "Lib_Examples/Kinova.API.CommLayerUbuntu.h"
#include "Lib_Examples/KinovaTypes.h"
#include <stdio.h>
#include <unistd.h>

using namespace std;

int main()
{
        int result;

        AngularPosition currentCommand;

        //Handle for the library's command layer.
        void * commandLayer_handle;

        //Function pointers to the functions we need
        int (*MyInitAPI)();
        int (*MyCloseAPI)();
        int (*MySendBasicTrajectory)(TrajectoryPoint command);
        int (*MyGetDevices)(KinovaDevice devices[MAX_KINOVA_DEVICE], int &result);
        int (*MySetActiveDevice)(KinovaDevice device);
        int (*MyMoveHome)();
        int (*MyInitFingers)();
        int (*MyGetAngularCommand)(AngularPosition &);

        //We load the library
        commandLayer_handle = dlopen("Kinova.API.USBCommandLayerUbuntu.so",RTLD_NOW|
RTLD_GLOBAL);

        //We load the functions from the library
        MyInitAPI = (int (*)()) dlsym(commandLayer_handle,"InitAPI");
        MyCloseAPI = (int (*)()) dlsym(commandLayer_handle,"CloseAPI");
        MyMoveHome = (int (*)()) dlsym(commandLayer_handle,"MoveHome");
        MyInitFingers = (int (*)()) dlsym(commandLayer_handle,"InitFingers");
        MyGetDevices = (int (*)(KinovaDevice devices[MAX_KINOVA_DEVICE], int
&result)) dlsym(commandLayer_handle,"GetDevices");
        MySetActiveDevice = (int (*)(KinovaDevice devices))
dlsym(commandLayer_handle,"SetActiveDevice");
        MySendBasicTrajectory = (int (*)(TrajectoryPoint))
dlsym(commandLayer_handle,"SendBasicTrajectory");
        MyGetAngularCommand = (int (*)(AngularPosition &))
dlsym(commandLayer_handle,"GetAngularCommand");

        if((MyInitAPI == NULL) || (MyCloseAPI == NULL) || (MySendBasicTrajectory ==
NULL) ||
           (MySendBasicTrajectory == NULL) || (MyMoveHome == NULL) || (MyInitFingers
== NULL))
        {
                cout << "* * *  E R R O R   D U R I N G   I N I T I A L I Z A T I O
N  * * *" << endl;
        }
        else
        {
                cout << "I N I T I A L I Z A T I O N   C O M P L E T E D" << endl <<
endl;

                result = (*MyInitAPI)();

                cout << "Initialization's result :" << result << endl;

                KinovaDevice list[MAX_KINOVA_DEVICE];
```

```
		int devicesCount = MyGetDevices(list, result);

		for(int i = 0; i < devicesCount; i++)
		{
			cout << "Found a robot on the USB bus (" <<
list[i].SerialNumber << ")" << endl;

			//Setting the current device as the active device.
			MySetActiveDevice(list[i]);

			cout << "Send the robot to HOME position" << endl;
			MyMoveHome();

			cout << "Initializing the fingers" << endl;
			MyInitFingers();

			TrajectoryPoint pointToSend;
			pointToSend.InitStruct();

			//We specify that this point will be used an angular(joint
by joint) velocity vector.
			pointToSend.Position.Type = ANGULAR_VELOCITY;

			pointToSend.Position.Actuators.Actuator1 = 0;
			pointToSend.Position.Actuators.Actuator2 = 0;
			pointToSend.Position.Actuators.Actuator3 = 0;
			pointToSend.Position.Actuators.Actuator4 = 0;
			pointToSend.Position.Actuators.Actuator5 = 0;
			pointToSend.Position.Actuators.Actuator6 = 48; //joint 6 at
48 degrees per second.

			pointToSend.Position.Fingers.Finger1 = 0;
			pointToSend.Position.Fingers.Finger2 = 0;
			pointToSend.Position.Fingers.Finger3 = 0;

			for(int i = 0; i < 300; i++)
			{
				//We send the velocity vector every 5 ms as long as
we want the robot to move along that vector.
				MySendBasicTrajectory(pointToSend);
				usleep(5000);
			}

			pointToSend.Position.Actuators.Actuator6 = -20; //joint 6 at
-20 degrees per second.

			for(int i = 0; i < 300; i++)
			{
				//We send the velocity vector every 5 ms as long as
we want the robot to move along that vector.
				MySendBasicTrajectory(pointToSend);
				usleep(5000);
			}

			cout << "Send the robot to HOME position" << endl;
			MyMoveHome();

			//We specify that this point will be an angular(joint by
joint) position.
			pointToSend.Position.Type = ANGULAR_POSITION;

			//We get the actual angular command of the robot.
			MyGetAngularCommand(currentCommand);

			pointToSend.Position.Actuators.Actuator1 =
currentCommand.Actuators.Actuator1 + 30;
```

```
                        pointToSend.Position.Actuators.Actuator2 =
currentCommand.Actuators.Actuator2;
                        pointToSend.Position.Actuators.Actuator3 =
currentCommand.Actuators.Actuator3;
                        pointToSend.Position.Actuators.Actuator4 =
currentCommand.Actuators.Actuator4;
                        pointToSend.Position.Actuators.Actuator5 =
currentCommand.Actuators.Actuator5;
                        pointToSend.Position.Actuators.Actuator6 =
currentCommand.Actuators.Actuator6;

                        cout << "*********************************" << endl;
                        cout << "Sending the first point to the robot." << endl;
                        MySendBasicTrajectory(pointToSend);

                        pointToSend.Position.Actuators.Actuator1 =
currentCommand.Actuators.Actuator1 - 60;
                        cout << "Sending the second point to the robot." << endl;
                        MySendBasicTrajectory(pointToSend);

                        pointToSend.Position.Actuators.Actuator1 =
currentCommand.Actuators.Actuator1;
                        cout << "Sending the third point to the robot." << endl;
                        MySendBasicTrajectory(pointToSend);

                        cout << "*********************************" << endl << endl
<< endl;
                }

                cout << endl << "WARNING: Your robot is now set to angular control.
If you use the joystick, it will be a joint by joint movement." << endl;
                cout << endl << "C L O S I N G   A P I" << endl;
                result = (*MyCloseAPI)();
        }

        dlclose(commandLayer_handle);

        return 0;
}
```

Appendix B

Dynamic Parameters

		Augmented Link i					
		1	2	3	4	5	6
Dynamic Parameters	m_{aug}	0.767664	0.99571300239	0.79668391394	0.41737	0.41737	1.07541
	r_{aug_x}	-0.000063842137053	-0.322594959701929	0.000026085587226	-0.035652445675323	-0.022530478339697	-0.004633704633582
	r_{aug_y}	-0.095871156339924	0.000022679833280	-0.001675695131239	-0.000016695497999	0.000016695497999	0.000013801061921
	r_{aug_z}	0.002424491386880	0.014126962029412	-0.017063936999105	0.019913097690161	0.023535326664142	-0.106251253568406
	$L_{t_{xx}}$	0.002231083822876	0.004162524943115	0.002861254222538	0.708476728234e-03	0.827477604255e-03	0.004833755945304
	$L_{t_{xy}}$	0.000006815637176	-0.000000552736891	0.000000402773293	0.008271643300e-03	0.008281078147e-03	0.000004331179432
	$L_{t_{xz}}$	-0.000019787600863	-0.001487468585390	0.000000566406981	0.112833961462e-03	-0.101690165377e-03	0.000361596139440
	$L_{t_{yy}}$	0.000620733840723	0.025495429281017	0.002738656386387	0.740496291632e-03	0.852081770282e-03	0.004841503493061
	$L_{t_{yz}}$	-0.000306288710418	-0.000000277011102	-0.000344659168888	0.000494273498e-03	-0.000054222717e-03	-0.000002455131406
	$L_{t_{zz}}$	0.002398007441187	0.021736935450151	0.000351242752544	0.178193295188e-03	0.170777897817e-03	0.000199821519482

Table B.1: Dynamic Parameters of the JACO^2 with masses in kilograms, COMs in meters and inertia tensor in kilograms per meter squared.

Appendix C

Control Types of the JACO^2

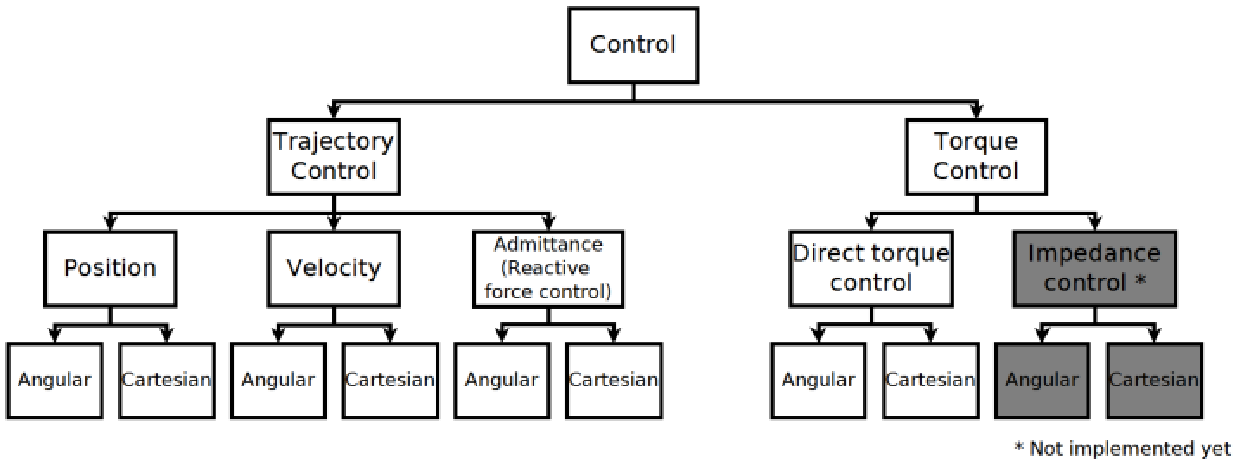


Figure C.1: Control Types.

Appendix D

Initial and Final Poses

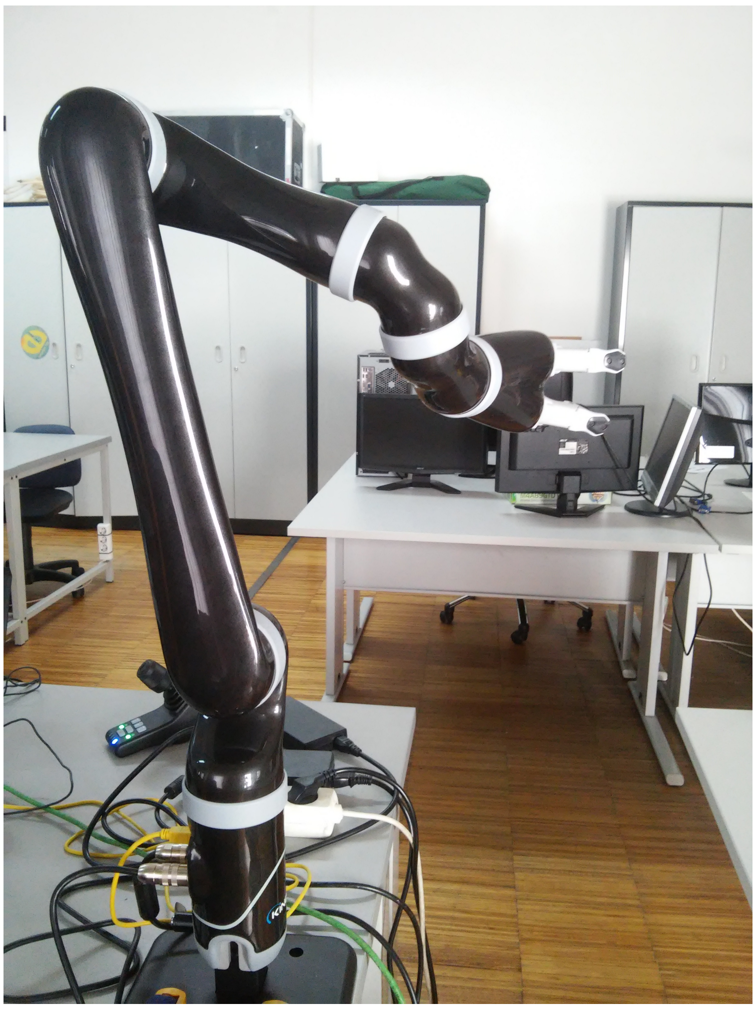


Figure D.1: Initial Position and Orientation.

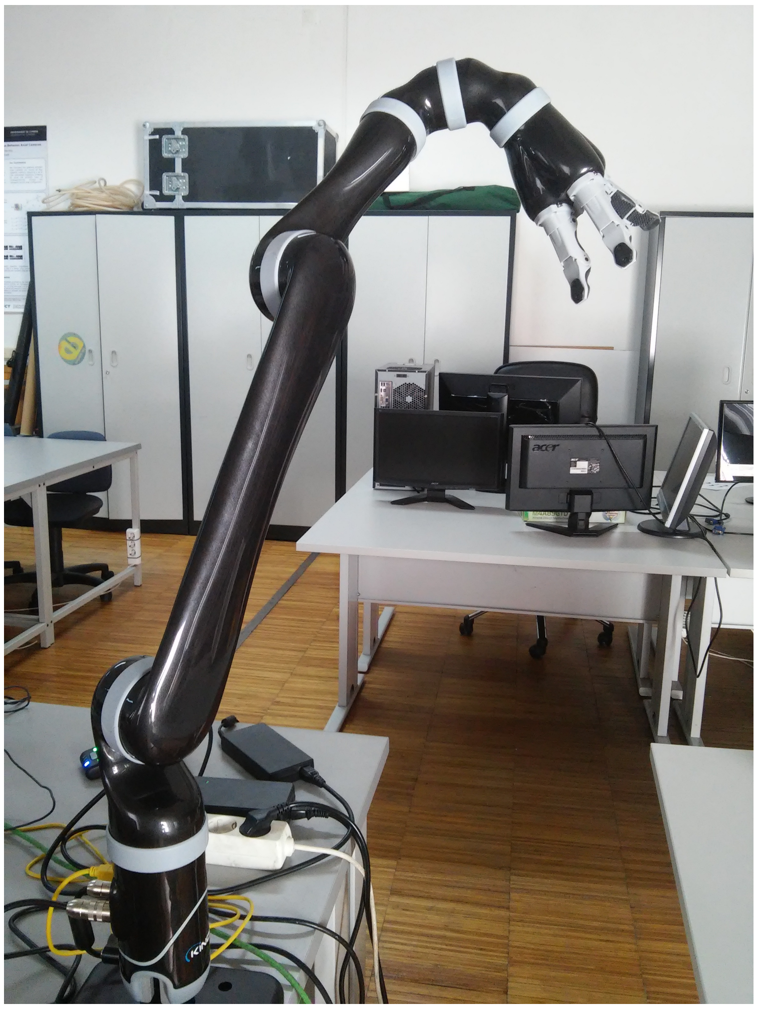


Figure D.2: Final Position and Orientation.

Appendix E

JACO^2 Product Specification

KINOVA ROBOTICS

Specifications

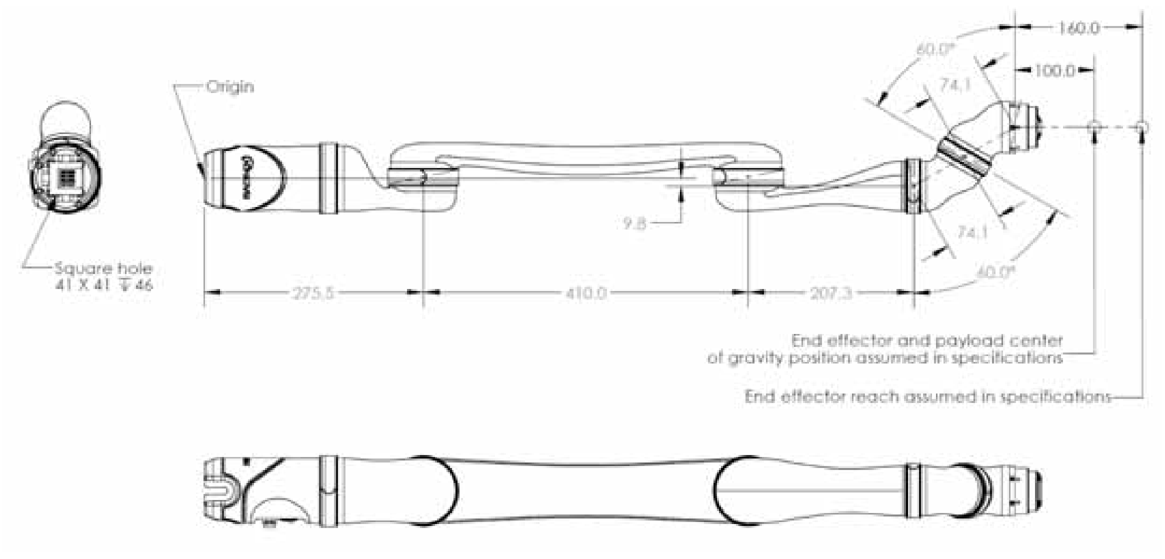
JACO^2

6 DOF

Tech Specs

JACO^2
6 DOF

Version 1.1 – April 2017


Origin
Square hole 41 X 41 ↧46
160.0
100.0
60.0°
74.1
9.8
275.5
410.0
207.3
74.1
40.0°
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)	500 Hz
CPU		360 MHz
SDK	APIs	High and low level
	Compatibility	Windows, Linux Ubuntu & ROS
	Port	USB 2.0, Ethernet
	Programming languages	C++
Control		Force, cartesian & angular

SPECIFICATIONS

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

Appendix F

Actuators Product Specification

KINOVA ROBOTICS

Specifications

ACTUATORS

K-75+ K-58

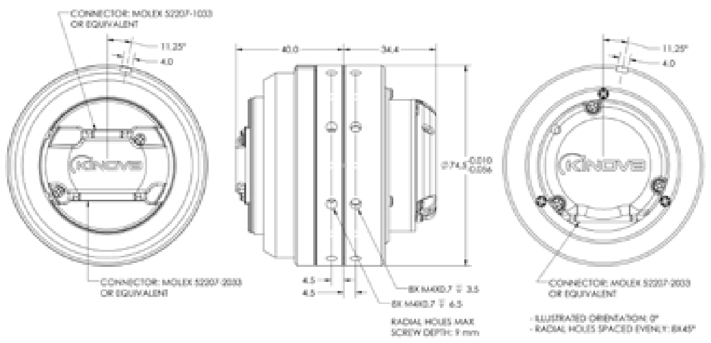
Tech Specs

ACTUATORS

Version 1.1 – May 2017

K-75+

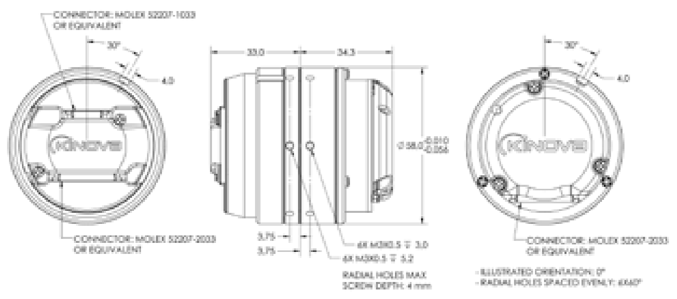
Ø74.5 mm, 12.0 Nm nominal, 37 Nm peak
Brushless DC motor, ratio 160 Harmonic Drive™



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

K-58

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



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

GEARED MOTOR (WITH 24V SUPPLY)

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

SENSORS

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

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

MECHANICAL

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

THERMAL

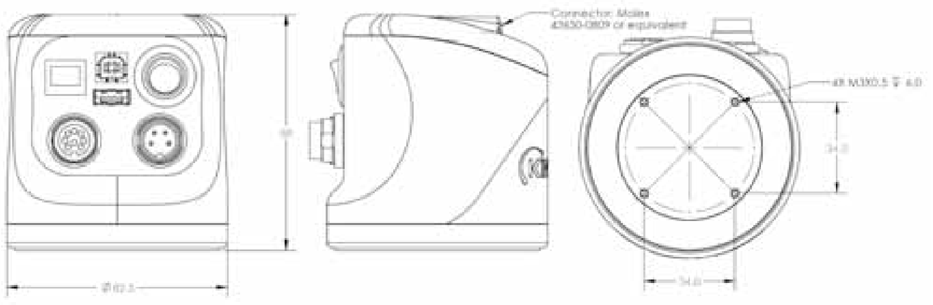
Operating temperature range	-10 °C to 40 °C
Max frame temperature *(overheat protection triggered)*	75 °C

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

ELECTRONIC

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

CONTROLLER



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

REFERENCE

A

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

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

B

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

C

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

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

D

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

E

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

M

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

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

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

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

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

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

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

N

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

Nominal speed:
The maximum speed under Nominal torque load.

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

O

Operating temperature range:
Actuator safe operating temperature range.

P

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

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

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

T

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

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

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

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

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

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

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

W

Weight:
The weight of the actuator module.

Appendix G

DSP Product Specification

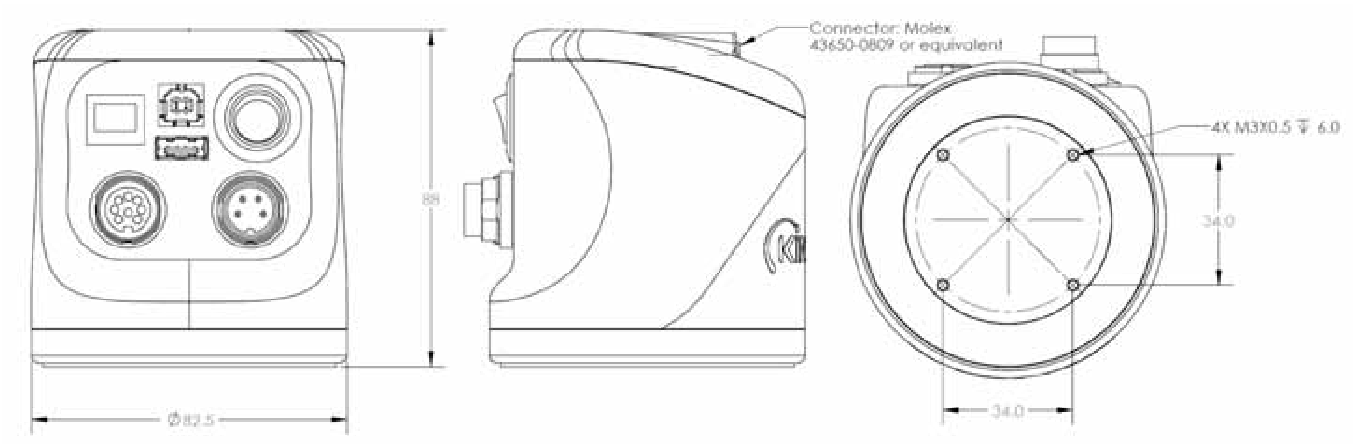
Specifications

CONTROLLER

Tech Specs

CONTROLLER

Version 1.1 – April 2017



SPECIFICATIONS

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